Huawei HCIA Certification Training

HCIA-openEuler

openEuler System Engineer

Learning Guide

ISSUE: 1.0

HUAWEI TECHNOLOGIES CO., LTD

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base Bantian, Longgang Shenzhen 518129 People's Republic of China

Website: https://e.huawei.com

Huawei Certification System

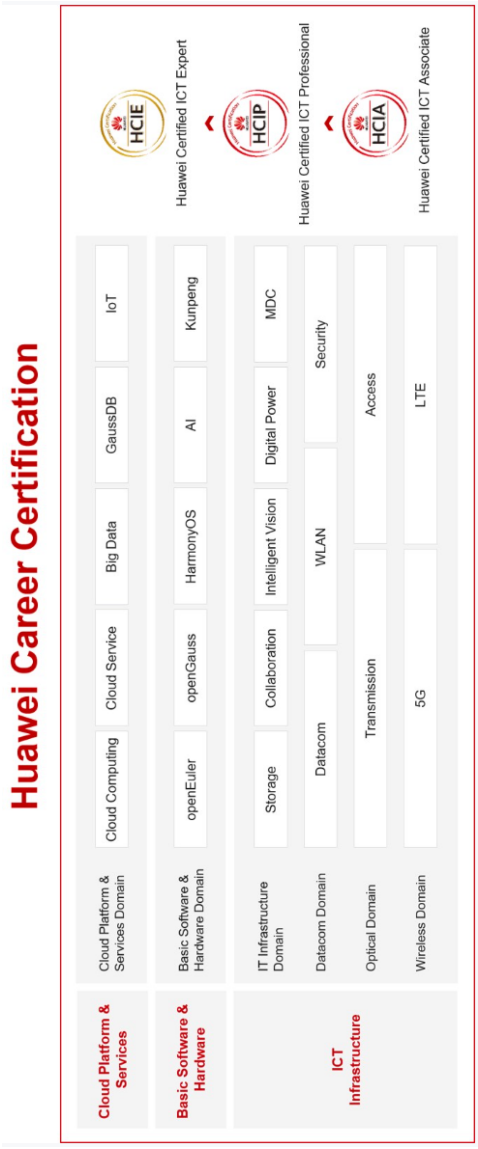
Huawei Certification is an integral part of the company's Platform + Ecosystem strategy. It supports the development of ICT infrastructure that features Cloud-Pipe-Device synergy. Our certification is always evolving to reflect the latest trends in ICT development.

Huawei Certification consists of three categories: ICT Infrastructure Certification, Basic Software & Hardware Certification, and Cloud Platform & Services Certification, making it the most extensive technical certification program in the industry.

Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

Our programs cover all ICT fields and follow the industry's trend of ICT convergence. With our leading talent development system and certification standards, we are committed to fostering new digital ICT talent and building a sound ICT talent ecosystem.

HCIA-openEuler is intended for frontline engineers at Huawei regional or representative offices, as well as other personnel who want to expand their knowledge of Linux technologies. The HCIA-openEuler certification includes the introduction to the openEuler OS, software installation, and basic operations, file processing and text editing, user and permission management, logical volume management, and other techniques and best practices aimed at getting the most out of openEuler.


Huawei Career Certification
Cloud Platform & Services
Basic Software & Hardware
ICT Infrastructure
Cloud Platform & Services Domain
Basic Software & Hardware Domain
IT Infrastructure Domain
Datacom Domain
Optical Domain
Wireless Domain
Cloud Computing
Cloud Service
Big Data
GaussDB
IoT
openEuler
openGauss
HarmonyOS
AI
Kunpeng
Storage
Collaboration
Intelligent Vision
Digital Power
MDC
Datacom
WLAN
Security
Transmission
Access
5G
LTE
HCIE
Huawei Certified ICT Expert
HCIP
Huawei Certified ICT Professional
HCIA
Huawei Certified ICT Associate

Forword

As the most widely used operating system (OS), Linux has been developed for nearly 30 years and has become the most basic platform in the information technology (IT) industry. openEuler is an open source OS that runs on Linux Kernel as well as an open and innovative platform. It not only supports multiple processor architectures including Kunpeng, but also explores the potentials of the OS and system architecture. The openEuler community will eventually become an open source ecosystem that leads technological innovation. Open source is an approach to construct the industry ecosystem. More and more companies around the world use open source to promote the construction of the industry chain ecosystem and even influence industry development, building an ecosystem chain from open source communities to open source-based enterprise-grade products and services.

Open source is a collaborative innovation mode. This mode greatly accelerates the speed of software development and shortens the time for formulating industry standards. In addition, an open and collaborative open source environment is more likely to inspire innovations and creativity. Innovations emerge one after another in open source communities.

Open source is a way of cultural exchange. Through open source, the wisdom of the world can be gathered to jointly develop and evolve large-scale software systems in different corners of the world. This has also, to a large extent, deepened communication and understanding around the world. Communication is the key to building a better future around the world.

openEuler is a free open source Linux distribution. In the openEuler community, global developers join hands to build an open, diversified, and architecture-inclusive software innovation ecosystem. In addition, openEuler is an innovative platform that encourages anyone to propose new ideas, explore new opportunities, and practice new solutions.

Preface

An OS is the heart and soul of a computer system. Learning and understanding OS principles are critical for those majoring in computer-related fields and for computer application developers.

This document describes the concepts and operations related to the openEuler OS, helping you prepare for HCIA-openEuler certification. HCIA-openEuler V1.0 is the first career certification launched by Huawei for the openEuler open source OS. It is also the only career certification for community-supported open source OSs in China.

The openEuler OS career certification is intended for openEuler OS O&M engineers. There are three levels of certification available: Associate, Professional, and Expert.

HCIA-openEuler is intended to develop engineers' basic operation capabilities on openEuler. Those who pass this certification will be able to install and use openEuler, as well as apply cutting-edge Linux knowledge.

This document focuses on HCIA-openEuler and combines theories with practices. You are advised to use it with HCIA-openEuler Lab Guide (PC Edition) to better master operations on openEuler.

As the old saying goes "A journey of a thousand miles begins with a single step", let's get started.

Contents

Forword **4**
Preface **5**
1 Getting Started with openEuler **9**
1.1 Introduction to the Linux OS 9
1.1.1 What Is an OS? 9
1.1.2 Common OSs 11
1.1.3 Origin and Development of Linux 12
1.2 Installing the openEuler OS 14
1.2.1 Introduction to openEuler 14
1.2.2 Installing the openEuler OS 15
1.3 Using the openEuler OS 20
1.4 Quiz 21
2 CLI Basics **22**
2.1 Basic Knowledge of Linux Commands 22
2.2 Basic Linux Commands 23
2.2.2 Login Commands 24
2.2.3 Power Management Commands 26
2.2.4 File Management Commands 27
2.2.5 Help Commands 39
2.3 Quiz 41
3 Text Editors and Text Processing **42**
3.1 Common Linux Text Editors 42
3.2 Text Editors and Text Processing 44
3.2.1 Vim Text Editor 44
3.2.2 Text Processing 49
3.3 Quiz 57
4 User and Permission Management **58**
4.1 User and User Group Management 58
4.1.1 User Management 58
4.1.2 User Groups Management 61
4.2 File Permission Management 63
4.2.1 Overview 63
4.2.2 File Permission Management 64
4.2.3 Special File Permissions 68

4.2.4 File ACL Permission 69
4.3 Other Permission Management 73
4.3.1 Temporary Privilege Escalation 73
4.4 Quiz 75
5 Software Installation and Service Management 76
5.1 Software Package Management 76
5.2 RPM Software Package Overview 76
5.2.1 Common RPM Options 76
5.3 DNF Software Management 77
5.3.1 DNF Configuration File 78
5.3.2 Current Configuration Display 81
5.3.3 Software Package Management 81
5.4 Installation Through Source Code 86
5.4.1 Procedure for Installation Through Source Code 87
5.5 Service Management 87
5.5.1 Service Management Overview 87
5.5.2 Commonly-Used Commands 88
5.5.3 Basic System Service Management 90
5.6 Quiz 92
6 File System and Storage Management 93
6.1 Linux Storage Basics 93
6.1.2 Overview of Drives and Partitions 94
6.1.3 Physical Partitions 95
6.2 LV Layer Management 96
6.2.1 Installing and Verifying LVM 96
6.2.2 Creating Partitions 97
6.2.3 Managing Physical Volumes 97
6.2.4 **Managing Volume Groups** 99
6.2.5 **Managing LVs** 101
6.2.6 Creating and Mounting File Systems 104
6.3 Quiz 106
7 System Management 107
7.1 Task Management 107
7.1.1 Task Executed Once: at 107
7.1.2 Periodic Task: crond 108
7.2 Network Management 109
7.2.1 OSI Protocol Stack (Extended) 109
7.2.2 IPv4 Address 114

7.2.3 ifcfg and iproute 114
7.2.4 Overview of NetworkManager 114
7.2.5 Connecting to a Network Using nmcli 117
7.2.6 Configuring Static Routes Using nmcli 117
7.2.7 Network Interface Configuration File 118
7.2.8 Host Name Management 119
7.3 Process Management 120
7.3.1 Process Overview 120
7.3.2 Process Monitoring 122
7.4 Quiz 127
8 Shell Scripts 128
8.1 Shell Basics 128
8.1.1 Shell Overview 128
8.1.2 Common Shells 128
8.1.3 Differences Between Shell and Compiled Languages 129
8.1.4 When to Use Shell 130
8.2 Shell Programming Basics 131
8.2.1 Shell Script Specifications and Running 131
8.2.2 Shell Variables 131
8.2.3 Special Shell Variables and Location Parameters 133
8.2.4 Shell Replacement 134
8.2.5 Shell Operators 136
8.2.6 Shell Character Strings 144
8.2.7 If-Else Statement 145
8.2.8 Case Statement 146
8.2.9 For Loop 146
8.2.10 While Loop 147
8.2.11 Script Instances 148
8.3 Quiz 149
9 Samba File Sharing Server Management 156
9.1 Samba Overview 156
9.1.1 Basic Configuration of the Samba Server 156
9.1.2 Samba Server Configuration 157
9.2 Samba File Sharing Server Example 158
9.3 Quiz 161
10 Reference 162

1 Getting Started with openEuler

1.1 Introduction to the Linux OS

1.1.1 What Is an OS?

An OS is a set of programs that controls and manages the hardware and software resources of an entire computer system, providing a convenient interface and environment for users and other software. As shown in Figure 1-1, an OS lies between the hardware and applications. As the software layer closest to the underlying hardware, the OS provides an environment where to run applications and hardware resources can collaborate efficiently to accomplish specific computing tasks. Users can directly interact with the computer through the user interface.

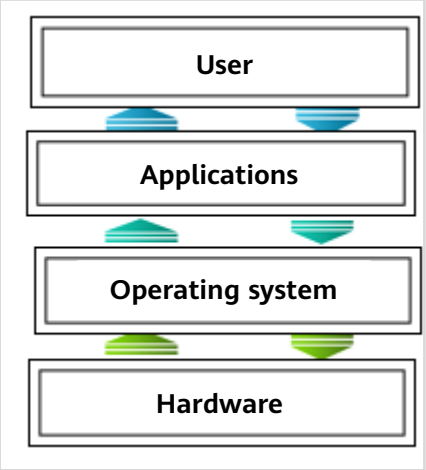


Figure 1-1 OS architecture

1.1.1.1 Key Functions of OSs

Process management

Modern OSs provide multiprogramming technology that executes many programs concurrently and shares system resources. After the emergence of multiprogramming systems, the concept of "process" was refined to describe the dynamic characteristics of concurrent programs and control their activity status. A process is the basic unit that describes the program execution and can be used to share resources. An OS allocates hardware resources to process and maintain the state of each process, to accomplish concurrent computer tasks.

To make different processes share hardware resources, the OS must manage hardware resources. The OS provides service interfaces for processes through system calls to restrict processes from directly performing operations on hardware resources. To perform restricted operations, the process needs to invoke these system call interfaces to send service requests to the OS and hand over the CPU control right to the OS. After receiving the request, the OS invokes the corresponding processing program to complete the service requested.

To execute concurrently, processes need to share CPU in time division multiplexing (TDM) mode, which means the OS should support process swapping, stopping the on-going process that occupies the CPU for a period of time and selecting the next process. To prevent malicious CPU occupancy, the OS can use the timer interrupt to stop the current process at regular intervals for process swapping.

Memory management

Programs and code in the system need to be loaded to the memory before being scheduled and executed by the CPU. Therefore, when multiple processes are executed concurrently, all of the processes need to be loaded to the memory. As a result, the memory becomes a key factor that affects the OS performance. The OS memory management mainly solves the memory sharing problem of concurrent processes. Technologies such as virtual memory, paging scheme, and physical memory expansion through external storage are used to improve the memory utilization and addressing efficiency.

File system management

Although the memory allows the system to access data quickly, due to the limited memory capacity, if an unexpected power outage occurs, the data in the memory may be lost. Therefore, computers usually use external storage such as disks to store data persistently. For ease of use, the OS abstracts external storage such as disks into files and directories, and manages them through the file system. There are multiple physical file systems available for the OS, including Ext4, FAT32, and NTFS. Users or applications can perform I/O operations and input and output data to disks through the file system for persistent data storage, and achieving management and protection of file storage space, directories, and file read/write

Hardware driver management

As an interface for users to operate underlying hardware, the OS is responsible for managing various I/O devices. Through the loadable module function, the OS can edit a driver into a module to identify underlying hardware, so that an application can use the I/O device. Therefore, the OS provides development interfaces for hardware vendors to develop their drivers. After obtaining hardware resources, the OS allocates devices, controls devices, and manages I/O buffers.

User interface

The OS provides an interactive environment for ease of use. Generally, interfaces for interaction between users and the OS are classified into command interfaces and APIs.

- Command interfaces

A user sends a series of instructions to the computer through an input device or in a job, so that the computer performs the task according to the instructions. There are two types of common command interfaces:

- Command line interface (CLI), which is a character-based user interface. The keyboard is used as the input tool to enter commands, options, and parameters to execute programs, achieving high efficiency. For example, the MS-DOS system.
- Graphical user interface (GUI), which presents all elements as graphical. The mouse is used as the main input tool, and buttons, menus, and dialog boxes are used for interaction, enhancing ease of use. For example, the Windows system.

- APIs

An API is mainly made up of system calls, through which applications can access resources in the system and obtain services provided by the kernel. Each system call corresponds to a subprogram that is performed in the kernel for a specific function.

1.1.2 Common OSs

The previous section introduces different OSs with different interactive interfaces, including CLI (for example the DOS system) and GUI (for example the Windows system). Due to the smooth user-computer interaction, Windows has become the most popular desktop OS for personal computers. In addition to Windows, there are other mainstream OSs, as shown in Figure 1-2.

OS	Application scenario
Linux	Enterprise servers. A free of charge, Unix-like, open source system, providing high stability and performance.
Unix	Enterprise servers. Provides high stability and performance.
Windows	PC. Focuses on ease of use.
Mac OS	PC. Focuses on ease of use and user experience.
Android	Mobile devices. Focuses on ease of use and user experience.
IOS	Mobile devices. Focuses on ease of use and user experience.
DOS	PC. Usually installed with Windows.

Figure 1-2 Common OSs

The Linux system is widely used in enterprise services, providing a stable and high-performance execution environment for developers. Linux is a Unix-like, open source OS designed by thousands of programmers around the world. Linux and its source code are free of charge that users can modify and distribute the system as required.

In addition, computers continue to get smaller, and mobile devices, such as smartphones and smart watches, have become part of our daily lives. iOS and Android are currently the most well-known OSs dedicated to mobile devices. iOS is a Unix-like, closed-source commercial OS released by Apple in 2007. In September 2008, Google released the Android source code under the Apache License. Android is an OS based on the Linux kernel, and designed primarily for touchscreen mobile devices.

1.1.3 Origin and Development of Linux

Unix development history

Back in the 1960s, computers were not widely used, and only a small number of individuals could even access them. At that time, computer systems were used for batch processing, whereby multiple tasks were submitted to the computer at once, which required users to wait for the results. No other interactions were supported during this process, resulting in a waste of computer resources. In addition, only a few terminals were allowed to access the host for input and output operations. In 1965, Bell Labs, MIT, and GE worked together to change this situation by developing a time-sharing multitasking system. Simply put, they realized the vision of multiple people using computers at the same time. The resulting computer system was named Multics, which is short for Multiplexed Information and Computing Service. However, in 1969, dissatisfied with the project's progress, Bell Labs eventually pulled out of the project.

Ken Thompson, a programmer in the Labs' computing research department, was in the process of developing a game called Space Travel while working on Multics. When Bell Labs pulled out, Thompson was no longer able to use the Multics environment. Hence, he wrote a small OS that was capable of running Space Travel. However, after showing off to peers, they were far more interested in his new system than in his game. The system was named as Unic, for UNiplexed Information and Computing Service, and as a pun on emasculated Multics. Nowadays, we call it "Unix" for short. 1970 is now the year of Unix. As a result, computer system times are calculated from the year 1970. In 1971, Thompson and Ritchie jointly invented the C language. After that, they rewrote UNIX in the C language and officially released it in 1974.

UNIX is a multiuser, multitasking OS initiated by AT&T Bell Labs in 1969. It was initially free of charge, and its security, efficiency, and portability made it ideal for servers. After the breakup of AT&T in 1982, the Unix became commercial and was widely used by many high-end applications in large-scale data centers. Some larger hardware companies have developed different versions of Unix based on their own computer systems, including early System V, UNIX 4.x BSD (Berkeley Software Distribution), FreeBSD, openBSD, Solaris of SUN, AIX of IBM, MINIX for teaching, and MAC OS X dedicated to Apple.

GNU and open source

To break the restrictions of the UNIX closed ecosystem, Richard M. Stallman initiated an international source code openness project called GNU in 1983 and founded the Free Software Foundation (FSF). FSF achieves freedom in four aspects: (1) freedom to run programs for any purpose; (2) freedom to learn and modify source code; (3) freedom to redistribute programs; and (4) freedom to create derivatives. The "free" emphasized by GNU is not only free of charge, but also freedom that you can access and modify software as required. Although the source code is free to use, you should pay for the value-added services such as software consulting, after-sales service, and software upgrade. The creation of GNU promotes the development of UNIX and Linux OSs.

The goal of the GNU project is to create a free, open source OS, which was not available at that time. Stallman first developed small programs running on Unix, such as Emacs, GNU C Compiler (GCC), and Bash shell.

By 1985, Stallman released the General Public License (GPL) to prevent free GNU software from becoming proprietary. It applies two measures to protect the rights of programmers: (1) copyright protection for software; (2) license for programmers, which gives them the legal permission to copy, distribute, and modify the software. In terms of copying and distribution, the GPL states that "You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee. However, if a software product uses the GPL products ("use" refers to class library reference, modified code, or derivative code), the software must use the GPL protocol, which must be open source and free of charge."

Currently, in addition to GPL, the commonly used open source licenses mainly include Mulan, LGPL, and BSD.

Mulan is China's first open source license. It involves five major aspects: copyright licensing, patent licensing, non-trademark licensing, distribution restriction, and disclaimer and liability restriction. It grants "every contributor" a permanent, global, free, non-exclusive, and irrevocable copyright license. Under the license, you can copy, modify, and distribute your "contributions" regardless of whether they are modified or not.

LGPL is an open source protocol designed for the use of class libraries. Different from GPL, LGPL allows commercial software to use the LGPL class library in link mode without requiring open-sourcing commercial software code. This allows open source code using the LGPL protocol to be referenced, released, and sold as a class library by commercial software. However, if the LGPL protocol code is modified or derived, all modified code, extra code involving modification, and derived code must use the LGPL protocol.

BSD is an open source protocol that you can use and modify the source code freely, or release the modified code as open source or proprietary software. When you release code that uses the BSD protocol, or perform secondary development based on the BSD protocol code, the following conditions must be met:

- If the redistributed software contains source code, the source code must include BSD of the original code.
- If the redistribution is only a binary class library/software, you need to include BSD of the original code in the class library/software documentation and copyright notice.
- Marketing using the original open source software's name, author's name, or institution's name is not permitted. BSD code encourages code-sharing but requires respect for the copyright of the code author. BSD allows users to modify and redistribute the code and allows the use or development of commercial software distribution and sales on the BSD code. It is a very friendly license for commercial integration.

The emergence of free software and openness in source code promote the rapid development of IT technologies.

Birth of Linux

In 1991, Linus Torvalds, a student of University of Helsinki, developed a new OS kernel based on Minix. Leveraging the free software such as bash and gcc provided by GNU, Linus successfully developed a new OS kernel, named Linux, and made it open source, calling on thousands of developers to improve the Linux OS. After that, programmers around the world joined with Torvalds to develop Linux. In March 1994, Linux 1.0 was officially released with the joint efforts of developers.

Though the Linux kernel was not part of the GNU project initially, it was developed with GNU. Currently, most of the Linux kernel-based OSs contain GNU software. Therefore, strictly speaking, these systems should be called GNU/Linux.

Linux now has many derivative versions. A Linux distribution is a suite that provides Linux kernel, some system software and utilities. The main differences between Linux distributions are the supported hardware devices and software package configurations. Mainstream Linux distributions include Red Hat, openSUSE, Ubuntu, and Deepin.

Linux distributions

You can visit https://www.kernel.org to view and download the Linux kernel version. The Linux kernel version number is composed of three digits:

- The first digit indicates the current major release.
- An even second digit indicates a stable version, while an odd second digit indicates a version under development.
- The third digit indicates the number of revisions.

Take openEuler 20.03 LTS as an example. The kernel version 4.19.90 is under development. The major release number is 4, and the number of revisions is 90. Compared with the stable version of kernel, 4.19.90 has many new functions.

Linux distributions are classified into either commercial or community. Commercial distributions, such as Red Hat, are maintained by companies and provide charged services, such as patch upgrades. Community distributions, such as CentOS, Debian, and openEuler, are maintained by community organization and free of charge.

1.2 Installing the openEuler OS

1.2.1 Introduction to openEuler

openEuler is a free, open source OS operated by the openEuler community. The current openEuler kernel is based on Linux and supports Kunpeng and other processors, fully unleashing the potential of computing processors. As an efficient, stable, and sustainable open source OS built by global open source contributors, openEuler applies to database, big data, cloud computing, and artificial intelligence (AI) scenarios.

The predecessor of openEuler is EulerOS running on Huawei universal servers. EulerOS is an open source OS based on the Linux kernel (currently based on Linux 4.19). It supports x86, ARM, and other processor architectures. Kunpeng processors have sparked the growth of EulerOS to become the powerful software infrastructure in the Kunpeng ecosystem. At

the end of 2019, EulerOS was officially open-sourced and renamed as openEuler. openEuler is also an innovative platform that encourages everyone to propose new ideas, explore new approaches, and practice new solutions. openEuler community is available for all developers, enterprises, and organizations. They can also develop their own OS versions based on the community release.

openEuler has two kinds of releases: innovation and long-term support (LTS). The innovation release supports technical and content innovations of Linux enthusiasts, such as openEuler 20.09. Generally, a new version is released every half a year. LTS is a stable version of openEuler, for example, openEuler LTS 20.03. Generally, a new version is released every two years. This course takes openEuler 20.03 LTS as an example to illustrate all operations involved.

1.2.2 Installing the openEuler OS

Figure 1-3 shows the openEuler OS installation process.

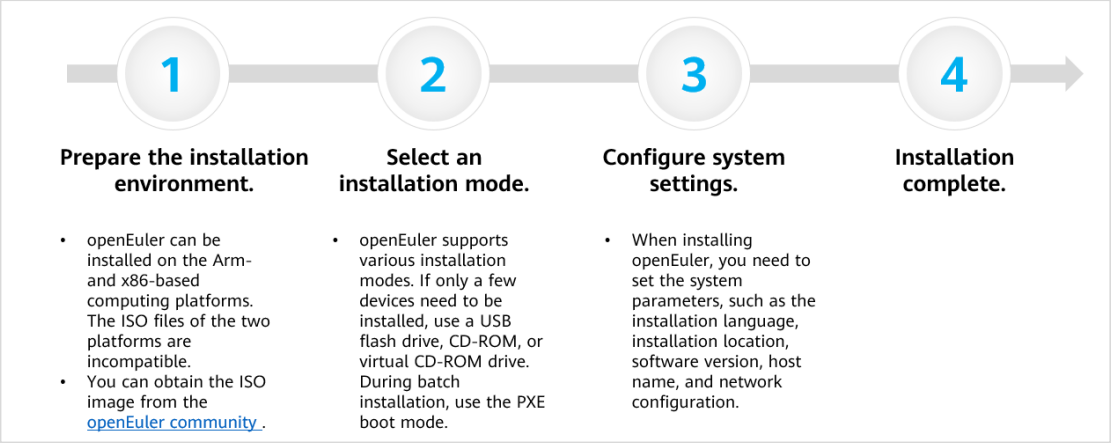


Figure 1-3 openEuler OS installation process

Preparing the installation environment

openEuler can be installed on the ARM- and x86-based computing platforms. The ISO files of the two platforms are incompatible. You can obtain the appropriate ISO image from https://openeuler.org/.

This course provides the following two environments for your reference:

- PC: Installation based on the VirtualBox virtualization environment. Generally, the PC is based on the x86 OS. Therefore, select the x86 version of openEuler (the file path is x86_64).
- Server: Installation based on the FusionCompute virtualization environment on the TaiShan 200 server. As the TaiShan 200 server uses Kunpeng 920, and the Kunpeng processor is a general-purpose processor developed by Huawei based on the ARMv8 architecture, select the ARM version of openEuler (the file path is aarch64).

The preceding two environments are prepared in the virtualization environment. Before continuing to the next step, you need to install the virtualization software, such as VirtualBox or FusionCompute, create a bare VM, and allocate proper CPU, memory, or drive space to the VM. Table 1-1 lists the minimum virtualization space required by openEuler.

Table 1-1 Minimum Virtualization Space Requirement

Component	Configuration Requirement	Remarks
Architecture	• AArch64 • x86_64	• Arm 64-bit architecture supported • Intel x86 64-bit architecture supported
CPU	2 CPUs	*HCIA-openEuler Lab Guide-PC* recommends that the CPU of the PC in the lab environment should be at least four cores.
Memory	At least 4 GB	*HCIA-openEuler Lab Guide-PC* recommends that the memory of the PC in the lab environment should be at least 16 GB.
Drive	At least 32 GB	*HCIA-openEuler Lab Guide-PC* recommends that the spare space of the drive on the PC in the lab environment should be greater than 100 GB.

The two versions involved in this course are the same. This document uses openEuler x86_64 as an example. For details, see section 1.2 "Configuring the Virtualization Environment" in *HCIA-openEuler Lab Guide-PC*.

Selecting an installation mode

openEuler supports various installation modes. If only a few devices need to be installed, use a USB flash drive, CD-ROM, or virtual CD-ROM drive. If batch installation is required, use the PXE boot mode. This course uses the virtual CD-ROM drive as an example. After the ISO file is mounted, restart the VM. The installation wizard page is displayed. Enter the default option **Test this media & install openEuler 20.03 LTS**, as shown in Figure 1-4.

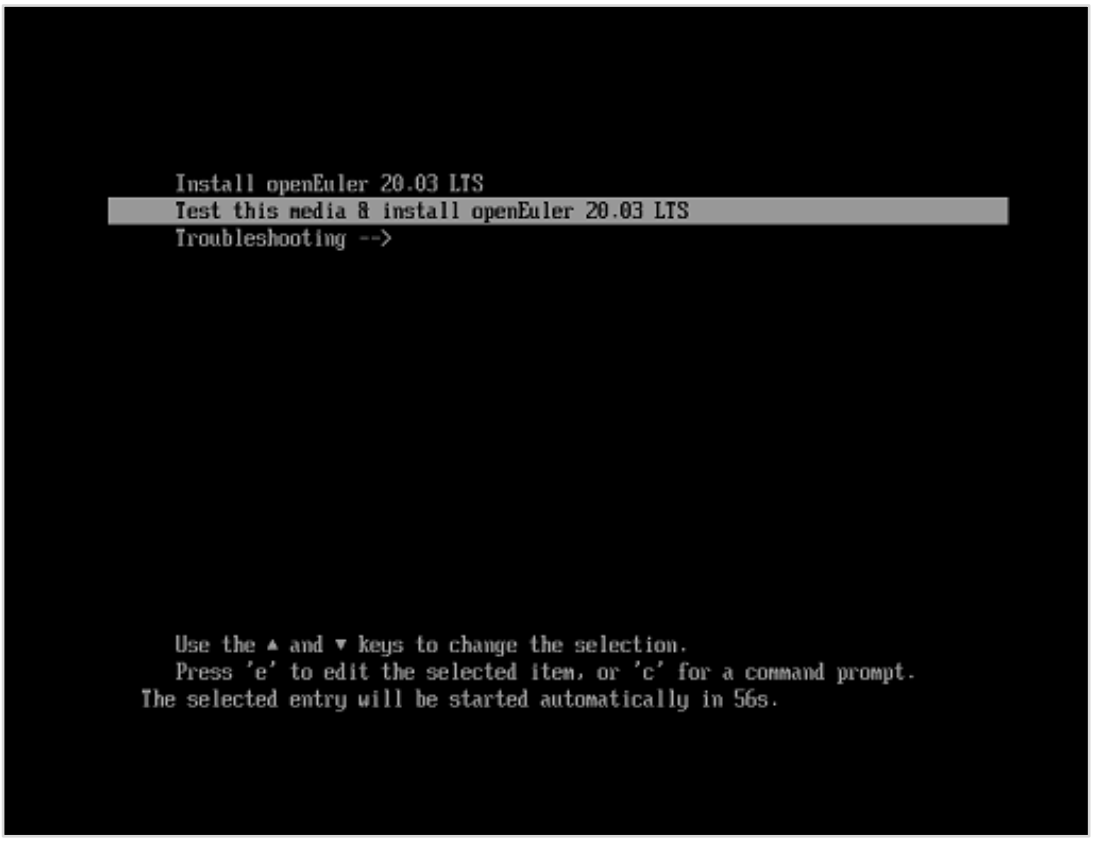


Figure 1-4 openEuler installation wizard

Configuring system settings

When installing openEuler, you need to set the system parameters, such as the installation language, installation location, software version, host name, and network configuration. Some configuration items may be displayed with warning mark. The warning mark will disappear after the item is configured, as shown in Figure 1-5.

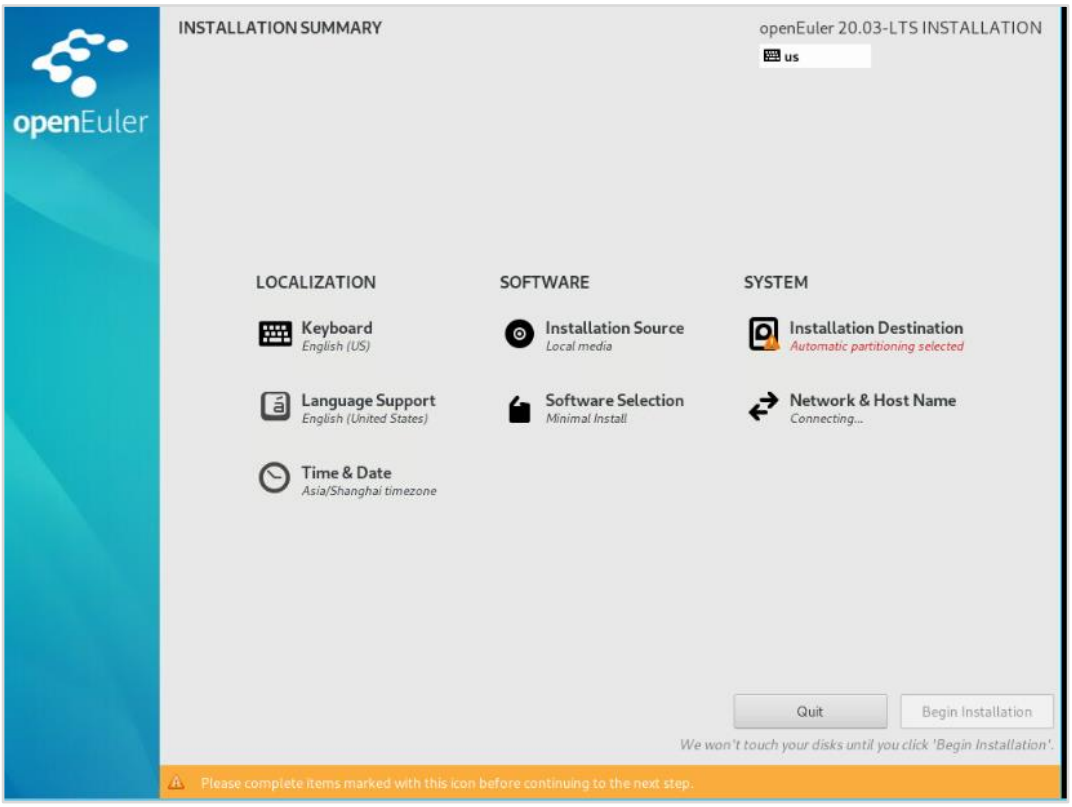


Figure 1-5 openEuler installation settings page

The following describes how to set the installation location, select the software to be installed, and create a user.

- **Setting the installation location.** Set the system installation location and system partitions manually or automatically. You are advised to set the following partitions for the openEuler system startup:
  - **swap**: Swap partition is used to swap dirty data in the memory when space is insufficient. If the memory is small, you are advised to set the swap partition size to twice the memory size. If the memory is large, you can reduce the swap partition size.
  - **/boot**: The **/boot** directory stores the files used to boot the OS. After the computer is powered on, the BIOS performs self-check and starts the computer according to the boot device (usually the drive) set in the BIOS. After taking over the hardware, the OS reads the kernel file in the **/boot** directory.
  - **/boot/efi**: the directory of the bootloader and application program to be started by the Unified extensible firmware interface (UEFI). The boot mode of openEuler for ARM is UEFI. You must create the **/boot/efi** partition before starting openEuler.
  - **/**: root partition. In Linux, everything starts from the root partition. It is the root of the file directory. All files are stored in the root directory.
- **Selecting software to be installed.** openEuler 20.03 LTS supports three software installation options:

  - In the minimum installation environment, some packages from the installation source are not installed. If the required package is not installed, you can mount the installation source to the local PC to create a repo source and use the DNF tool to install the package.
  - Under the **Server** base environment, the system has integrated easy-to-manage server components.
  - If you select **Virtualization Host**, virtualization components qemu, libvirt, and edk2 are installed by default. You can determine whether to install components such as OVS in the add-ons area.
- **Setting the password of the root user and creating a user**. During openEuler installation, you need to set the password of the **root** user, who is the super administrator of the system and has the highest permissions. Generally, the Linux administrator cannot use the **root** user to manage the system. You can create common users as required. For example, you can create a common user named openEuler or set other user names for it. During the installation, set a password with high complexity.

End

After the installation is complete, restart the system, and log in to the system as the **root** user, as shown in Figure 1-6.

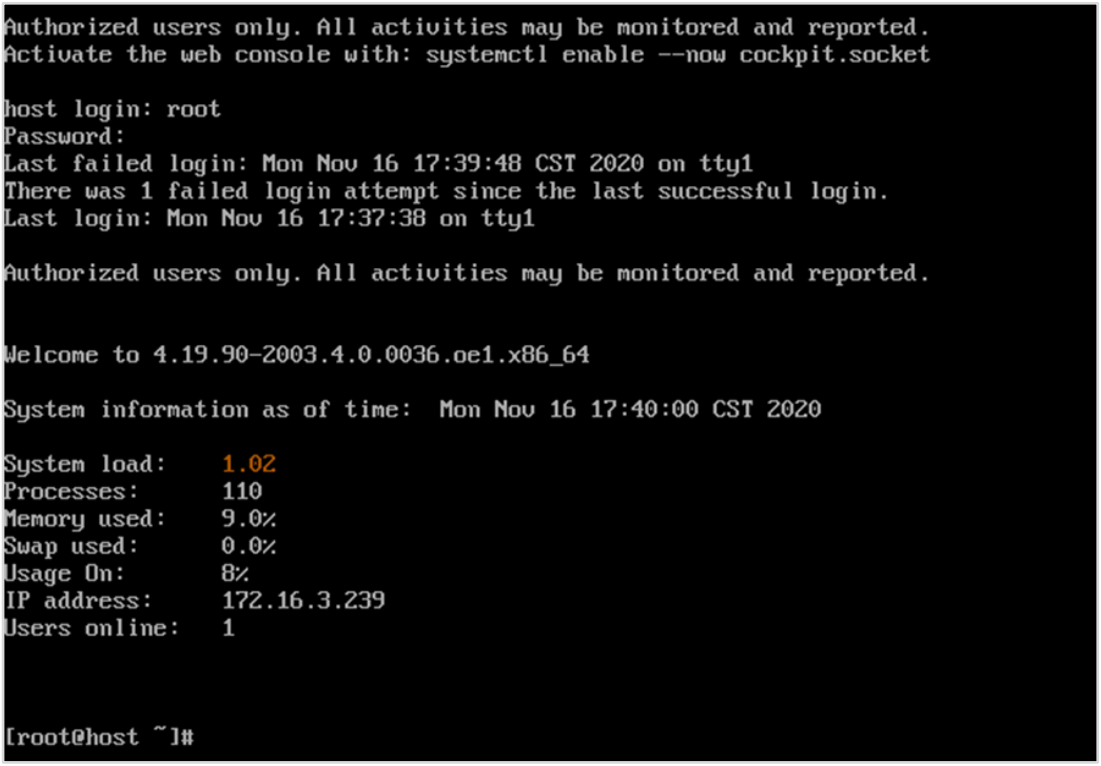


Figure 1-6 openEuler login page

For installation details, see section 1.3 "1.3 Installing the openEuler OS" in *HCIA-openEuler Lab Guide-PC*.

1.3 Using the openEuler OS

As mentioned before, An OS offers users interactive environment. Similar to Windows, Linux provides a graphical user interface (GUI), where you can perform visualization operations by using the mouse. CLI is short for command line interface, which uses the keyboard as the input tool. The Linux command line is provided by the shell program. Shell is a program compiled in C language, serving as a bridge for those who wish to use Linux. Users control the Linux system through shells, which are also used by the Linux system to display system information. **bash** is a type of shell and is similar to **cmd.exe** in Windows.

Currently, openEuler 20.03 LTS does not support GUI. The default login shell is **bash**. Therefore, this course uses the **bash** shell to illustrate operations.

There are two login methods for Linux: local login and remote login. Local login is similar to starting up your own computer or directly connecting the server to the monitor. A typical Linux system runs six virtual consoles. You can press **Ctrl+Alt+F[1,6]** to switch between the six virtual consoles. The openEuler OS supports remote login by default. You can also use PuTTY or Xshell to remotely log in to the openEuler OS.

After the startup of openEuler host, you need to enter the user name and password to log in to the system. The **root** user is the super administrator of the Linux OS and has the highest permissions. As the password of the **root** user has been set during the installation, you can log in to the system as the **root** user, as shown in Figure 1-6.

You can check whether the current user is the **root** user or a common user through the command prompt. In the Unix or Linux OS, the command prompt of the **root** user generally ends with **#**, while that of a common user generally ends with **$**. If a common user has been created during the installation, you can log in to the system as the common user using its user name and password, as shown in Figure 1-7. The default system prompt is [*Current login user*@*Host name Current location*]$. As shown in the figure, the current user is a common user named **openeuler**, and the host name is **host**. **~** indicates that the current location is the **/root** directory.

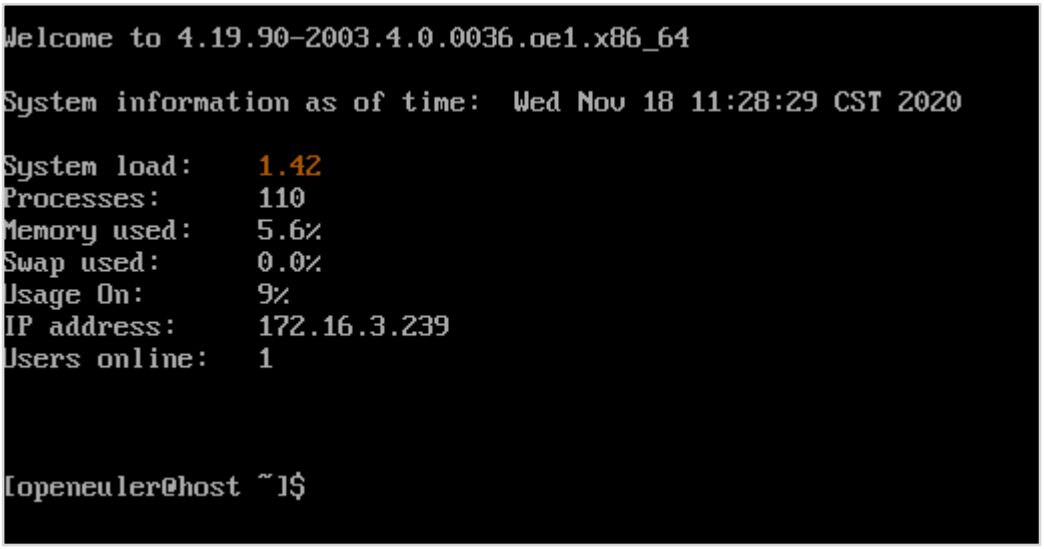


Figure 1-7 Login page of an openEuler common user

Now you can start using the openEuler OS.

You can run the following commands to view the system information. The description of the commands will be described in the following sections.

- View the system information.

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

- View the CPU information.

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

- View the memory information.

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

- View the disk information.

```
[root@host ~]# fdisk –l
```

- View the IP address.

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

1.4 Quiz

- What are the differences between the Windows OS and the Linux OS?
- What is open source software?
- What are the differences between the x86 architecture and ARM architecture?
- What are the differences between the **root** user and a common user?
- What are the other shell tools besides **bash**? What are the differences between them?

2 CLI Basics

2.1 Basic Knowledge of Linux Commands

From the previous chapter, we have learned that the shell program provides an interface through which you can access the OS kernel. It wraps the Linux kernel from the outside, as shown in Figure 2-1.

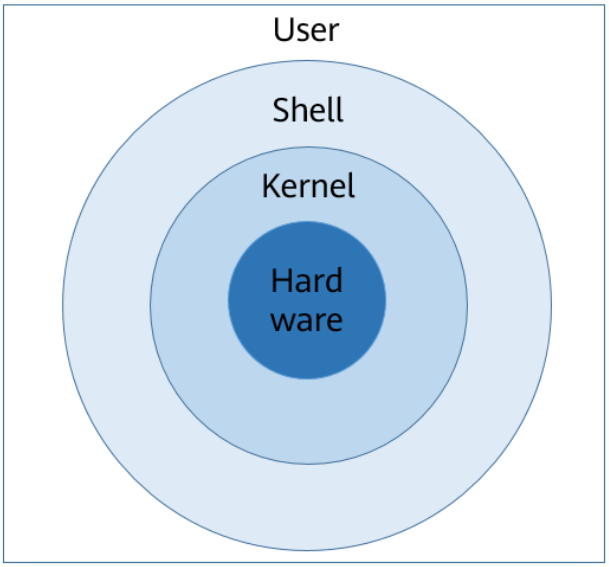


Figure 2-1 Position of the shell on Linux

By default, the openEuler user logs in to the bash shell CLI to perform operations. The CLIs are required due to the following features:

- More efficient: The Linux system allows rapid operations using a keyboard, rather than a mouse. CLIs in a script can be compiled to complete all required tasks, such as deleting expired log files, while the GUI is fixed and cannot repeat operations.
- Low overhead (compared with GUI): Running a GUI requires a large number of system resources, whereas a CLI is far more efficient. As a result, system resources can be allocated to other operations.
- CLIs are often the only choice: Most OSs for running servers do not utilize a GUI, and tools for maintaining and managing connected devices do not provide a GUI.

This course introduces the openEuler OS based on CLIs. On Linux, the command format is: ***command [-option] [parameter]***. The **[]** symbol indicates an option. Commands are composed of three parts:

- The command to be executed. Take **ls -la** as an example: **ls** displays the file list. The command word used on the Linux CLI uniquely identifies a command.

- The option. The number and content of options vary with commands.
  - If there are multiple options, you can write them together. For example, **ls -la** contains two options: **-l** and **-a**. **-l** lists the file list and detailed file information together. **-a** displays the information about all the files or directories in the current directory.
  - Options can be short by following one hyphen (-) or long by following two hyphens (--). For example, **ls -a** equals **ls --all**.
- The parameter indicates the operation object. Generally, it can be a file name, directory, or username.

Getting familiar with CLI skills in advance is helpful for you to master the operations. For example, in the bash environment, you can press **Tab** to automatically supplement commands or file names, saving time and improving accuracy. If no command is entered, press **Tab** twice to list all available commands. If you have entered a part of the command name or file name, pressing **Tab** will supplement them automatically.

In addition, the following table lists the common shortcuts.

Table 2-1 Common Linux command line keyboard shortcuts

Shortcut	Description
↑	Displays recently executed commands and enables you to quickly select and run them.
↓	Facilitates command selection (used together with ↑).
Home	Moves the cursor to the beginning of the current line.
Ctrl+A	Moves the cursor to the beginning of a line.
Ctrl+E	Moves the cursor to the end of a line.
Ctrl+C	Stops the current program.
Ctrl+L	Clears the screen.

2.2 Basic Linux Commands

Linux commands can be classified as follows. Common commands of different types are introduced in later sections.

Table 2-2 Classification of Linux commands

Category	Example Commands	Section
Login and power management	login, shutdown, halt, reboot, install, exit, and last	2 "CLI Basics"
File processing	file, mkdir, grep, dd, find, mv, ls, diff, cat, and ln	2 "CLI Basics" and 3 "Text Editors and Text Processing"
System management	df, top, free, quota, at, ip, kill, and crontab	7 "System Management"
Network operation	ifconfig, ip, ping, netstat, telnet, ftp, route, rlogin, rcp, finger, mail, and nslookup	7 "System Management"
File system and storage operations	fdisk, df, parted, mkfs, pvcreate, vgcreate, lvcreate, vgs, lvextend, mount, and format	6 "File System and Storage Management"
System security	passwd, su, umask, chgrp, chmod, chown, chattr, sudo ps, and who	4 "User and Permission Management"
Others	tar, unzip, gunzip, unarj, mtools, and man	2 "CLI Basics"

- Note: Linux commands are far richer than the examples above. Common commands of different types are introduced in later sections.

2.2.2 Login Commands

2.2.2.1 Introduction to Login Commands

login

After the host with openEuler installed is started, you are required to log in to the system. If you choose to log in to the Linux OS in command line mode, the first command required is **login**. Enter the **root** username or the username defined during installation, press **Enter**, and enter the password to log in to the system. For security purposes, characters are not displayed on the screen and the cursor is not moved when you enter the password.

Linux is a multi-user OS that allows multiple users to log in at the same time and a single user to log in multiple times. This is because Linux, like many versions of Unix, provides a virtual console access mode that allows users to log in to the console (a monitor and keyboard that are directly connected to the system) multiple times simultaneously. Each virtual console can be regarded as an independent workstation and you can switch between workstations. You can press **Alt** and a function key (typically **F1** to **F6**) to switch between virtual consoles. For example, if you press **Alt+F2** after login, the **login:** prompt is displayed, indicating that you have seen the second virtual console. You can also press **Alt+F1** to return to the first virtual console. A newly installed Linux system allows users to access the first six virtual consoles using the **Alt+F1** to **Alt+F6** keys. The most helpful thing about the

virtual console is that when a program goes wrong and causes a system deadlock, you can switch to another virtual console and close the program.

last

last is used to display the recent logins of users or terminals, and is applicable to all users. Run the **last** command to view a program's log. The user will know who used, or attempted to connect to, the system. Main options:

- **-n**: specifies the number of output records.
- **-t tty**: displays the login status of the specified virtual console.
- **-y**: displays the year, month, and day of the record.
- **-ID**: displays the username.
- **-x**: displays the history of system shutdowns, user logins, and user logouts.

exit

exit is used to exit the current shell, and is applicable to all users.

logout

logout is used to log out of the system, and is applicable to all users. **logout** can only be used when the current shell is the login shell.

2.2.2.2 Login Command Exercise

Exercise: Perform the following operations using the login commands.

- Log in as the **root** user.

```
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: 2020-12-07 Monday 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 ~]#
```

- Open a new virtual console by pressing **ALT+F2**.
- Switch to an openEuler user.

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

- View recent user logins.

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

- View recent terminal logins.

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

2.2.3 Power Management Commands

shutdown

shutdown is used to shut down the computer, and is only applicable to the superuser. For a computer system, a superuser is a special user for system management. Compared with common users, it has the highest permissions to configure and maintain the entire system. Common users are only granted a subset of its permissions. Main options:

- **-h**: powers off the computer after it is shut down.
- **-r**: powers on the computer after it is shut down. (This operation is equivalent to restarting the computer.)
- **-t**: indicates the time after which the **init** program is shut down before changing to another run level.
- **-k**: sends a warning signal to each user. It does not shut down the computer.
- **-F**: forcibly performs file system consistency check (fsck) when the computer is restarted.
- **-time**: specifies the time before the shutdown.

The **shutdown** command safely shuts down the system. It is dangerous to shut down a Linux system by directly powering it off. Different from Windows, Linux runs many processes in the background. Therefore, forcible shutdown may cause loss of process data, making the system unstable and even damaging hardware in some systems. If you run the **shutdown** command to shut down the system, the system administrator notifies all login users that the system will be shut down and the **login** command will be frozen. As a result, no further users can log in to the system.

halt

halt is used to shut down the system, and is only applicable to the superuser. When **halt** is executed, the application process is killed. System calls are synchronized to forcibly write the data in the buffer to the disk. After the write operation of the file system is complete, the kernel is stopped. If the system run level is 0 or 6, it shuts down the system. Otherwise, use the **shutdown** command (with the **-h** option) instead. Main options:

- **-n**: prevents synchronization of system calls. It is used after the root partition is repaired using fsck, and prevents the kernel from overwriting the repaired superblock with that of an earlier version. The **sync** command is used to forcibly write the data in the buffer to the disk immediately. The **fsck** command is used to check and rectify faults in the file system. A superblock is located at very front of a block group. It describes the data structure of the overall information about a file system, including the static distribution of directories and files in the file system, and the size and quantity of each structure of the file system.
- **-w**: writes the **wtmp** file in **/var/log/wtmp** instead of restarting or shutting down the system. **/var/log/wtmp** is a binary file that records the number of login times and duration of each user.
- **-i**: shuts down all network interfaces before shutting down or restarting the system.
- **-d**: shuts down the system without making a record.

reboot

reboot is used to restart the computer, and is applicable to the system administrator. Main options:

- **-n**: saves the data and restarts the system.
- **-w**: writes records to the **/var/log/wtmp** file. It does not restart the system.
- **-d**: does not write records to the **/var/log/wtmp** file. (The **-n** option contains **-d**.)
- **-i**: restarts the system after disabling the network settings.

2.2.4 File Management Commands

2.2.4.1 File Directories

On the Linux OS, everything is a file. The file directory utilizes a tree structure, with **/** as the root directory which extends to the branches of directories and subdirectories, as shown in Figure 2-2. The root directory plays the most important role in the entire system, and is related to operations such as startup, restoration, and system recovery. Directory levels are also divided by **/**. For example, **/home/openEuler** indicates the openEuler directory in the home directory.

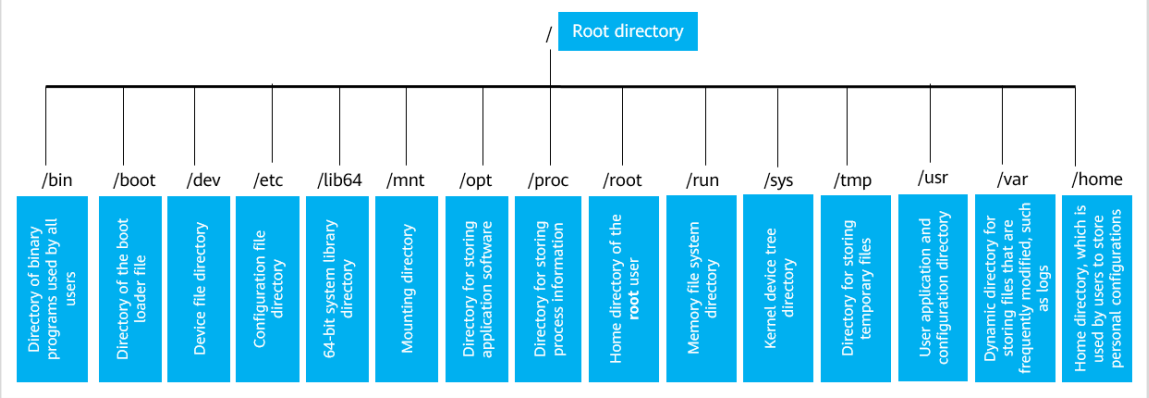


Figure 2-2 File directory structure and the root directory

After logging in to the system, run the **ls /** command to view the files or file directories in the root directory. The command output is as follows:

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

The following table lists the usage of some directories in the root directory:

Table 2-3 Main Linux directories and usage

Directory	Main Files and Their Functions
/bin	**bin** is short for binary. This directory stores the most frequently used commands.
/boot	Stores some core files used for starting the Linux OS, including some connection files and image files.
/dev	**dev** is short for device, and this directory stores Linux external device files. The method used to access devices on Linux is the same as that for accessing files.
/etc	Stores all configuration files and subdirectories required for system management.
/home	The home directory of a user. On Linux, each user has a directory. Generally, the directory is named after the username.
/lib	Stores the system's most basic dynamic link libraries (DLLs). The function of this directory is similar to the storing of DLL files on Windows. Almost all applications need to use these shared libraries. **/lib64** stores the shared library of a 64-bit system.
/mnt	Temporarily mounts other file systems.
/opt	Stores additional software that is installed on the host.
/proc	A virtual directory, which is the mapping of the system memory. You can obtain the system information by directly accessing this directory.
/root	This directory is the home directory of the system administrator, who is also called the superuser.
/sbin	**s** indicates the superuser. This directory stores the system management program used by the system administrator.
/srv	Stores the data that needs to be extracted after a service is started.
/tmp	Stores temporary files.
/usr	Many user applications and files are stored in this important directory, which is similar to the **program files** directory on Windows. **/usr/bin** is the application used by system users. **/usr/sbin** is an advanced management program and system daemon used by the superuser. **/usr/src** is the default directory for storing the kernel source code.

Directory	Main Files and Their Functions
/var	Stores content that is constantly expanded, such as log files. You are advised to place frequently modified directories here.
/run	A temporary file system that stores the information generated after the system is started. The information is cleared or deleted when the system is restarted.

2.2.4.2 File Paths

When using the shell or invoking an application, specify the path of the invoked program. The path can be an absolute or a relative path.

- Absolute path: On Linux, an absolute path starts from **/** (also called the root directory) and is irrelevant to its position in the directory structure. If a path starts from **/**, it must be an absolute path.
- Relative path: The relative path is relative to the current directory and starts from the current position in the directory structure.

Assume that there are two directories: **/home/smc/config** and **/home/smc/bin**. To access **/home/smc/bin** from **/home/smc/config**, the user has two ways:

- Absolute path: cd /home/smc/bin
- Relative path: **cd ../bin**. Here, two dots (..) indicate the upper-level directory. If there is only one dot, it indicates the current directory.

2.2.4.3 Basic File Operation Commands

ls

ls is short for list. The **ls** command is one of the most frequently used Linux commands, and lists the content of a directory or file information. The output of this command is sorted by file name by default. If no target is specified, the content of the current directory is listed.

Syntax:

ls [OPTIONS] [FILE]

- **-a**: displays all files and directories. Files or directories whose names start with a dot (.) are hidden.
- **-l**: lists information such as the file type, permission, owner, and file size, in addition to the file name.
- **-t**: lists files by creation time.
- **-R**: lists any files in the current directory in sequence.

cd

The **cd** command is used to change the current working directory.

Syntax:

cd *[dir]*

- **cd /usr**: accesses the /usr directory.

- **cd .**: accesses the current directory.
- **cd ..**: accesses the upper-level directory. Two dots indicate the parent directory.
- **cd -**: accesses the previous directory. This command is used to quickly switch between two directories.
- **cd ~**: accesses the home directory. The **root** user returns to the **/root** directory, and common users return to the **/home** directory.
- The **cd** command accesses the home directory by default if no parameter is added.

pwd

The **pwd** command is used to print the current working directory. The **pwd** command has two options: **-L** and **-P**.

-L: outputs the connection path when the directory is linked.

-P: outputs the physical path.

Exercise: file commands ls, cd, and pwd:

- View the files or file directories in the root directory:

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

- Access the root directory.

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

- View details about all files in the root directory and sort the files by time:

```
[root@host /]# ls –alt
```

- Access the home directory through an absolute path:

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

- View the files in the home directory through an absolute path:

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

- Access the home directory through a relative path and the following file directory is displayed, for example, the openEuler directory.

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

- View the current file directory:

```
[root@host openEuler]# pwd
# The openEuler file directory in the home directory is accessed. Run the pwd command and the following information is displayed:
/home/openEuler
```

- Return to the upper-level directory:

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

- Return to the main directory:

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

- View the current file directory:

```
[root@host ~]# pwd
# When the user is the root user, run the pwd command and the following information is displayed:
/root
```

mkdir

The **mkdir** command is used to create one or multiple directories or folders. If a directory already exists, an error is reported by default. The **-p** option enables the command to report no error in this case, and can also be used to automatically create a parent directory.

Syntax:

mkdir [OPTIONS] DIRECTORY

Common usage:

- Create a directory named **dir1**:

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

- Create multiple directories:

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

- If **dir1** and **dir2** do not exist, create the **dir1** parent directory, **dir1/dir2** subdirectory, and **dir1/dir2/dir3** subdirectory.

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

- When a directory is created, the creation process is displayed.

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

touch

The **touch** command is used to create an empty file and change the timestamp of a file.

Syntax:

touch *[OPTIONS] FILE*

Common usage:

- Create a blank file named **file** and set the timestamp to the current time:

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

- Modify only the file access time:

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

- Modify only the content change time:

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

- Set the timestamp to a specified time:

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

cp

The **cp** command is used to copy files or directories. You can copy a single file or multiple files at a time. Exercise caution when running this command, as there is a risk of data loss.

Syntax:

cp ***[OPTIONS] SOURCE DIRECTORY or FILE***

- **-a**: copies the files of a directory while retaining the links and file attributes.
- **-p**: copies the file content, modification time, and access permissions to the new file.
- **-r**: copies all subdirectories and files in the source directory file.
- **-l**: generates a link file but does not copy the file.

Common usage:

- Copy the file **f1** and name the new file **f2**.

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

- Copy **f1** to the **d1** directory and keep the name of the new file unchanged.

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

- Copy multiple files to the same directory. When multiple files are copied, the destination must be a directory.

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

- If **f2** already exists, wait for confirmation before overwriting it. Enter **y** to overwrite the file.

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

- Copy all subdirectories and files in the **d1** directory, and name the new directory **d2**. The **-r** option is required for copying the directory.

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

- Add the **-v** option to display the replication process.

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

- Use the **-a** option to keep the original attributes when copying the **f1** file. It can be used to copy block devices, character devices, and pipe files.

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

mv

The **mv** command is used to move a file or directory. Exercise caution when running this command, as there is a risk of data loss. If the source file and target file are in the same parent directory, the **mv** command is used to rename the file.

Syntax:

mv ***[OPTIONS] SOURCE DIRECTORY or FILE***

- **-b**: backs up a file before overwriting it.
- **-f**: forcibly overwrites the target file without asking the user.
- **-i**: overwrites the target file at the destination after obtaining the user's consent.
- **-u**: updates the target file only when the source file is newer than the target.

The usage of **mv** is the same as that of **cp**. The common usage is as follows:

- Move **f1** to **f2**.

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

- Move **f1** to the **d1** directory and keep the name of the new file unchanged.

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

- Move multiple files to the same directory.

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

- If **f2** already exists, wait for confirmation before overwriting it. Enter **y** to overwrite the file.

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

- Move **f1** to overwrite the existing **f2** and back up **f2** before overwriting it.

```
[root@host ~]# mv –b f1 f2
```

- Move all subdirectories and files in the **d1** directory, and name the new directory **d2**. The **-r** option is required for moving the directory.

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

- When all subdirectories and files in the **d1** directory are moved, the moving process is displayed.

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

- Move **f1** to **f2**. If **f2** already exists, use **-f** to overwrite **f2** without confirming. The function of **-f** is opposite to that of **-i**. If the command contains both **-i** and **-f**, the command written on the right takes effect.

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

rm

The **rm** command is used to delete files or directories. Exercise caution when running this command, as it is not possible to completely restore files deleted in this manner. As the **rm** command does not move files to a place from which they can be restored, such as a "recycle bin", the deletion operation cannot be revoked.

Syntax:

rm ***[OPTIONS] DIRECTORY or FILE***

- **-i**: performs interactive deletion. The system asks the user before the deletion.
- **-f**: forcibly deletes a file without asking.
- **-r**: instructs **rm** to recursively delete all directories and subdirectories listed in the parameter.
- **-v**: displays the detailed procedure.

Common usage:

- Delete the **f1** file.

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

- Delete files **f1** and **f2**.

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

- Interactively delete the **f1** file. Before the deletion, wait for confirmation. Enter **y** to delete the file.

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

- Forcibly delete file **f1**. If the command contains both **-i** and **-f**, the command written on the right takes effect.

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

- Recursively delete the **d1** directory and all subdirectories and files in the directory.

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

- Recursively delete the **d1** directory and all subdirectories and files in the directory, and display the process.

```
[root@host ~]# rm –rv d1
```

find

The **find** command is used to search for files in a specified directory. You can specify search criteria, such as file name, file type, user, and even timestamp.

Syntax:

find ***[PATH] [EXPRESSION]***

- **-name**: searches for files by file name.
- **-perm**: searches for files by file permission.
- **-user**: searches for files by file owner.
- **-mtime -n +n**: searches for files by modification time.
- Search for files whose names contain **book**.

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

- Search for files whose sizes are 0.

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

- Search for files whose file types are soft link.

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

- Search for files whose names contain **passwd** in the **/etc** directory.

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

- Search for empty files or directories and delete them.

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

locate

The **locate** command is used to quickly check whether a specified file exists in the file system. It creates a database with a file name and path, and then searches the database for the file name and path.

Syntax:

locate ***[OPTIONS] PATTERN***

- **-e**: specifies the exempt scope.
- **-f**: excludes specified files.
- **-r**: uses a regular expression as the search criteria.
- **-o**: specifies the file name.
- **-d**: specifies the file path.

When you run the **locate** command to search for a file, the system searches for the file in the index database. If the database is not updated for a long time or does not exist, the system displays the message "locate: can not open `/var/lib/mlocate/mlocate.db`: No such file or directory". In this case, run the **updatedb** command to update the database.

which

The **which** command is used to search for executable files in the directory specified by *PATH*. You can run the **which** command to check whether a system command exists and where the command is executed.

Syntax:

which ***[OPTIONS] PROGRAMNAME***

Common usage:

- Search for the absolute path of the **ls** command.

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

- Search for the absolute path of the **ls** command. If there are matched files in multiple directories, all the files are displayed.

```
[root@host ~]# which –a ls
```

- Search for multiple files.

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

ln

ln is short for **link**. The **ln** command is used to create a link file. It creates a synchronous link for a file in another location. When the same file is required in different directories, instead of placing the file in each required directory, users only need to place it in a fixed directory and then run the **ln** command to create a link to the file in other directories, which also saves disk space.

There are two types of links on Linux: soft link (also known as symbolic link) and hard link, as shown in the following table.

Table 2-4 Linux link file types

Soft Link	Hard Link
A path, similar to a Windows shortcut	A file copy, which does not occupy the actual space
A link, which becomes invalid after the source file is deleted	A link, which has no impact on the source file after being deleted
Linking to a directory is supported	Linking to a directory is not supported
Cross-file system linking is supported	Cross-file system linking is not supported

If the **ln** command does not contain any option, a hard link is created by default.

Syntax:

ln ***[OPTIONS] SOURCE [DIRECTORY or FILE]***

- **-b**: deletes and overwrites the existing link.

- **-d**: allows the superuser to create hard links to directories.
- **-f**: forcibly executes the command.
- **-i**: indicates the interactive mode. If the file exists, the system prompts you to overwrite it.
- **-n**: regards symbolic links as common directories.
- **-s**: indicates a soft link (symbolic link).

Common usage:

- Create a hard link for the **f1** file and name it **f2**.

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

- Create a soft link for the **d1** directory and name it **d2**.

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

- Interactively create the **f1** file link and name it **f2**. If **f2** already exists, the system asks you whether to continue the creation. Enter **y** to confirm the creation and overwrite the original file.

```
[root@host ~]# ln –i f1 f2
```

- Forcibly create a file link between **f1** and **f2**.

```
[root@host ~]# ln –f f1 f2
```

- Create a link file **f2** for **f1** and overwrite **f2** if it already exists.

```
[root@host ~]# ln –b f1 f2
```

For more exercises about basic file operations, see section 2.2 "Basic Operations of the bash Command" in the *HCIA-openEuler Lab Guide (PC Edition)*.

2.2.4.4 File Packing and Compressing Commands

gzip

gzip is a command used to compress and decompress files on Linux. Specifically, **gzip** can be used to compress large, rarely-used files to save disk space. According to statistics, the **gzip** command has a compression ratio of 60% to 70% for text files. After a file is compressed using **gzip**, the file name is suffixed with **.gz**. Syntax:

gzip ***[OPTIONS] DIRECTORY or FILE***

- **-d**: decompresses a package.
- **-f**: forcibly compresses a file, regardless of whether the file name exists or whether the file is a symbolic link.
- **-l**: lists information about compressed files.
- **-r**: recursively processes all files and subdirectories in a specified directory.
- **-v**: displays the command execution process.

Common usage:

- Compress file **f1**.

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

- Compress all files and subdirectories in the **d1** directory.

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

- Decompress file **f1**.

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

- Decompress the **d1** directory and display the execution process.

```
[root@host ~]# gzip –drv d1
```

tar

The **tar** command is used to pack files. You can pack multiple files into a package to facilitate data transfers. The **tar** command is usually used together with options **-z**, **-j**, and **-J**, which corresponds to the **gzip**, **bzip2**, and **xz** compression tools, respectively. After a compression option is specified, the **tar** command starts the corresponding compression tool to compress or decompress data and transmits data through the pipe and compression tool.

Syntax:

tar ***[OPTIONS] [FILE]***

- **-c**: creates a compressed file.
- **-x**: extracts files from a compressed file.
- **-t**: displays the content of a compressed file.
- **-z**: supports **gzip** decompression.
- **-j**: supports **bzip2** decompression.
- **-v**: displays the operation process.

Common usage:

- Pack the **dir1** directory and all the content in the directory.

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

- List the content in the package.

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

- Decompress the package to the current directory.

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

- Pack the files and compress them using **gzip**.

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

- Pack the files and compress them using **bzip2**.

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

- Pack the files and compress them using **xz**.

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

- Decompress the package to the **/tmp** directory. (By default, the path is the current directory.)

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

- Display the decompression process.

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

For more exercises about basic file operations, see section 2.4 "Packaging and Compression Commands" in the *HCIA-openEuler Lab Guide (PC Edition)*.

2.2.5 Help Commands

man

Linux provides various documents, which can be accessed by executing commands such as **man**, **info**, and **txt**. **man** is used to view the manual, which is classified into the following nine types:

Table 2-5 man command types

No.	Description
1	Commands or programs that can be operated by users in the shell
2	Functions and tools that can be invoked by the system kernel
3	Common functions and function libraries
4	Device document description, which is usually in the **/dev** directory
5	File format and conventions
6	Games
7	Miscellaneous (including macros and conventions)
8	System management commands (applicable only to the **root** user)
9	Kernel routines (non-standard)

The common types of **man** documents are 1, 4, 5, and 8. **man** searches results by chapter number in the manual. For example, after **man sleep** is entered, only command manuals are displayed by default. To view library functions, enter **man 3 sleep**, in which the type **3** represents some common functions and function libraries. You can search for the **man** document by keyword (**man –k** *KEYWORD*). For example, enter **sleep** and view the type corresponding to the keyword.

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

Figure 2-3 shows the command output.

```
[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 ~]#
```

Figure 2-3 Searching for man documents by keyword

help

Due to the vast number of commands in the Linux system, it is almost impossible to remember them all. However, you can run the **help** command to obtain detailed information.

Syntax:

help ***[OPTIONS] [COMMAND]***

- **-d**: displays a brief description of the command.
- **-s**: displays a brief description of the command syntax.

Common usage:

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

As shown in Figure 2-4, the following information is displayed after you run **help** for the **pwd** command:

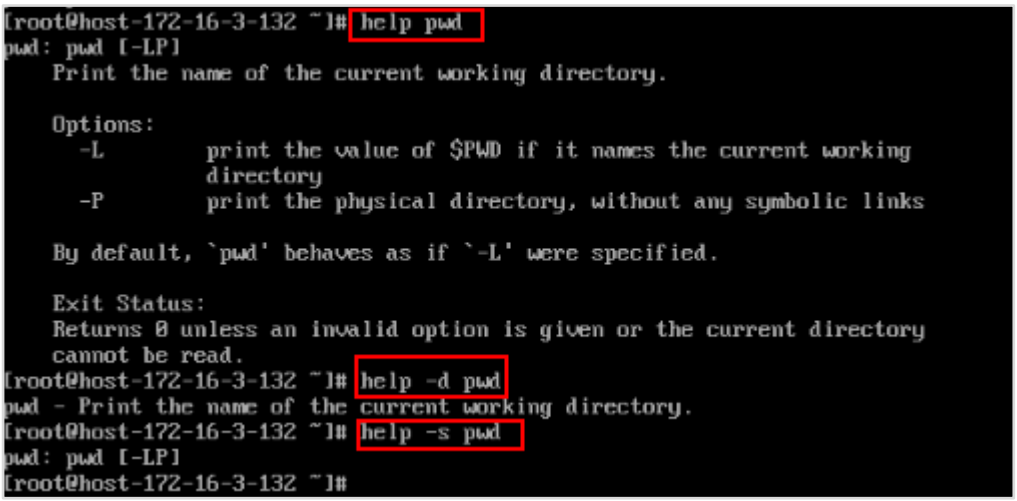


Figure 2-4 Running the help command to obtain the help information about the pwd command

For more exercises about help commands, see section 2.5 "Help Command" in the *HCIA-openEuler Lab Guide (PC Edition)*.

2.3 Quiz

- Which home directories will be accessed when you run the **cd ~** command for different users?
- What are the differences between the **shutdown** and **halt** commands?
- What are the differences between absolute paths and relative paths? How to use them?
- What are the differences between soft links and hard links?
- What are the impacts on soft links and hard links after source files are deleted?

3 Text Editors and Text Processing

3.1 Common Linux Text Editors

Text processing is a basic OS operation used to manage files. A text editor, a type of computer software, is mainly used to write and view text files. Different text editors have different auxiliary functions. There are many types of Linux text editors. Some accompany specific environments and others can be installed as needed. Some common Linux text editors are Emacs, Nano, gedit, KEDIT, Vi, and Vim.

Emacs

Emacs is more like an OS than an editor. It comes with a built-in web browser, IRC client, calculator, and even Tetris. Emacs is used on a Linux GUI. It is powerful and extensible, and can be integrated with many free software programming tools. However, it is difficult for common users to get started.

Nano

Nano is a simple text editor on a command line interface. Developed to replace the closed-source Pico text editor, the first version was released, licensed under the GNU General Public License (GPL), in 1999. This free software is also a part of the GNU Project. Nano has many user-friendly features such as syntax highlighting, regular expression searching and replacing, smooth scrolling, multiple buffers, custom shortcut keys, undo, and repeat editing. Nano is easy to use and very suitable for simple text editing. However, complex text editing is time-consuming, because it has no powerful commands available for complex operations, for example, no support for macros, editing multiple files at a time, window splitting, vertical block/rectangle selection and editing, and automatic completion.

gedit

gedit is a text editor in the GNOME desktop environment and is compatible with UTF-8. gedit is free software which is easy to use and provides syntax highlighting, multiple character encodings including GB2312 and GBK, and GUI tabs for editing multiple files. It also supports a full undo and redo system, searching and replacing, and editing remote files with the GNOME VFS library. It is easy to use on a GUI, featuring a Windows-like experience, for example, the same shortcut keys for operations like copy and paste. However, it requires an installed GUI.

KEDIT

Similar to gedit in the GNOME environment, KEDIT is a text editor in the K desktop environment (KDE). It is a small editor, especially suitable for browsing text and various configuration files. It is easy to use on a GUI, featuring a Windows-like experience. However, it also requires an installed GUI.

Vi

Vi, the oldest text editor, is a standard Unix text editor and one of the most popular text editors. All Linux and Unix OSs have the Vi text editor by default. Although Vi operations are different from those of other text editors (such as gedit), Vi is still used frequently because it only needs a character interface and can be used in all Unix and Linux environments. Vi has three command modes:

- **Command**: used to enter commands
- **Insert**: used to insert text
- **Visual**: used to browse text

Vim

Vim is a text editor developed from Vi. Equipped with many convenient programming functions such as code completion, compilation, and error redirection, Vim is frequently used by programmers, and is also one of Unix-like system users' favorite editors alongside Emacs. The first version of Vim was released in 1991 by Bram Moolenaar. The initial name was Vi IMitation. After it developed more functions, the name was changed to Vi IMproved. It is now free and open-source software. Vim has multiple modes:

- Basic modes:
  - **Normal**: This mode allows the usage of editor commands, such as moving a cursor and deleting text. It is the default mode after Vim is started, which is the opposite of what many new users expect (most editors are in **Insert** mode by default). Vim's powerful editing capabilities come from its **Normal**-mode commands, which require an operator at the command end. For example, the **dd** command is used to delete the current line, and you can replace the second **d** with another move command (for example, **j**) to move one line down and delete both the current and next lines. In addition, you can specify the number of times that a command can be repeated. For example, **2dd** indicates that the command is repeated twice, and the effect is the same as that of **dj**. After learning various move and jump commands and Normal-mode edit commands, and understanding how to flexibly combine these commands, you can edit text more efficiently, compared with using modeless editors. There are many ways to enter **Insert** mode from **Normal** mode. For example, press **a** (append) or **i** (insert).
  - **Insert**: In this mode, most keys are used to insert text into the text buffer. New users often want the text editor to remain in this mode throughout the editing process. You can press **Esc** to enter **Normal** mode from **Insert** mode.
  - **Visual**: While this mode is similar to **Normal** mode, the main difference is that using a move command will enlarge the highlighted text area, which can be a character, a line, or a piece of text. A non-move command is executed on the highlighted area. Vim's text object can also be used in this mode, similar to the move command.
  - **Select**: This mode is similar to a modeless editor in terms of function (Windows standard text control). In this mode, you can use a mouse or an arrow key to highlight the text. However, if you enter any character, Vim replaces the selected highlighted text block with that character and automatically enters **Insert** mode.
  - **Command-line**: In this mode, you can enter text that will be interpreted and executed. For example, you can search (using **/** and **?**) as well as execute and filter commands (using **:** and **!** respectively). After a command is executed, Vim

returns to the mode before **Command-line** mode, which is usually **Normal** mode.

- **Ex**: This mode is similar to **Command-line** mode. You can run multiple commands at a time before running the **:visual** command to exit **Ex** mode.

- Derivative modes:
  - **Operator-pending**: In this mode, Vim waits for an action to complete a command after an operation command is executed. Vim also supports the use of text objects as an action in **Operator-pending** mode, such as **aw** (a word), **as** (a sentence), and **ap** (a paragraph).
  - **Insert Normal**: You can enter this mode by pressing **Ctrl+O** in **Insert** mode. In this case, the system temporarily enters **Normal** mode. After a command is executed, Vim returns to **Insert** mode.
  - **Insert Visual**: You can enter this mode by pressing **Ctrl+O** in **Insert** mode when starting a visual selection. Vim returns to **Insert** mode when the visual selection ends.
  - **Insert Select**: You can enter this mode by dragging the mouse or pressing **Shift** in **Insert** mode. Vim returns to **Insert** mode when the selection ends.
  - **Replace**: This is a special version of **Insert** mode that allows you to perform the same operations, but each input character overwrites an existing one in the text buffer. You can enter this mode by pressing **R** in **Normal** mode.
- eVim
  - **eVim**: easy Vim (eVim) is a special GUI mode that makes Vim function like a modeless editor. The editor automatically enters and stays in **Insert** mode. You can operate text only by using a menu, a mouse, and keyboard control keys. You can enter this mode inputting **evim** or **vim -y** in the CLI.

3.2 Text Editors and Text Processing

3.2.1 Vim Text Editor

Vim facilitates operations such as text editing, modifying, and saving, and provides many shortcut keys for text processing. This part describes how to use Vim to open a file, move the cursor, perform data operations, display and hide a line number, find and replace characters, highlight search results, modify a file, undo or redo an operation, save a file and exit.

Opening a File

Run the following command to open a file:

vim [OPTIONS] [FILE]

- **-c**: runs a specified command before opening a file.
- **-R**: opens a file in read-only mode but allows you to forcibly save the file.
- **-M**: opens a file in read-only mode and does not allow you to forcibly save the file.
- **-r**: recovers a crashed session.

- **+***num*: starts at line *num*.

Examples:

- If the **filename** file exists, it is opened and its content is displayed. If the file does not exist. Vim displays **[New File]** at the bottom of the screen and creates the file while saving it for the first time.

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

- If the **filename** file is opened in read-only mode, Vim displays **[readonly]** at the bottom of the screen and saves it forcibly.

```
[root@openEuler ~]$ vim filename
~
~
~
~
~
"filename" [readonly]
```

- The following is displayed when the **i** command is executed:

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

- When the **:wq** command is executed, a message indicating that **!** needs to be added to forcibly overwrite the file is displayed.

```
E45: 'readonly' option is set (add ! to override)
```

- The **filename** file is displayed in read-only mode, but it cannot be forcibly saved.

```
[root@host ~]# vim –M filename
```

- The following is displayed when the **i** command is executed:

```
E21: Cannot make changes, 'modifiable' is off
```

- When the **:wq** command is executed, the following information is displayed:

```
E142: File not written: Writing is disabled by 'write' option
```

Moving the Cursor

In Vim **Normal** mode, you can use shortcut keys to move the cursor, simplifying operations. The following table lists common shortcut keys.

Table 3-1 Common shortcut keys for moving the cursor

Shortcut Key	Effect
h or left arrow key (←)	Moves the cursor back one character.
j or down arrow key (↓)	Moves the cursor down one character.
k or up arrow key (↑)	Moves the cursor up one character.
l or right arrow key (→)	Moves the cursor forward one character.
0	Moves the cursor to the beginning of the current line.
$	Moves the cursor to the end of the current line.
g0	Moves the cursor to the leftmost character of the current line that is on the screen.
g$	Moves the cursor to the rightmost character of the current line that is on the screen.
:n	Moves the cursor to line *n*.
gg	Moves the cursor to the first line of the file.
G	Moves the cursor to the last line of the file.

Data Operations

In Vim **Normal** mode, you can use shortcut keys to copy, paste, and delete content in Vim. The following table lists common shortcut keys.

Table 3-2 Common shortcut keys for data operations

Function	Shortcut Key	Effect
Copy	yy or y	Copies an entire line of text.
	y[n]w	Copies *n* words
Paste	**p** below the current line	Pastes data below the current line.
	p above the current line	Pastes data above the current line.
	p behind the cursor	Pastes data behind the cursor.
	p before the cursor	Pastes data before the cursor.
Deletion/Cut	[n]dd	Deletes (cuts) *n* words.
	d[n]w	Deletes (cuts) *n* lines.

Displaying and Hiding a Line Number

In Vim **Command-line** mode, you can run a command to display the text line number so that you can quickly obtain the corresponding line content.

Examples:

- Run **:set nu** to display a line number.

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

- Run **:set nonu** to hide a line number.

Finding and Replacing

In Vim **Command-line** mode, you can run commands to quickly search for or replace text content. The following is some examples.

- Search for the *word* string after the cursor.

```
:/word
# Press n to search forwards, and Shift+n to search backwards.
```

- Search for the *word* string before the cursor.

```
:?word
# Press n to search forwards.
```

- Replace **word1** in lines 1 to 5 with **word2**.

```
:1,5s/word1/word2/g
# If g is not added, only the first word1 in each line is replaced.
```

- Replace **word1** in a document with **word2** regardless of the case sensitivity.

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

Highlighting Searching Results

In Vim **Command-line** mode, you can highlight searching results temporarily for better reading experience. For permanent setting, add **set hlsearch** in **/etc/vimrc**, and update variables. The following is an example:

- Highlight "hello" in the text:
  - Set highlight in **Command-line** mode:

```
:set hlsearch
```

  - Search for "hello" in in **Command-line** mode:

```
:/hello
# Command output:
```

```
hello
openEuler
hello
world
~
~
:/hello
```

As shown in the preceding information, the repeated "hello" is marked yellow.

- Cancel the highlight:

```
:set nohlsearch
```

Modifying a File

After you run the **vim** *filename* command to open a file, the system enters **Normal** mode. To modify the file, press **i** to enter **Insert** mode. The system displays a message at the bottom, indicating that the current mode is **Insert**. You can press **Ecs** to exit **Insert** mode and return to **Normal** mode. The following is an example:

- Run the following command, and "hello openEuler" is input in the **filename** file:

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

Undoing or Redoing

The following describes how to undo or redo an operation in the inserted text content.

- Input **u** in **Normal** mode to undo the latest change. For example, if the input "hello openEuler" is undid, the following information is displayed:

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

- Input **U** in **Normal** mode to undo all changes at the current line since the cursor has been positioned at the line. The command output is similar to the preceding information.
- Press **Ctrl+R** to redo the last undid change. For example, if the canceled "hello openEuler" is restored, the following information is displayed:

```
[root@openEuler ~]$ vim filename
hello openEuler
~
~
~
```

```
1 more line; after #1  8 seconds ago
```

Saving a File and Exiting

After the compilation, you need to save the file and exit. You first need to exit **Insert** mode by pressing **Esc** to return to **Normal** mode. In **Normal** mode, input **:** to enter **Command-line** mode, and input corresponding commands to save the file and exit.

- **:w** saves a file.
- **:wq** saves and exits a file.
- **:q!** forcibly exits a file but does not save the file. This may cause file loss. Exercise caution when performing this operation.
- **:q** exits a file. It can be used when the file is not edited, but a message is displayed if the file is edited. You can run **:wq** and **:q! to** exit the edited file.
- **wq!** forcibly saves a file and exits.

The Vim **Normal**, **Insert**, and **Command-line** modes can be switched by inputting corresponding commands, as shown in Figure 3-1.

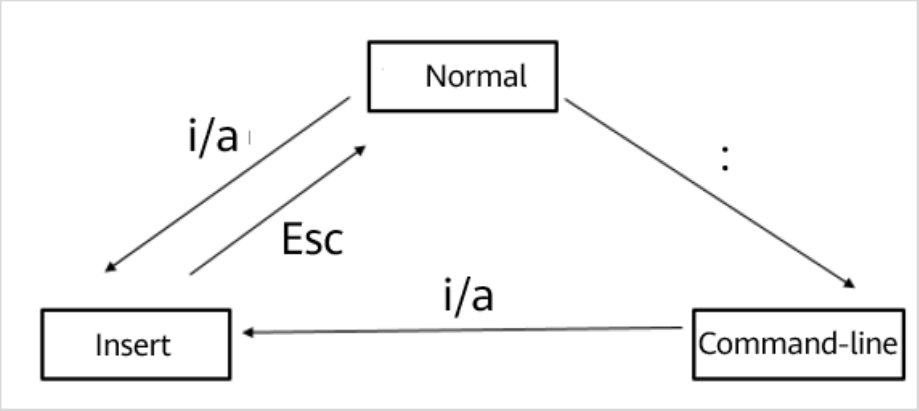


Figure 3-1 Switch between Vim modes

For more information about the help commands, see "openEuler Text Editor" in the *HCIA-openEuler Lab Guide (PC Edition)*.

3.2.2 Text Processing

The following table lists the common commands for text processing in Linux:

Table 3-3 Common text processing commands

Function	Command
File viewing	cat, more, less
File extraction	head, tail
File content extraction	cut, awk, grep
Text sorting and comparison	wc, sort, diff
Text operation	sed, tr

3.2.2.2 File Viewing

Viewing a File - cat

cat is a tool for viewing and connecting text files. The syntax is as follows:

cat [OPTIONS] [FILE]

- **-n**: numbers all lines starting from 1 and displays the number at the beginning of each line.
- **-b**: numbers non-empty lines starting from 1 and displays the number at the beginning of each line.
- **-s**: outputs only one blank line when multiple blank lines exist.
- **-E**: adds **$** at the end of each line.

Examples:

- Display the content of **file1**.

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

- Edit the **file1** file.

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

- Combine **file1** and **file2** into **file3**.

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

- View the **/etc/profile** file content and number non-blank lines starting from 1.

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

- View the **/etc/profile** file content and display the line number at the beginning of each line.

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

- View the **/etc/profile** file content. If multiple blank lines exist, only one is displayed.

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

Viewing a File - more

more can be used to view one file at a time or a standard input page. Unlike **cat**, **more** can be used to view the file content by page or to directly jump to another line. The syntax is as follows:

more [OPTIONS] [FILE]

- **+*n***: displays from the *n*th line.
- **-*n***: defines the screen size to *n* lines.

- **-c**: clears a screen from the top and displays information.
- **-s**: displays multiple consecutive blank lines in one line.

The following table lists the common operations:

Table 3-4 Common operations

Shortcut Key	Function
Enter	Scrolls down one line by default.
Ctrl+F	Scrolls down to the next page.
Space key	Scrolls down to the next page.
Ctrl+B	Scrolls up one page.
b	Scrolls up one page.
=	Outputs the current line number.
:f	Outputs file name and current line number.
q	Exits **more**.

Viewing a File - less

The **less** command is used to read the content and display the content on multiple screens. The **less** command is similar to the **more** command, and both of them can be used to browse files. However, the **less** command can move forwards and backwards and does not load the whole file before displaying the file content. The syntax is as follows:

less [OPTIONS] [FILE]

- **-f**: forcibly opens a special file, such as peripheral device code, directory, or binary file.
- **-g**: specifies only the last found keyword.
- **-i**: ignores case sensitivity during search.
- **-N**: displays the line number of each line.
- **-s**: outputs only one blank line when multiple blank lines exist.
- **-o** <*file name*>: saves the **less** output to a specified file.

The following table lists the common operations:

Table 3-5 Common operations

Shortcut Key	Function
b	Turns to the previous page.
d	Turns to the next half page.
h	Displays the help information.
q	Exits **less**.

u	Turns to the previous half page.
y	Turns to the previous line.
Space key	Turns to the next line.
Enter	Turns to the next page.
↑ and ↓	Turns to the previous or next line, respectively.

3.2.2.3 File Extraction

Extracting a File - head

head is used to display the beginning of a file to the standard output. By default, the first 10 lines of a file are displayed. The syntax is as follows:

head [OPTIONS] [FILE]

- **-q**: hides a file name during output. By default, a file name is not displayed.
- **-v**: displays a file name during output.
- **-c** *num*: displays the first *num* bytes.
- -n *num*: displays the first *num* lines.

Examples:

- Display the first 20 lines in the **/etc/passwd** file.

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

- To display all the lines before the last 20 lines in the **/etc/passwd** file, set the variable to a negative value. For example, if a file contains 37 lines, you can display the first 17 lines by running the following command.

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

Extracting a File - tail

tail is used to display the end of a file to the standard output. By default, the last 10 lines of a file are displayed. The syntax is as follows:

tail [OPTIONS] [FILE]

- **-f**: reads log files cyclically for log management.
- **-q**: hides a file name. File names are not displayed by default.
- **-v**: displays a file name during output.
- **-c** *num*: displays the last *num* bytes of a file.
- **-n** *num*: displays the last *num* lines of a file.

Examples:

- Display the last 20 lines in the **/etc/passwd** file.

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

- To display all the lines after the 20th line in the **/etc/passwd** file, add **+** before the variable value. For example, if a file contains 37 lines, you can display lines 21 to 37 by running the following command.

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

- Display lines 11 to 20 in the **/etc/passwd** file.

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

3.2.2.4 File Content Extraction

Extracting a Column or Field - cut

cut is used to display a specific column of a file or standard input. The syntax is as follows:

cut [OPTIONS] [FILE]

- **-b** [*range*]: displays only the content within the specified range in a line.
- **-c** [*range*]: displays only characters within the specified range in a line.
- **-d**: specifies a field separator. The default field separator is **TAB**.
- **-f** [*range*]: displays the content of the specified *num* th field. Multiple fields can be separated by commas (,).

The specified characters or ranges are expressed in the following format:

- *N*-: starts from the *N*th byte, character, or field to the end.
- *N*-*M*: starts from the *N*th byte, character, or field to the *M*th byte, character, or field (inclusive).
- -*M*: starts from the first byte, character, or field to the *M*th byte, character, or field (inclusive).

Examples:

- Display the first column of the **/etc/passwd** file separated by colons (:).

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

- Display the third to eighth characters in each line in the **/etc/passwd** file.

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

Extracting a Column or Field - awk

awk is a powerful text analysis tool. It reads files or standard input by line, uses spaces as the default separator to slice each line, and performs analysis on each slice. The syntax is as follows:

awk ACTION [FILE]

Examples:

- Display the last five accounts that have logged in to the system.

```
[root@host ~]# last -n 5 | awk '{print $1}'
```

```
# $1 is the variable name, indicating that the printed object is the content in the first column. $2 and $3 are the content in the second and third columns. The rule applies to other $N variables. If the variable is $0, an entire line of data is printed.
```

Extracting Keywords - grep

grep is a powerful text search tool that uses regular expressions to search for text, and then print matched lines. It searches for string templates in one or more files. To find a pattern more than one word long, enclose the text string with single or double quotation marks, otherwise search terms after the first word will be misinterpreted as file names to search in. The search result is sent to standard output, leaving the original file(s) unaltered. The syntax is as follows:

grep [OPTIONS] [FILE]

- **-c**: prints a count of matching lines.
- **-i**: ignores case sensitivity.
- **-w**: prints whole word matches.
- **-x**: prints only results matching the whole line.

Examples:

- Display the line containing openEuler in the **/etc/passwd** file.

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

3.2.2.5 Text Sorting and Comparison

Text Statistics - wc

wc is used to calculate the number of words, bytes, or columns of a file. If the file name is not specified or the given file name is **-**, **wc** reads data from the standard input device. The syntax is as follows:

wc [OPTIONS] [FILE]

- **-c**: displays the number of bytes.
- **-l**: displays the number of lines.
- **-w**: displays the number of words (English letters).

Examples:

- Obtain the number of words, lines, and bytes in the **/etc/passwd** file.

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

Text Sorting - sort

sort is used to sort files and output the sorting result in standard mode. You can use **sort** to obtain input from a specific file or standard input (stdin). The syntax is as follows:

sort [OPTIONS] [FILE]

- **-b**: ignores the space character at the beginning of each line.
- **-c**: checks whether files are sorted in sequence.
- **-d**: processes only letters, digits, and spaces during sorting.

- **-f**: regards lowercase letters as uppercase letters during sorting.
- **-n**: sorts by value.
- **-r**: sorts in reverse order.
- **-o** <*file*>: saves the sorted result to a specified file.
- **-u**: ignores repeated lines.

Text Comparison - diff

diff compares the similarities and differences between text files line by line. If a directory is specified, **diff** compares files with the same file name in the directory but does not compare subdirectories. The syntax is as follows:

diff [OPTIONS] FILE1_or_DIRECTORY1 FILE2_or_DIRECTORY2

- **-B**: does not check blank lines.
- **-b**: does not check white-space characters at each line.
- **-c**: displays all content and marks the differences.
- **-i**: ignores case differences.
- **-r**: compares files in subdirectories.
- **-w**: ignores all space characters.

Examples:

- Compare **f1** with **f2** and list only the differences.

```
[root@host ~]# diff f1 f2
# Different texts of f1 and f2 are listed and separated by ---. The upper part is the different content of f1, and the lower part is the different content of f2.
```

- Compare **f1** with **f2**, display all the content, and mark the differences.

```
[root@host ~]# diff -c f1 f2
# All content of f1 and f2 are listed and separated by ---. The upper part is the content of f1, and the lower part is the content of f2. ! is used to indicate different lines.
```

- Compare the files in subdirectory **d1** with those in subdirectory **d2**.

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

3.2.2.6 Text Operation Tools

tr

tr reads data from a standard input device, converts character strings, and outputs the result to the standard output device. It is used to convert or delete characters in a file. The syntax is as follows:

tr [OPTIONS] SET1 [SET2]

- **-c**: inverts the selection of characters. That is, characters that meet **set1** are not processed, and the remaining characters are converted.
- **-d**: deletes characters.

- **-s**: reduce consecutive repeated characters to a specified single character.
- **-t**: reduces the specified range of **set1** to the length of **set2**.

Examples:

- Convert lowercase letters in the **text.txt** file to uppercase letters.

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

sed

Compared with **tr**, **sed** can modify character strings. **sed**, a streamlined, noninteractive editor, can edit standard input and text from files. When a command is executed, **sed** reads a line from the file or standard input and copies it to buffers. **sed** continues to read each next line until all lines have been edited. As such, **sed** only changes the copied text stored in buffers. To edit the original file directly, use the **-i** option. You can also redirect results to a new file. The syntax is as follows:

sed [OPTIONS] {ACTION} [INPUT_FILE]

- **-n**: cancels the default output.
- **-e**: specifies multipoint editing. Multiple subcommands can be executed.
- **-f**: reads commands from a script file.
- **-i**: directly edits the original file.
- **-l**: specifies the line length.
- **-r**: uses an extended expression in a script.

Examples:

- Display the content of the **/etc/passwd** file, excluding lines 2 to 5.

```
[root@host ~]# cat /etc/passwd | sed '2,5d'
# '2,5d' indicates that lines 2 to 5 are deleted, and d is short for deletion.
```

- Display the content of the **/etc/passwd** file, excluding the content after line 10.

```
[root@host ~]# cat /etc/passwd | sed '11,$d'
# 11,$d' indicates that the content from line 11 to the last line is deleted.
```

- Display the content of the **/etc/passwd** file. However, "openEuler" is added to line 3.

```
[root@host ~]# cat /etc/passwd | sed '2a openEuler'
# a is short for add.
```

- Display the content of the **/etc/passwd** file. "openEuler" is added to line 2, and the original file can be edited.

```
[root@host ~]# cat /etc/passwd | sed -i '2i openEuler'
# i is short for insert. The first i indicates a parameter, which inserts content to the original file. The second i inserts "openEuler" to line 2.
```

- Display the content from lines 10 to 20 in the **/etc/passwd** file.

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

- Display the content of the **/etc/passwd** file. Replace the content in lines 10 to 20 with openEuler.

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

For more examples about file viewing, see "Viewing a File" in the *HCIA-openEuler Lab Guide (PC Edition)*.

3.3 Quiz

- What are the functions of the Vim **Normal**, **Insert**, and **Command-line** modes? How do we switch between these modes?
- What is the difference between **:wq** and **:wq!**? When to use them?
- What are the similarities and differences between **cut** and **grep**?
- How to fix or upgrade files when **diff** is used to find their differences?
- What **sed** commands have the same effect as **head**, **tail**, and **cut** commands?

4 User and Permission Management

On Linux, each user has an account containing information such as a user name, password, and home directory. Some users are created by the user administrator, and some are created by the system for special usages. The most important user is the administrator user, with the default user name **root**. To facilitate managing permissions of different accounts, Linux designs another concept: user group. Each user belongs to at least one group so that users with the same permissions can be easily managed.

User and user group management is an important part of system security management. This chapter describes user and group management commands provided by openEuler and explains how to assign permissions to common users.

4.1 User and User Group Management

4.1.1 User Management

User management in the openEuler system mainly involves adding, modifying, and deleting users.

Adding a user is to create a user in the system and allocate UID, user group, home directory, and login shell to the user.

4.1.1.1 Creating a User

Command: useradd

With the **root** user permission, you can run the **useradd** command to add information about a new user to the system. In the command, [*options*] indicates related information of a user, and *username* indicates the user name.

```
useradd [options] username
```

For example, to create a user named **openEuler**, run the following command as the **root** user:

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

If no prompt is displayed, the user is successfully created. After the user is created, run the **passwd** command to assign a password to the user. A new user without a password cannot log in to the system.

To view information of the new user, run the **id** command:

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

To change the password of user **openEuler**, run the following command:

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

When changing a user password, ensure that the password meets the following complexity requirements:

- A password is a string of at least eight characters.
- A password contains at least any three of the following: uppercase letters, lowercase letters, digits, and special characters.
- A password must be different from the user name.
- A password cannot contain words in the dictionary.

Then, type the password and confirm it as prompted: The process is as follows:

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

- If **BAD PASSWORD: The password fails the dictionary check - it is too simplistic/systematic** is printed in the command output, the password is too simple. Set it with higher complexity.
- Note: In the openEuler environment, you can run the following command to export the dictionary library file **dictionary.txt**. Then, check whether the password exists in the dictionary.

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

4.1.1.2 Modifying a User

Modifying a user involves modifying the user password, shell, home directory, UID, validity period, and other information.

Changing the Password

Common users can change their passwords using the **passwd** command. Only the admin is allowed to use the **passwd *username*** command to change passwords for other users.

Changing User's Login Shell

Common users can change their login shell using the **chsh** command. Only the admin is allowed to run the **chsh *usename*** command to change login shell for other users.

Common users can also run the **usermod** command to modify the shell information. Run the following command as the **root** user. In the command, *new_shell_path* indicates the target shell path, and *username* indicates the name of the user to be modified. Change them to the actual path and name.

```
usermod -s new_shell_path username
```

For example, to change the shell of user **openEuler** to csh, run the following command:

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

Changing the Home Directory

To change the home directory, run the following command as the **root** user. In the command, *new_home_directory* indicates the path of the created target home directory, and *username* indicates the name of the user to be changed. Change them to the actual directory and name.

```
usermod -d new_home_directory username
```

To move the content of the existing home directory to a new directory, use the **-m** option:

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

Changing a UID

Run the following command as the **root** user to change the UID. In the command, *UID* indicates the target user ID, and *username* indicates the user name. Change them to the actual ID and name.

```
usermod -u UID username
```

This command can change a user's UID in all files and directories under the user's home directory. For files outside the home directory, you can only run the **chown** command to manually change the ownership.

Changing the User Validity Period

If a shadow password is used, run the following command as the **root** user to change the validity period of a user. In the command, *MM* indicates the month, *DD* indicates the day, *YY* indicates the year, and *username* indicates the user name. Change them to the actual time and name.

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

4.1.1.3 Deleting a User

You can run the **userdel** command as the **root** user to delete an existing user.

For example, to delete user **Test**, run the following command:

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

If you need to delete a user home directory and all contents in the directory, run the **userdel** command with the **-r** option to delete them recursively.

- **Note**: Do not directly delete a user who has logged in to the system. To forcibly delete a user, run the **userdel -f** *Test* command.

4.1.1.4 Authorizing Administrator Permissions

The **sudo** command allows common users to run commands that can be executed only by the administrator.

The **sudo** command allows users specified in the **/etc/sudoers** file to run commands that can only be executed by the administrator. For example, an authorized common user can run the following command:

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

The **sudo** configuration can specify the dos and don'ts for a common user specified in the **/etc/sudoers** file.

The configuration line of **/etc/sudoers** is as follows:

```
[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
```

- Note: A blank line or commented line (starting with #) has no specific function.

4.1.1.5 User Information Files

The following files contain user information:

- **/etc/passwd**: user account information file.
- **/etc/shadow**: encrypted user account information file.
- **/etc/group**: group information file.
- **/etc/default/useradd**: default setting file.
- **/etc/login.defs**: system wide setting file.
- **/etc/skel**: default initial configuration file.

4.1.2 User Groups Management

Each user in the openEuler system has a user group. The administrator can manage all users in a user group in a centralized manner. User group management in openEuler involves adding, modifying, and deleting user groups.

4.1.2.1 Operations

Adding a User Group

To add a new user group to the system, run the **groupadd** command as the **root** user. In

the command, **options** indicate related parameters, and **groupname** indicates the user group name.

```
groupadd [options] groupname
```

Creating a User Group Instance

To create a user group named **groupexample**, run the following command as the **root** user:

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

Modifying a GID

To change a user group ID, run the following command as the **root** user. In the command, **GID** indicates the target user group ID, and **groupname** indicates the user group.

```
groupmod -g GID groupname
```

Changing a User Group Name

to change the user group name, run the following command as the **root** user. In the command, **newgroupname** indicates the new user group name, and **oldgroupname** indicates the existing user group name to be changed.

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

Deleting a User Group

To delete a user group, run the **groupdel** command as the **root** user.

For example, to delete the user group **Test**, run the following command:

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

- Note: The **groupdel** command cannot delete the primary group of a user. To forcibly delete the primary group of a user, run the **groupdel -f Test** command.

Adding a User to or Removing a User from a User Group

To add a user to or remove a user from a user group, run the **gpasswd** command as the **root** user.

For example, to add the user **openEuler** to the user group **Test**, run the following command:

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

For example, to remove the user **openEuler** from the user group **Test**, run the following command:

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

Switching to a User Group

If a user belongs to multiple user groups, the user can run the **newgrp** command to switch to another user group after logging in to the system so that the user has the permissions of other user groups.

For example, to switch the user **openEuler** to the user **groupTest**, run the following command:

```
[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 User Group Information Files

The files related to the user group information are as follows:

- **/etc/gshadow**: group information encryption file
- **/etc/group**: group information file
- **/etc/login.defs**: system-wide settings

4.2 File Permission Management

4.2.1 Overview

In openEuler, you can run the **ll** or **ls –l** command to display the attributes of a file or directory and file owner or group owner. For example:

```
[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
```

In the preceding example, the first **In** attribute of the **addftinfo** file is represented by a hyphen (-). In Linux, the hyphen indicates that the file is a common file.

In openEuler, the first character indicates that the file is a directory, file, or link file.

Table 4-1 File type description

File Type	Description
-	Common files except the other six types of files

D	Directory
B	Block device file, which is a random access device
C	Character device file, which is a one-time read device such as keyboard and mouse
L	Symbolic link file, which points to another file
P	Named pipe file
S	Socket file

In subsequent characters, three characters are classified into one group combined based on [rwx]. In the combination, [r] indicates read, [w] indicates write, and [x] indicates execute. Note that the order of the three permission bits remains unchanged. If the corresponding permission is not granted, the corresponding letter will be replaced by a minus sign [-].

- Read
- Permission to read the actual content of a file.
- Permission to read the directory structure list.
- Write Permission
- Permission to edit, add, or modify file content.
- Permission to modify, delete, or move files in a directory.
- Execution
- Permission to execute files.
- Permission to access a directory.

In most cases, binary numbers are used to replace the corresponding permission to facilitate operations. For example, 7 indicates rwx.

Figure 4-1 File permission

According to the above figure, the owner permission is 7, the owner group permission is 5, and the permission of other users is 5.

The permission on the directory file is 755.

4.2.2 File Permission Management

4.2.2.1 Overview

The visitors to a Linux file are divided into three categories: file owner, group, and others. The **chmod** command specifies who can visit a file.

Permission: file owner

chmod: changes nine attributes of a file.

openEuler uses numbers or symbols to set file attributes.

4.2.2.2 Number

The openEuler file has nine basic permissions: owner, owner group, and other user roles have their read, write, and execute permissions.

The three permissions of each role can be converted to binary numbers. For example, if the permission is **-rwxrwx---**, the associated numbers are:

- Owner: rwx = 4+2+1 = 7
- Owner group: rwx = 4+2+1 = 7
- Other users: --- = 0+0+0 = 0

Therefore, you can use 770 to change permissions. The syntax of the **chmod** command is as follows:

```
chmod [-R] xyz file or directory
```

Options and parameters:

- *xyz*: permission attribute in numeric type, which is the sum of the values of the rwx attribute.
- **-R**: Changes files and directories recursively. That is, all files in the same directory are changed.

For example, to enable all permissions on the .bashrc file, run the following command:

```
[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
```

What if you want to change the permission to **-rwxr-xr--**? Then the permission associated number is:

```
[4+2+1][4+0+1][4+0+0]=754.
```

4.2.2.3 Symbol

As mentioned above, a file has nine permissions, which belong to the following:

- user: owner
- group: owner group
- others: other users

In this case, **u**, **g**, and **o** can be used to represent the three roles.

In addition, **a** represents all, that is, all roles. The read and write permissions can be written as **r**, **w**, or **x**.

Table 4-2 chmod parameters

Command	User/User Group	Action	Permission	File
chmod	u g o a	+ (add) - (remove) = (set)	r w x	File or directory

To set the file permission to **-rwxr-xr--**, run the **chmod u=rwx,g=rx,o=r** command.

```
[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
```

What if you want to remove a permission without changing other existing permissions? For example, to remove the execute permission of all users, run the following command:

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

chown: changes the owner and owner group of a file.

- Linux is a multi-user and multi-task system, where all files have owners. You can run the **chown** command to change an owner of a specific file to a specified user or group.
- Permission: administrator (**root**)

Syntax:

```
chown [–R] Owner_name File_name
chown [-R] Owner_name:Owner_group_name File_name
```

Examples:

- Go to the **/** directory and change the owner of the **test.txt** file.

```
[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
```

- Change the owner and group owner of the **test.txt** file back to **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: changes the owner group of a file.

You can run the **chgrp** command to change a group to which a file or directory belongs.

Permission: administrator (**root**)

Syntax:

```
chgrp [-cfhRv][--help][--version] [Owner group] [File or directory]
```

Or

```
chgrp [-cfhRv][--help] [--reference=<Reference file or directory>] [--version] [File or directory]
```

Options:

- **-c** or **--changes**: similar to the **-v** option, but returns only the modified part.
- **-f**, **--quiet**, or **--silent**: does not display error information.
- **-h** or **--no-dereference**: modifies only symbolic link files.
- **-R**, **--recursive**: recursively processes all files and subdirectories in a specified directory.
- **-v**, **--verbose**: displays the command execution process.
- **--help**: provides online help.
- **--reference**=<*Reference file or directory*>: sets an owner group of a specified file or directory to be the same as the owner group of a reference file or directory.
- **--version**: shows version number.

Examples:

- Change the group attribute of a file.

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

- The group to which **test.txt** belongs has been changed.

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

umask: mask code

You can run the **umask** command to preset the permission mask when creating a file.

Permission: administrator and common user

Syntax:

```
umask [-S][Permission mask]
```

Options:

- **-S**: specifies a permission mask in text format.

Examples:

- Check the current permission mask.

```
[root@host /]# umask                                    #Obtain the current permission mask.
```

- The command output is as follows:

```
0077
```

- Run the **mkdir** command to create a directory, and then run the **ls** command to obtain detailed information about the directory.

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

- The command output is as follows:

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

- Note: In the preceding output information, **drwxr-xr-x=777-022=755**.

4.2.3 Special File Permissions

In Linux, in addition to the common read, write, and execute permissions, there are three special permissions: SetUID, SetGID, and sticky bit.

- SetUID: If **s** appears in the execute **x** permission of a file owner, it is called SetUID (SUID for short).
- SetGID: If **s** appears in the execute **x** permission of a user group, it is called SetGID (SGID for short).
- Sticky bit: If **t** appears in the execute **x** permission of a directory, it is called sticky bit (SBIT for short).

SUID: enables a common user to temporarily own the identity of the file owner (the prerequisite is that the file is an executable binary file).

In Linux, only **/usr/bin/passwd** has the **s** permission by default.

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

The following example shows setting the SUID permission on a file. If the file is executable, it will execute with the permissions of its owner.

```
[openEuler@host ~]$ ls /root
ls: Failed to open the /root directory. The permission is insufficient.
[openEuler @host ~]# ls -ld /root
```

Common users do not have **root** permission. Therefore, they cannot view the information.

```
[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
```

After the **s** permission is added, the openEuler user has the **root** permission temporarily.

chmod u-s filename: removes the **s** permission of a common user.

chmod u=rws filename can also be set, but **u=rwS**. **S** indicates that the **x** permission is not enabled.

SGID: enables a common user to temporarily becomes a member of the owner group of a file.

```
[openEuler@host ~]$ ls -l /root
ls: Failed to open the /root directory. The permission is insufficient.
[root@host ~]# chmod g+s /usr/bin/ls
[root@host ~]$ ls -l /root
Total usage: 4
```

When SGID is applied to a directory, the directory or file created by any user in the directory belongs to the same group as the directory, as shown in the following.

```
[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
Total usage: 0
  [root@host ~]# chmod g-s abc
[root@host ~]# touch abc/b.txt
[root@host ~]# mkdir abc/b
[root@host ~]# ls -l abc
Total usage: 0
```

SBIT prevents a user except for the **root** user from deleting files of other users.

Syntax: **chmod o+t** *File/Directory*

```
[root@host /]# ls -ld /tmp
drwxrwxrwt 10 root root 240 Dec 14 14:28 /tmp
The permission of other users is rwt, where t indicates the sticky bit.
```

4.2.4 File ACL Permission

4.2.4.1 Overview

The **chmod**, **chown**, **chgrp**, and **umask** commands for common permissions can be used to modify file permissions. Why do we use the Access Control List (ACL)?

- Without using ACL, the visitors to the Linux system are divided into three categories:

file owner, user group, and other users. However, as technology develops, traditional file permission control cannot meet the requirements of complex scenarios. For example, there are multiple employees (such as user01 and user02) in a department (a user group), and different permissions are assigned to employees with different responsibilities. For example, the read and write permissions are assigned to user01, the read-only permission is assigned to user02, and no permissions are assigned to user03. As these employees belong to the same department, employee permissions cannot be further refined. The access control list (ACL) is developed to solve this problem. ACL provides permission settings in addition to common permissions (such as **rwx** and **ugo**), and you can set specific permissions for a single user or a user group.

Common types

- Assign permissions to a file owner.
- Assign permissions to a user group of a file.
- Assign permissions to other users.
- Etc
- ...

4.2.4.2 Operations

On Linux, we can use ACL to manage a file and its specific user and user group permissions. Simply put, an ACL requires only three commands: **setfacl**, **getfacl**, and **chacl**.

- **getfacl**: obtains a file ACL.

ACL Rules

The **setfacl** command can identify the following rule formats:

- **- [d[efault]:] [u[ser]:]uid [:perms]** specifies the permission of the user and the permission of the file owner (if uid is not specified).
- **- [d[efault]:] g[roup]:gid [:perms**] specifies the permission of a group and the permission of owner group of the file (if gid is not specified).
- **- [d[efault]:] m[ask][:] [:perms]** specifies the effective rights mask.
- **- [d[efault]:] o[ther] [:perms]** specifies the permission of others.

Proper ACL rules are used in modify and set operations. You can specify either a name or a number for uid and gid. The **perms** field is a combination of characters that indicate the read (r), write (w), execute (x) permissions. The execute permission applies only to directories and some executable files. The **pers** field can also be set to the octal format.

Automatically Created Rules

Initially, files and directories contain only three base ACL rules. To validate an ACL rule, the following conditions must be satisfied:

- - The three base rules cannot be removed.
- - Whenever an ACL contains named users or named groups, it must also contain an effective permission combination.
- - Whenever an ACL contains any default ACL rules, the default ACL rules must exist.

Definition of ACL

An ACL consists of a series of access entries. Each access entry defines the operation permissions of a specific user type on files.

An access entry consists of three parts:

- - Entry tag type
- - Qualifier (optional)
- - Permission

The following lists some entry tag types:

- **ACL_USER_OBJ**: permission of file owner
- **ACL_USER**: permissions that additional users can have on a file
- **ACL_GROUP_OBJ**: permission of a group
- **ACL_GROUP**: permissions that additional groups can have on a file
- **ACL_MASK**: the maximum permissions of **ACL_USER**, **ACL_GROUP_OBJ**, and **ACL_GROUP**.
- **ACL_OTHER**: permission of others
- **chacl**: changes the ACL of a file or directory.

The **chacl** command is similar to the **chmod** command, but is more powerful and precise. While the **chmod** command specifies who can invoke a file, if a file of a user needs to be viewed only by a specific user, the **chacl** command must be executed to meet the user's requirements.

Example:

```
[root@host ~]# touch /test
[root@host ~]# chmod 777 /test
[root@host ~]# getfacl /test                    //Obtain a file ACL.
getfacl: Removing leading '/' from absolute path names
# file: test                                     //File name
# owner: root                                    //File owner
# group: root                                    //Owner group of the file
user::rwx                                        ///Permission of the file owner
group::rwx                                       //Permissions of users in the same group
other::rwx                                       //Permission of other users
```

Other users have the read, write, and execute permissions. Now, we modify the ACL policy so that the user **code** has only the read permission.

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

Now, look at the ACL attributes of the file again.

```
[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
```

The user permission is not determined only by the ACL configuration. It is determined by the AND operation between the basic permission of the user **code** and the configured ACL permissions, that is, **other:rwx and code:r-- = code:r--**.

Now check whether the user **code** has write permission.

When a file is written, the following information is displayed:

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

ACL can also be set for user groups and effective rights mask. For example, for a user group, you can set **g:[user group]:[rwx]**.

The user **code** has read permission on the **/test** file. If we change the effective rights mask to write-only, the ACL permission is not within the effective rights mask. In this case, the user **code** cannot read the **/test** file.

```
[root@host ~]# setfacl -m m:w /test                    //Set the effective rights mask to write-only.
```

- Check the ACL of **/test**.

```
[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 ~]#
```

- Before reading the file content as the user **code**, write some content as the **root** user to make the test more intuitive.

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

- Cancel the ACL permissions.

```
[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                    //Run properly.
```

4.3 Other Permission Management

4.3.1 Temporary Privilege Escalation

In the Linux OS, the default account is a common user. However, only the **root** user can modify system files or run certain commands.

The following commands are commonly used to switch to the **root** user:

- **su**: switches a user identity to the **root** user. The user is still a common user in the shell environment.
- **su -**: switches a user to the **root** user for both the user identity and shell environment.
- **sudo**: allows common users to execute commands that can be executed only by administrator accounts.

The Linux switch user (**su**) command is used to change the identity of a user except for the **root** user. The password of the user is required.

- Note: In openEuler, a common user cannot switch to the **root** user.

Permission: all users

Syntax:

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

Options:

- **-f** or **--fast**: does not need to read the boot file (such as **csh.cshrc**). This parameter is used only for csh or tcsh.
- **-m**, **-p**, or **--preserve-environment**: specifies that environment variables are not changed when the **su** command is executed.
- **-c command** or **--command=command**: switches to the account *USER* and restores to the original user after the command is executed.
- **-s shell** or **--shell=shell**: specifies the shell to be executed (such as bash, csh, and tcsh). The preset value is the *USER* shell in **/etc/passwd**.
- **--help**: shows the description file.
- **--version**: shows version information.
- **--l** or **--login**: After this option is added, it seems that the user logs in. Most environment variables (such as **HOME SHELL USER**) are owned by the *USER*, and the working directory changes. If the *USER* is not specified, the value is **root**.

- **USER**: indicates the user account after the change.
- ARG: passes new shell parameters.

The Linux **sudo** command is executed by the system administrator. That is, the **sudo** command seems to be executed by the **root** user.

Permission: users who appear in **/etc/sudoers**

Syntax

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

Options:

- **-V**: displays the version number.
- **-h**: displays a version number and command instructions.
- **-l**: displays the permissions of the user who executes the **sudo** command.
- **-v**: confirms a password after **sudo** is executed for the first time, or if it is not executed within *N* minutes (*N* is set to **5** by default). If the execution duration exceeds *N* minutes, password confirmation is also required.
- **-k**: forces a user to enter a password when the **sudo** command is next executed (regardless of whether the user has run the **sudo** command within *N* minutes).
- **-b**: executes a command in the background.
- **-p prompt**: changes the password request prompt. *%u* is replaced with the user's account name. *%h* is replaced with a host name.
- **-u username/#uid**: specifies that if this option is not added, the command is executed as the **root** user. If this option is added, the command is executed as the **username** user. **#uid** indicates the user ID of **username**.
- **-s**: executes a shell specified by the environment variable *SHELL,* or a shell specified in **/etc/passwd**.
- **-H**: specifies the environment variable *HOME* to a home directory of the target user. (If the **-u** is not added, the user is the **root** system administrator.)
- Commands are executed as the system administrator or another user specified by **-u**.

Examples:

- Run the **sudo** command.

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

- Specify a user to run the command.

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

- Show **sudo** settings

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

- Run the previous command as the **root** user.

```
$ sudo
Edit text as a specific user.

$ sudo -u uggc vi ~www/index.html
//Edit the index.html file in the www directory in the home directory as the uggc user.
```

- List the current permissions.

```
sudo -l
```

- List the sudo version information.

```
sudo –V
```

4.4 Quiz

- Create user group **it** whose GID is 1010, and create user group **mg** whose GID is 1020.
- Create a user named **user1**, whose primary group is its private group and secondary group is **it**. Create a user named **user2**, whose primary group is **mg** and secondary group is **it**.
- Set the password validity period of **user1** and **user2** to 30 days and set the notification days before password expiration to 3.
- Create a **/tmp/test.txt** file. The owner of the test file must be **root** and the owner group must be **it**. **user1** has the read permission on the file, and **user2** has the read and write permissions on the file.

5 Software Installation and Service Management

5.1 Software Package Management

Linux software packages are classified into source code and binary packages, which have different formats, for example, RPM and TGZ. RPM and compilation are two primary software installation methods.

RPM is the primary tool for installing software and is easy to use.

Different distributions of OSs such as openEuler, Red Hat, SUSE, and CentOS, also use RPM to manage software packages.

The software packaging format and tool varies according to the related platform. Debian and Ubuntu use the DEB package and apt-get source installation methods to manage software packages. FreeBSD uses the ports and TXZ packaging formats and the make and pkg tools.

5.2 RPM Software Package Overview

RPM is a packaging and automatic installation tool that can download packages from the Internet. It generates .rpm files to install, uninstall, and maintain applications.

Name format:

- **name-version-release.arch.rpm**
- *Software name-version-release version-processor architecture*

Advantages:

- Simple, convenient, and compatible
- Parameter information is recorded in a database for easy software query, upgrade, and uninstallation.

Disadvantages:

- The installation environment must be the same as the packaging environment.
- Dependencies must be processed before you uninstall the software to prevent other software being unusable.

5.2.1 Common RPM Options

RPM has a large number of command options. This document lists only common ones.

Syntax:

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

Main options:

- **-i**: specifies the software package to be installed.
- **-h**: uses a number sign (#) to display the process and progress of RPM installation.
- **-v**: displays details of the installation process.
- **-U**: upgrades a specified software package.
- **-q**: queries whether a specified software package has been installed on the system or queries the content of a specified RPM package.
- **-a**: views all software packages that have been installed on the system.
- **-V**: queries the version of an installed software package.
- **-c**: displays all configuration files.
- **-p**: queries or validates files in a software package.
- **-vh**: displays the installation progress.
- **-qpl**: lists the files in an RPM software package.
- **-qpi**: displays the description of an RPM software package.
- **-qf**: searches for the RPM software package to which a specified file belongs.
- **-Va**: validates all RPM software packages to search for lost files.
- **-qa**: searches for a file, for example, **rpm -qa mysql**.

Example:

- Install software.

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

- Display the software installation information.

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

5.3 DNF Software Management

Many Linux release versions use the Linux software management tool YUM, which can automatically download RPM packages from a specified server and install them. It also serves as a software repository to manage software packages and resolve the dependency problem between software packages, achieving a higher efficiency.

However, YUM delivers poor performance with high memory usage and low speed of dependency parsing. In addition, YUM is very reliant on YUM repo files. If such a file is faulty, YUM-related operations may fail. In this case, the DNF tool is used to overcome the YUM bottlenecks and improve the memory usage, dependency analysis efficiency, and running speed for better use experience.

DNF is a Linux software package management tool that manages RPM software packages. DNF can query software package information, obtain required software packages from a specified software repository, install and uninstall the software packages by automatically processing dependencies, and update the system to the latest available version.

DNF is fully compatible with YUM and provides YUM-compatible command lines and APIs for extensions and plugins.

To use DNF and run commands in this chapter, you must have the administrator permissions.

5.3.1 DNF Configuration File

The main DNF configuration file is **/etc/dnf/dnf.conf**, which consists of the following two parts:

- **main**: stores the global DNF settings.
- **repository**: stores the software source settings. There can be one or more repositories.

The **/etc/yum.repos.d** directory also stores one or more repo source files, which can define different repositories.

Therefore, you can configure the openEuler software source in either of the following ways: (1) directly configure the **repository** part in the **/etc/dnf/dnf.conf** file. (2) Add a repo file to the **/etc/yum.repos.d** directory.

Configuring the main Part

The following is a configuration example of the **main** part in the **/etc/dnf/dnf.conf** file:

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

Table 5-1 Table 5-1 dnf.conf configuration options

Configuration Option	Description
cachedir	Indicates the cache directory, which is used to store RPM packages and database files.
keepcache	Determines whether to cache the RPM packages and header files after a successful installation. The value can be **1** or **0** (default, not cache).
debuglevel	Sets the debug information output by the DNF. The value range is [0-10]. A larger value indicates more detailed debug output. The default value is **2**. The value **0** indicates no debug output.
clean_requirements_on_remove	Deletes dependencies that are no longer used during **dnf remove**. If a software package is

Configuration Option	Description
	installed using DNF instead of an explicit user request, the software package can only be deleted through **clean_requirements_on_remove**. That is, the software package is introduced as a dependency. The default value is **True**.
Best	Always tries to install the latest version during package upgrade. If the latest version cannot be installed, the system displays a cause of the failure and stops installation. The default value is **True**.
obsoletes	Specifies whether to update outdated RPM packages. The value can be **1** or **0**. The default value is **1**, indicating that the update is allowed.
gpgcheck	Specifies whether to perform GPG verification. The value can be **1** or **0**. The default value is **1**, indicating that the verification is required.
plugins	Specifies whether to enable the DNF plugin. The default value is **1**, indicating that the DNF plugin is enabled.
installonly_limit	Specifies the number of packages that can be synchronously installed and are listed in the **installonlypkgs** command. The default value is **3**. You are advised not to decrease the value.

Configuring the repository Part

The **repository** part allows you to customize openEuler repositories, and each repository name must be unique to avoid conflicts. You can configure repositories in either of the following ways: (1) Directly configure the **repository** part in the **/etc/dnf/dnf.conf** file. (2) Configure the repo files in the **/etc/yum.repos.d** directory.

Directly configure the **repository** part in the **/etc/dnf/dnf.conf** file.

- The following is a bare-minimum example:

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

openEuler provides an online image source at https://repo.openEuler.org/. Take the AArch64 version of openEuler 20.03 as an example.

Table 5-2 Table 5-2 Configuration options in the repository part

Configuration Option	Description
name=repository_name	A string describing the repository.
baseurl=repository_url	URL to the directory where the repository is located. • If the repository is available over Hypertext Transfer Protocol (HTTP), the URL is http://path/to/repo. • If the repository is available over File Transfer Protocol (FTP), the URL is ftp://path/to/repo. • If the repository is local to the machine, the URL is file:///path/to/local/repo.

Configuration example:

- openEuler provides multiple repo sources for online use. You can obtain the meaning of each repo source in system installation. The following uses the OS repo source of the AArch64 architecture as an example. Add the openEuler repo source to the **openEuler_aarch64.repo** file as user **root** as follows:

```
# 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
```

- Notes:
- **enabled** indicates whether to enable the software source repository. The value can be **1** or **0**. The default value is **1**, indicating that the software source repository is enabled.
- **gpgkey** is the public key used to verify the signature.

5.3.2 Current Configuration Display

To display the current configurations, run the following command:

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

To display the configuration of a software source, run the following command to query the corresponding repo ID first:

```
dnf repolist
```

Then run the following command, in which **repository** is the queried repo ID.

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

You can also use a global regular expression to view the configurations of all matching sections.

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

5.3.3 Software Package Management

DNF can be used to install, query, and delete software packages.

5.3.3.1 Searching for Software Packages

To search for an RPM package by name, abbreviation, or description, run the following command:

```
dnf search term
```

Example:

```
[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 Listing Software Packages

To list information about all installed and available RPM packages, run the following command:

```
dnf list all
```

To list the installed and available RPM packages that match a particular global regular

expression, run the following command:

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

Example:

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

5.3.3.3 Displaying RPM Package Information

To display information about one or more RPM packages, run the following command:

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

Search example:

```
[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 Installing RPM Packages

To install a single package and all of its non-installed dependencies, run the following command as user **root**:

```
dnf install package_name
```

You can also install multiple packages simultaneously by appending their names as arguments. Add **strict=False** to the **/etc/dnf/dnf.conf** configuration file and run the **dnf** command to add **--setopt=strict=0** to the file. To do so, run the following command as user **root**:

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

Example:

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

- Note: If an RPM package fails to be installed, rectify the fault according to the

installation prompts, such as those for software package conflicts, file conflicts, and package missing.

5.3.3.5 Downloading Software Packages

To use DNF to download a software package, run the following command as user **root**:

```
dnf download package_name
```

To download non-installed dependencies, add **--resolve** and run the following command:

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

Example:

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

5.3.3.6 Deleting Software Packages

To uninstall a particular package and any dependency packages, run the following command as user **root**:

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

Example:

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

5.3.3.7 Managing Software Package Groups

A package group is a collection of packages that serve a common purpose, for instance, **System Tools**. With DNF, you can manage (install/delete) a group of software packages simultaneously, saving time considerably.

Listing Software Package Groups

To view the number of installed groups, available groups, and available environment groups, run the following command with the **summary** option:

```
dnf groups summary
```

Example:

```
[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
```

To list all package groups and their IDs, run the following command:

```
dnf group list
```

Example:

```
[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
```

Displaying Package Group Information

To list mandatory and optional packages contained in a particular group, run the following command:

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

The following is an example of displaying the **Development Tools** information:

```
[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
```

Installing Software Package Groups

Each software package group has a name and an ID (**groupid**). You can install one by its name or ID.

To install a software package group, run the following command as user **root**:

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

For example, to install the software package group of **Development Tools**, run the following commands:

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

Deleting Software Package Groups

To uninstall a software package group, run the following command with the group name or ID as user **root**:

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

For example, to delete the software package group of **Development Tools**, run the following commands:

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

Checking and Updating Packages

DNF checks whether your system has any updates that wait to be applied. You can also use DNF to list the software packages that need to be updated and update them as a whole or update specified packages.

Checking for Updates

To view the updates available to the current system, run the following command:

```
dnf check-update
```

Example:

```
[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
...
```

Upgrading

To upgrade a single software package, run the following command as user **root**:

```
dnf update package_name
```

The following is an example of updating an RPM package:

```
[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]:
```

Similarly, to upgrade a software package group, run the following command as user **root**:

```
dnf group update group_name
```

Update all packages and their dependencies.

To update all packages and their dependencies, run the following command as user **root**:

```
dnf update
```

5.4 Installation Through Source Code

Source code offers an alternative way to install software. On Linux, most software is released as source packages, which are more portable than binary software packages. Only one source package needs to be released for each software and can be executed by different users after compilation. However, the configuration and compilation processes are complex.

RPM is preferred for software installation on openEuler, but source code may also be required in the following scenarios:

- The RPM software package version is outdated, and compilation parameters do not apply to the current service.
- No RPM software package is available for the software to be installed.
- RPM software packages lack certain features.
- Compilation parameters are optimized to boost performance.

Advantages:

- During a compilation process, you can flexibly set parameters as required to install the software.
- After local compilation, the software installed using source code is fully compatible with a local host.

Disadvantages:

- The configuration and compilation processes are complex.
- The dependency package may not exist following a new software installation (or other problems). Consequently, software upgrade is complex and risky.

5.4.1 Procedure for Installation Through Source Code

Procedure for installing software through source code

- Download and decompress a source package and verify its integrity.
- View the **README** and **INSTALL** files, which record software installation methods and precautions.
- Create a makefile by running the **./configure** script.
- Run the **make** command to automatically compile the source code into a binary file.
- Run the **make install** command to install the binary file compiled in the previous step to the corresponding directory. The default installation path is **/usr/local/**, and the corresponding configuration file is located in **/usr/local/etc** or **/usr/local/***/etc**.

Example:

- Download and decompress a source package and verify its integrity.

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

- Go to the source code directory and view the **README** file.

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

- Run the **./configure** command to generate a makefile.

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

- Run the **make** command to start compilation.

```
make/make clean
```

- Run the **make install** command to install the software.

```
make install
```

5.5 Service Management

5.5.1 Service Management Overview

On Linux, systemd is a system and service manager compatible with SysV and LSB initialization scripts. It provides a number of features such as Socket & D-Bus activation of services, on-demand activation of daemons, snapshot and system status restoration, and

mount & automount point management. With systemd, the service control logic and parallelization are refined.

In systemd, the targets of most actions are units. Units are categorized by the type of resources they represent and defined in unit configuration files. For example, the **avahi.service** unit represents the Avahi daemon and is defined in the **avahi.service** file.

5.5.2 Commonly-Used Commands

systemd provides the **systemctl** commands to start, stop, restart, view, enable, and disable system services. The **systemctl** commands of systemd function similar to the **sysvinit** commands. Note that the current version is still compatible with **service** and **chkconfig** commands, but you are advised to use the **systemctl** commands to manage system services.

Table 5-3 Table 5-3 systemd and sysvinit commands

sysvinit	systemd	Remarks
service network start	systemctl start network.service	Starts a service (without restarting an existing one).
service network stop	systemctl stop network.service	Stops a service (without restarting an existing one).
service network restart	systemctl restart network.service	Restarts a service.
service network reload	systemctl reload network.service	Reloads a configuration file without interrupting the wait operation.
service network condrestart	systemctl condrestart network.service	Restarts a service only if it is running.
service network status	systemctl status network.service	Checks the service running status.
chkconfig network on	systemctl enable network.service	Enables a service when the service activation time arrives or a trigger condition for enabling the service is met.
chkconfig network off	systemctl disable network.service	Disables a service when the service activation time arrives or a trigger condition for disabling the service is met.
chkconfig network	systemctl is-enabled network.service	Checks whether a service is enabled.
chkconfig --list	systemctl list-unit-files --type=service	Lists all services in each runlevel and checks whether

sysvinit	systemd	Remarks
		they are enabled.
chkconfig network –list	ls /etc/systemd/system/*.wants/network.service	Lists the runlevels in which a service is enabled and those in which the service is disabled.
chkconfig network –add	systemctl daemon-reload	Reloads the systemd process. Used when you need to create a service file or change settings.

Displaying Service Status

To display the status of a service, run the following command:

```
systemctl status name.service
```

To check whether a service is enabled, run the following command:

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

For example, to display the status of **gdm.service**, run the following command.

```
[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.
```

Starting a Service

To start a service, run the following command as user **root**:

```
systemctl start name.service
```

For example, to start the httpd service, run the following command:

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

Stopping a Service

To stop a service, run the following command as user **root**:

```
systemctl stop name.service
```

For example, to stop the bluetooth service, run the following command:

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

Restarting a Service

To restart a service, run the following command as user **root**:

```
systemctl restart name.service
```

This command stops the selected service and immediately starts it again. If the selected service is not running, this command starts it too.

For example, to restart the bluetooth service, run the following command:

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

Enabling a Service

To configure a service to start automatically at system boot time, run the following command as user **root**:

```
systemctl enable name.service
```

For example, to configure the httpd service to start automatically at system boot time, run the following command:

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

Disabling a Service

To prevent a service from starting automatically at system boot time, run the following command as user **root**:

```
systemctl disable name.service
```

For example, to prevent the bluetooth service from starting automatically at system boot time, run the following command:

```
[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 Basic System Service Management

systemd uses the **systemctl** commands to shut down, restart, and hibernate the OS. Some common Linux management commands are still compatible, but you are advised to use **systemctl** when possible.

Table 5-4 Table 5-4 systemctl and Linux system commands

Common Linux Management Command	systemctl Command	Description
Halt	systemctl halt	Shuts down the system.
Poweroff	systemctl poweroff	Powers off the system.
Reboot	systemctl reboot	Restarts the system.

Shutting Down the System

To shut down and power off the system, run the following command as user **root**:

```
systemctl poweroff
```

To shut down the system without powering it off, run the following command as user **root**:

```
systemctl halt
```

After each of the preceding commands is executed, a message is sent to all login users. If you do not want systemd to send the message, add the **--no-wall** parameter. The command is as follows:

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

Restarting the System

To restart the OS, run the following command as user **root**:

```
systemctl reboot
```

After the preceding command is executed, a message is sent to all login users. If you do not want systemd to send the message, add the **--no-wall** parameter. The command is as follows:

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

Suspending the System

To suspend the system, run the following command as user **root**:

```
systemctl suspend
```

Hibernating the System

To hibernate the system, run the following command as user **root**:

```
systemctl hibernate
```

To suspend and hibernate the system, run the following command as user **root**:

```
systemctl hybrid-sleep
```

5.6 Quiz

- Download the Nginx source code from the **Huawei Cloud Mirrors**, and compile and install Nginx.
- Configure the host firewall service to ensure that the computer can still access the Nginx home page through a web page after the computer is restarted.

6 File System and Storage Management

6.1 Linux Storage Basics

There is an important concept in the Linux system: everything is a file. As a Unix-like OS, Linux inherits the philosophy from Linux, where resources are seen as files, including hardware devices. In Unix, each hardware device is represented as a file called device file, allowing users to access hardware devices by reading and writing files.

Logical volume management (LVM) is a mechanism for managing drive partitions in Linux. It is a logical layer built above drives and partitions and below the file systems, which shields the underlying drive partition layout and improves the flexibility of drive partition management.

The process for managing and partitioning drive space in Linux is as follows:

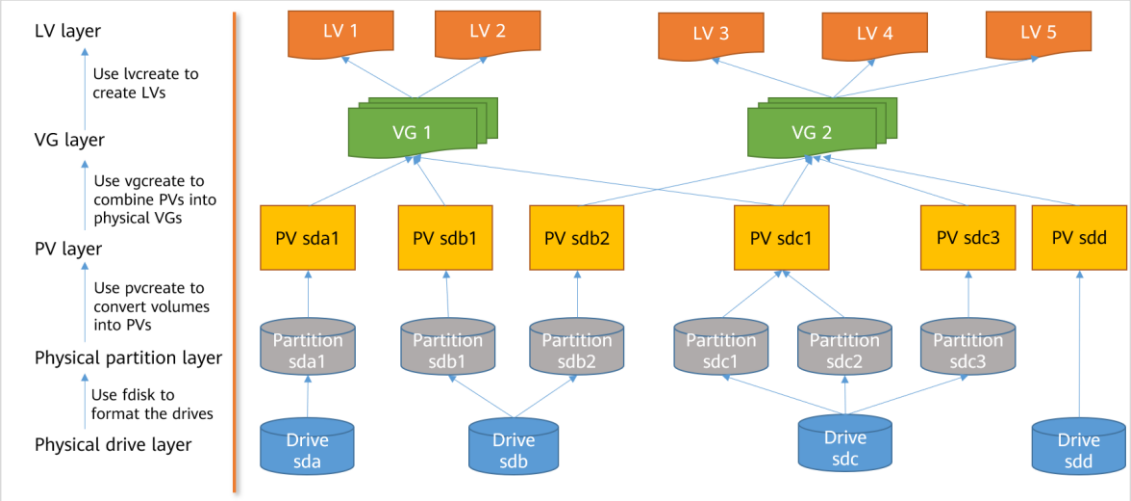


Figure 6-1 Drive space management process

The basic procedure for drive management using LVM is as follows:

- Create partitions
- Create physical volumes based on the partitions
- Combine multiple physical volumes into a volume group
- Create logical partitions in the volume group
- Create file systems based on the logical partitions

With LVM, a file system can be created on top of multiple drives and can be easily resized as needed. This way, the file system size is no longer limited by the underlying drive capacity.

Basic Concepts

Physical media: physical storage device of the system, such as a hard disk drive (HDD). In Linux, files such as **/dev/hda** and **/dev/sda** are storage units at the lowest layer of the storage system.

Logical volume: Linux partitions are different from Windows partitions. In Linux, the drive device name is **hdx** (x ranges from a to d) as a maximum of four IDE drives are supported. SCSI, SATA, and USB drives are **sdx** (x ranges from a to z). A drive can have a maximum of four primary partitions. Therefore, the primary partitions start from **sdb1** to **sdb4**, and the logical partitions start from **sdb5**.

Physical Volume (PV): a drive partition or a device (such as a RAID group) that logically functions the same function as a drive partition. A PV is a basic logical storage block of LVM. A PV has a special label that is stored in the second 512-byte sector by default or one of the first four sectors. This label contains a random unique identifier (UUID) of the PV and records the size of the block device and the storage location of LVM metadata in the device.

Volume group (VG): consists of PVs and shields the details of underlying PVs.

Logical volume (LV): A VG can be used only after it is partitioned into LVs. LVs can be formatted into different file systems and can be directly used after being mounted.

Physical extent (PE): A PV is stored as PEs of the same size. The size of a PE is the same as that of a logical extent in a VG.

Logical extent (LE): LVs are stored as LEs. The LE sizes of all LVs in a VG are the same.

6.1.2 Overview of Drives and Partitions

Physical drives can be classified into hard disk drives (HDDs) and solid state drives (SSDs) based on drive materials. Physical drives can also be classified into SCSI drives, SATA drives, and SAS drives according to the interfaces.

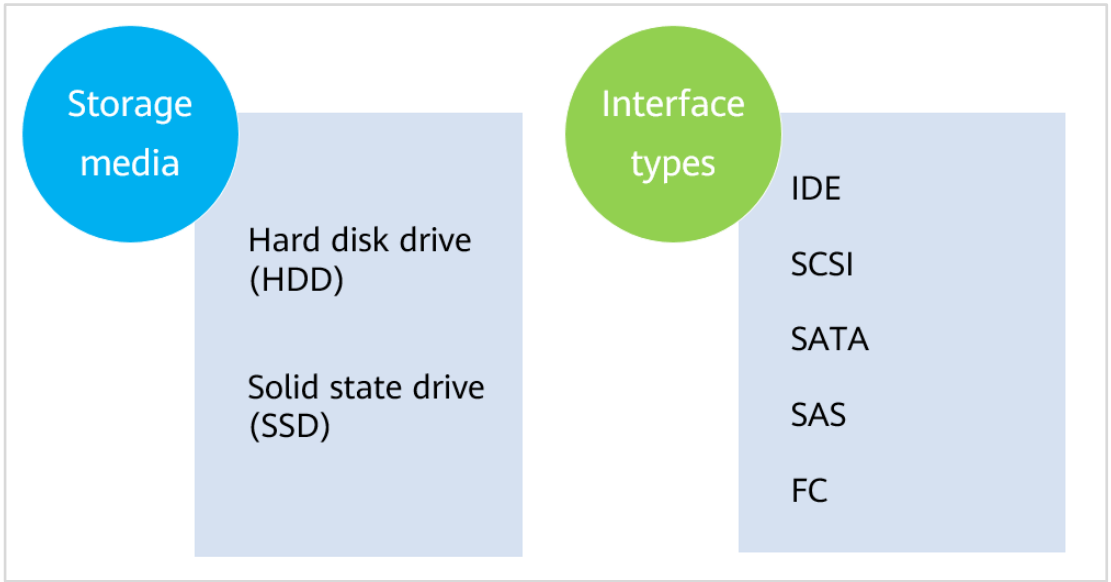


Figure 6-2 Common drive types and interfaces

Through drive partitioning, a drive is divided into multiple logical storage units called partitions. The system administrator can use different partitions for different functions.

Advantages of drive partitioning:

- The available space of applications or users can be restricted.
- The machine can boot into multiple OSs from different partitions on the same drive.
- OS files are separated from program and user files.
- A separate area can be created for OS virtual memory swapping.
- Drive space usage can be restricted to improve the performance of diagnosis tools and image backups.

6.1.3 Physical Partitions

As everything is a file in Linux, you need to access devices through files or directories. Device names are stored in the **/dev** directory.

The devices are named as follows:

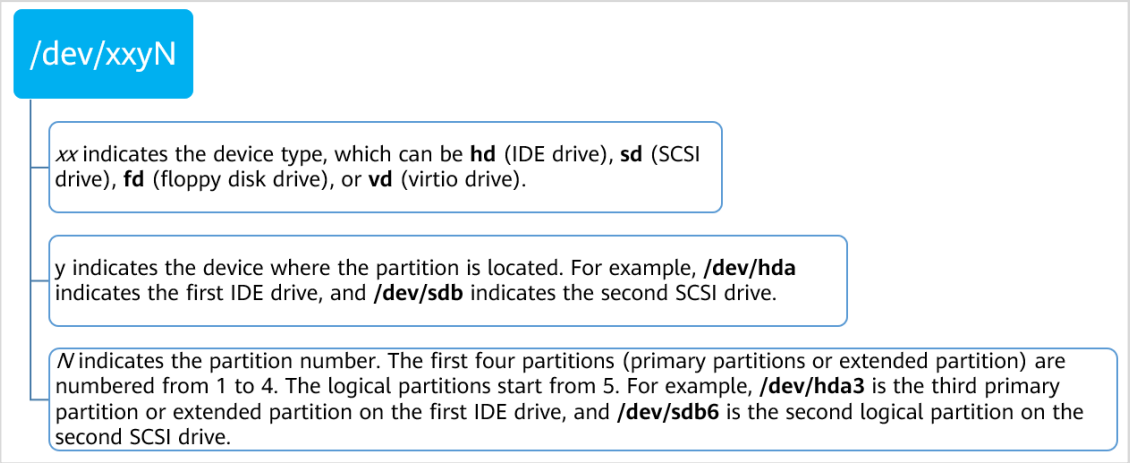


Figure 6-3 Drive naming

In Linux, SAS drives, SATA drives, and SSDs are identified by **sd**, and IDE drives are identified by **hd**.

By partitioning the drive, you are modifying the drive partition table. Pay attention to the following while partitioning:

- For the continuity of data, you are advised to place the extended partition in the last cylinders.
- A drive has only one extended partition. Except the primary partition space, the rest space is allocated to the extended partition.
- Drive capacity = primary partition capacity + extended partition capacity; Extended partition capacity= Sum of the capacities of all logical partitions.

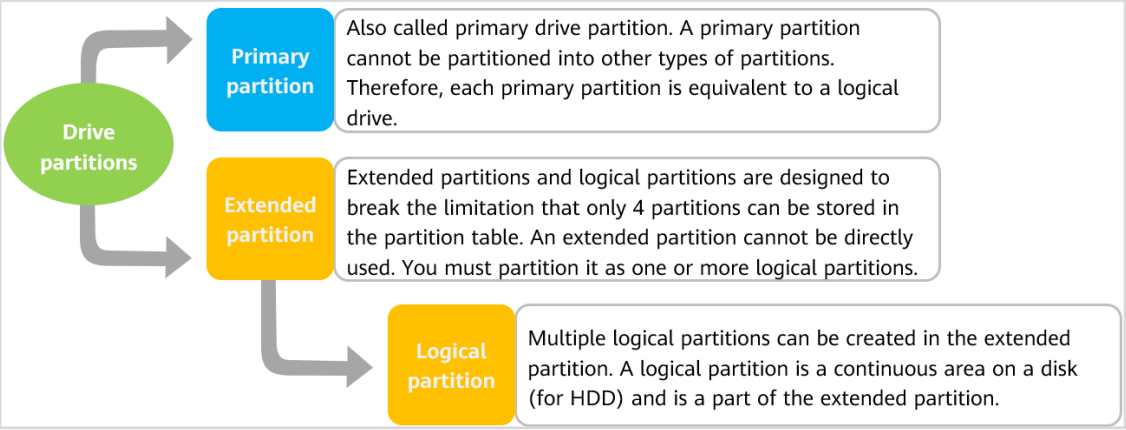


Figure 6-4 MBR partitions

Primary partition

A primary partition is also called a boot partition. The total number of primary partitions and extended partitions of a drive cannot exceed four.

A primary partition can be used immediately after being created, but there is a limit on the number of partitions.

Extended partition

Only one extended partition can be created in a drive.

An extended partition can be divided into any number of logical partitions.

An extended partition cannot be directly used after being created. You need to create logical partitions in the extended partition.

Logical partition

A logical partition is a partition created in an extended partition.

A logical partition is equivalent to a storage medium and is completely independent of primary partitions and other logical partitions.

6.2 LV Layer Management

6.2.1 Installing and Verifying LVM

LVM is installed on openEuler by default. You can run the **rpm -qa | grep lvm2** command to query the installation status. If the command output contains **lvm2**, LVM2 has been installed. If no command output is displayed, LVM2 is not installed. In this case, perform the following steps to install LVM2.

- Install LVM as the **root** user.

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

- View the installed RPM package.

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

6.2.2 Creating Partitions

```
[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       # Primary partition
Partition number (1-4, default 1):        # Enter
First sector (2048-10485759, default 2048):        # Enter
Last sector, +sectors or +size{K,M,G,T,P} (2048-10485759, default 10485759):        # Enter

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

Command (m for help): t       # Change the partitions type. In CentOS 8, you do not need to set the
partition type to 8e.
Selected partition 1
Hex code (type L to list all codes): 8e       # LVM type
Changed type of partition 'Linux' to 'Linux LVM'.

Command (m for help): w       # Save the configuration.
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 Managing Physical Volumes

6.2.3.1 Creating Physical Volumes

Run the **pvcreate** command as the **root** user to create a PV.

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

In the command:

- **option**: specifies the command option. Common options include:
  - **-f**: forcibly creates a PV without user confirmation.
  - **-u**: specifies the UUID of the device.
  - **-y**: answers yes to all questions.
- **devname**: specifies the device file name for the PV to be created. If multiple devices are created, use spaces to separate them.

Example 1: Create **/dev/sdb** and **/dev/sdc** as PVs.

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

Example 2: Create **/dev/sdb1** and **/dev/sdb2** as PVs.

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

6.2.3.2 Viewing Physical Volumes

Run **pvdisplay** as the **root** user to view PV information, including the PV name, VG to which the PV belongs, PV size, PE size, total number of PEs, number of available PEs, number of allocated PEs, and UUID.

```
pvdisplay [option] devname
```

In the command:

- **option**: specifies the command option. Common options include:
  - **-s**: indicates that the output is in short format.
  - **-m**: displays the mapping from the PE to the LE.
- **devname**: specifies the device file name for the PV to be viewed. If no PV name is specified, information about all PVs is displayed.

Example: Display the basic information about the physical volume **/dev/sdb**.

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

6.2.3.3 Modifying Physical Volume Attributes

Run the **pvchange** command as the **root** user to modify the attributes of a PV

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

In the command:

- **option**: specifies the command option. Common options include:
  - **-u**: creates a new UUID.
  - **-x**: indicates whether to allow PE allocation.
- **pvname**: specifies the name of the device corresponding to the PV whose attributes are to be modified. To modify attributes in batches, enter multiple device names and

separate them with spaces.

Example: Prohibit allocation of PEs on the **/dev/sdb** physical volume.

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

6.2.3.4 Deleting Physical Volumes

Run the **pvremove** command as the **root** user to delete a PV.

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

In the command:

- **option**: specifies the command option. Common options include:
  - **-f**: forcibly deletes a PV without user confirmation.
  - **-y**: answers yes to all questions.
- **pvname**: specifies the name of the device corresponding to the PV to be deleted. To delete in batches, enter multiple device names and separate them with spaces.

Example: Delete the physical volume **/dev/sdb**.

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

6.2.4 Managing Volume Groups

6.2.4.1 Creating Volume Groups

Run the **vgcreate** command as the **root** user to create a VG.

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

In the command:

- **option**: specifies the command option. Common options include:
  - **-l**: specifies the maximum number of LVs that can be created on the VG.
  - **-p**: specifies the maximum number of PVs that can be added to the VG.
  - **-s**: specifies the PE size of a PV in the VG.
- **vgname**: specifies the name of the VG to be created.
- **pvname**: specifies the name of the PV to be added to the VG.

Example: Create VG **vg1** and add the PVs **/dev/sdb** and **/dev/sdc** to the VG.

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

6.2.4.2 Viewing Volume Groups

Run the **vgdisplay** command as the **root** user to view the VG information.

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

In the command:

- **option**: specifies the command option. Common options include:
  - **-s**: indicates that the output is in short format.
  - **-A**: displays only the attributes of the active VG.
- **vgname**: specifies the name of the VG to be viewed. If no VG name is specified, information about all VGs is displayed.

Example: Display the basic information about VG **vg1**.

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

6.2.4.3 Modifying Volume Groups Attributes

Run the **vgchange** command as the **root** user to modify the attributes of a VG.

```
vgchange [option] vgname
```

In the command:

- **option**: specifies the command option. Common options include:
  - **-a**: sets the activity status of the VG.
- **vgname**: specifies the name of the VG whose attributes are to be modified.

Example: Change the status of **vg1** to active.

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

6.2.4.4 Extending Volume Group Capacity

Run the **vgextend** command as the **root** user to dynamically expand a VG. The command increases the capacity of a VG by adding PVs to the VG.

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

In the command:

- **option**: specifies the command option. Common options include:
  - **-d**: debugging mode.
  - **-t**: test only.
- **vgname**: specifies the name of the VG whose capacity is to be expanded.
- **pvname**: specifies the name of the PV to be added to the VG.

Example, add the physical volume **/dev/sdb** to the VG **vg1**.

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

6.2.4.5 Reducing Volume Group Capacity

Run the **vgreduce** command as the **root** user to delete PVs from a VG to reduce the VG capacity. The last remaining PV in the VG cannot be deleted.

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

In the command:

- **option**: specifies the command parameter option. Common parameter options are as follows:
  - **-a**: If the PV to be deleted is not specified in the command line, all empty PVs are deleted.
  - **--removemissing**: deletes the lost physical volumes from the volume group to restore the volume group to the normal state.
- **vgname**: specifies the name of the VG whose capacity is to be shrunk.
- **pvname**: specifies the name of the PV to be deleted from the VG.

Example: Remove the physical volume /dev/sdb2 from the volume group vg1.

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

6.2.4.6 Deleting Volume Groups

Run the **vgremove** command as the **root** user to delete a VG.

```
vgremove [option] vgname
```

In the command:

- **option**: specifies the command parameter option. Common parameter options are as follows:
  - **-f**: forcibly deletes a VG without user confirmation.
- **vgname**: specifies the name of the VG to be deleted.

Example: Delete the VG **vg1**.

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

6.2.5 Managing LVs

6.2.5.1 Creating LVs

Run the **lvcreate** command as the **root** user to create a LV.

```
lvcreate [option] vgname
```

In the command:

- **option**: specifies the command parameter option. Common parameter options are as follows:
  - **-L** specifies the size of the LV. The unit can be K, M, G, or T (case-insensitive).
  - **-l**: specifies the size of the LV (number of LEs).
  - **-n**: specifies the name of the LV to be created.

  - **-s:** indicates to create snapshots.
- **vgname**: specifies the name of the VG where the LV is to be created.

Example 1: Create a 10 GB LV in the VG **vg1**.

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

Example 2: Create a 200 MB LV in the VG **vg1** and name it **lv1**.

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

6.2.5.2 Viewing LVs

Run the **lvdisplay** command as the **root** user to view the LV information, including the LV space size, read/write status, and snapshot information.

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

In the command:

- **option**: command parameter option. Common parameter options are as follows:
  - **-v**: displays the mapping from the LE to the PE.
- **lvname**: specifies the device file corresponding to the LV whose attributes are to be displayed. If no LV is specified, all LV attributes are displayed.

Note: The device file corresponding to the LV is stored in the VG directory. For example, if the LV **lv1** is created in the VG **vg1**, the device file corresponding to the LV is **/dev/vg1/lv1**.

Example: Display the basic information about the LV **lv1**.

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

6.2.5.3 Adjusting LV Capacity

Run the **lvresize** command as the **root** user to increase or decrease the sizes of the LVs. Exercise caution when running the **lvresize** command to adjust the sizes of the LVs because data may be lost.

```
lvresize [option] vgname
```

In the command:

- **option**: specifies the command parameter option. Common parameter options are as follows:
  - **-L** specifies the size of the LV. The unit can be K, M, G, or T (case-insensitive).
  - **-l**: specifies the size of the LV (number of LEs).
  - **-f**: forcibly adjusts the size of an LV without user confirmation.
- **lvname**: specifies the name of the LV to be adjusted.

Example 1: Add 200 MB space to the LV **/dev/vg1/lv1**.

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

Example 2: Reduce the space of the **/dev/vg1/lv1** LV by 200 MB.

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

6.2.5.4 Extending LV Capacity

Run the **lvextend** command as the **root** user to dynamically expand the space of an LV online without interrupting the access of applications to the LV.

```
lvextend [option] lvname
```

In the command:

- **option**: specifies the command parameter option. Common parameter options are as follows:
  - **-L** specifies the size of the LV. The unit can be K, M, G, or T (case-insensitive).
  - **-l**: specifies the size of the LV (number of LEs).
  - **-f**: forcibly adjusts the size of an LV without user confirmation.
- lvname: specifies the device file of the LV whose space is to be expanded.

Example: Add 100 MB space to the LV /dev/vg1/lv1.

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

6.2.5.5 Reducing LV Capacity

Run the **lvreduce** command as the **root** user to reduce the space occupied by the LV. If you run the **lvreduce** command to reduce the space of an LV, the existing data on the LV may be deleted. Therefore, you must confirm the operation before running the **lvreduce** command.

```
lvreduce [option] lvname
```

In the command:

- **option**: specifies the command parameter option. Common parameter options are as follows:
  - **-L** specifies the size of the LV. The unit can be K, M, G, or T (case-insensitive).
  - **-l**: specifies the size of the LV (number of LEs).
  - **-f**: forcibly adjusts the size of an LV without user confirmation.
- **lvname**: specifies the device file of the LV whose space is to be expanded.

Example: Reduce the space of the LV **/dev/vg1/lv1** by 100 MB.

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

6.2.5.6 Deleting LVs

Run the **lvremove** command as the **root** user to delete an LV. If an LV has been mounted by running the **mount** command, you cannot run the **lvremove** command to delete it. You must run the **umount** command to unmount the LV first.

```
lvremove [option] vgname
```

In the command:

- **option**: specifies the command parameter option. Common parameter options are as follows:
  - -f: forcibly deletes an LV without user confirmation.
- **vgname**: specifies the LV to be deleted.

Example: Delete the LV **/dev/vg1/lv1**.

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

6.2.6 Creating and Mounting File Systems

After the LV is created, you need to create a file system on the LV and mount the file system to a directory.

6.2.6.1 Creating a File System

Run the **mkfs** command as the **root** user to create a file system.

```
mkfs [option] lvname
```

In the command:

- **option**: specifies the command parameter option. Common parameter options are as follows:
  - **-t**: specifies the type of the Linux file system to be created, such as **ext2**, **ext3**, and **ext4**. The default type is **ext2**.
- **lvname**: specifies the device file name of the LV corresponding to the file system to be created.

Example: Create an ext4 file system on the LV **/dev/vg1/lv1**.

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

6.2.6.2 Manually Mounting a File System

A manually mounted file system becomes valid after the OS is restarted. You can run the **mount** command to mount the file system as the **root** user.

```
mount lvname mntpath
```

In the command:

- **lvname**: specifies the device file name of the LV to which the file system is to be

mounted.

- **mntpath**: specifies the mount path.

Example: Mount the LV **/dev/vg1/lv1** to the **/mnt/data** directory.

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

6.2.6.3 Automatically Mounting a File System

A manually mounted file system becomes invalid after the OS is restarted, and you need to mount the file system again. However, you can perform the following operations as the **root** user after manually mounting the file system, so that the file system will be automatically mounted when the OS is restarted.

Run the **blkid** command to query the UUID of the LV. **/dev/vg1/lv1** is used as an example.

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

Check the printed information. The printed information contains the following information, in which **uuidnumber** is a string of digits, indicating the UUID, and **fstype** indicates the file system.

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

Run the **vi /etc/fstab** command to edit the **fstab** file and add the following content to the end of the file:

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

The file content is described as follows:

- The first line indicates the UUID. Enter the UUID queried in the previous step.
- The second line indicates the mount directory **mntpath** of the file system.
- The third line indicates the type of the file system. Set this parameter to **fstype** queried in the previous step.
- The fourth line indicates the mounting option. In this example, **defaults** is used.
- The fifth line indicates the backup option. Enter either **1** (the system automatically backs up the file system) or **0** (the system does not back up the file system). In this example, **0** is used.
- The sixth line indicates the scanning option. Enter either **1** (the system automatically scans the file system at system start) or **0** (the system does not scan the file system). In this example, **0** is used.

Verify the automatic mounting function.

- Run the **umount** command to unmount the file system. Assume that the LV is **/dev/vg1/lv1**.

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

- Run the following command to remount the file systems corresponding to the

content in the **/etc/fstab** file:

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

- Run the following command to query the mounting information about the file system. The mount directory **/mnt/data** is used as an example.

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

- Check the command output. If the command output contains the following information, the automatic mounting function takes effect:

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

6.3 Quiz

- You need to consider security of LVs. Create an LVM that prevents data loss when a drive at the bottom layer is damaged.
- Configure an LVM that contains 100 LEs., and the size of each PE is 8 MB.

7 System Management

7.1 Task Management

During Linux system O&M, you need to configure specific tasks to perform a certain action at a specific time or periodically. This chapter describes how to manage tasks.

7.1.1 Task Executed Once: at

7.1.1.1 Introduction to at

You can use **at** to set an absolute time written in any of the following formats: *hh:mm* (hour:minute) on the current day (If the time has already passed, the job will be executed the following day); **midnight**, **noon**, **teatime** (16:00); 12-hour time followed by **am** or **pm**; or time + date (*month day*, *mm/dd/yy*, or *dd.mm.yy*). The specified date must be followed by the specified time.

Alternatively, you can set a relative time written in the following format: **now+***N* **minutes**, **hours**, **days**, or **weeks**, where *N* indicates the number of minutes, hours, days, or weeks;

Another method is to use **today** and **tomorrow** to specify the time to complete commands.

There may be two files used to restrict the use of the **at** command in the **/etc** directory of the system. One is the blocklist (**etc/at.deny**) and the other is the allowlist (**/etc/at.allow**). Generally, only the blocklist file exists because the **at** command can be executed in most cases and you only need to write one or two blocked users in the blocklist.

- If the **/etc/at.allow** file exists in the system, only the users written in the file can run the **at** command (one username per line). In this case, the **/etc/at.deny** file is ignored.
- If only the **/etc/at.deny** file exists, the users written in the file cannot run the **at** command. The blocklist does not take effect for the **root** user.
- If neither of the two files exists, only the **root** user can run the command.

7.1.1.2 Example of Using at

Run the **at** commands to create the following scheduled tasks:

- Run the **/bin/ls** command at 5 p.m. three days later.

```
[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
```

- Export the time to the specified file at 17:20 tomorrow.

```
[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 Periodic Task: crond

Task scheduling in Linux is classified into system task scheduling and user task scheduling. Linux system tasks are controlled by the cron (crond) system service, which is started by default. You can run the **crontab** command to set scheduled tasks.

cron is one of the most practical tools in Linux or Unix-like systems. The cron service (daemon process) runs in the background and continuously checks the **/etc/crontab** file and the **/etc/cron.*/** directory. It also checks the **/var/spool/cron/** directory. Each user can have their own crontab files. Although these files are stored in the **/var/spool/cron/crontabs** directory, they are not intended to be edited directly. You need to run the **crontab** command to edit or configure your own scheduled tasks.

crontab Command

The cron service provides the **crontab** command. The options of the **crontab** command are described as follows:

- **crontab -u** // Set a user's cron service. This option is required only when the crontab command is run by the root user.
- **crontab -l** // List details about a user's cron service.
- **crontab -r** // Remove a user's cron service.
- **crontab -e** // Edit a user's cron service. (Press **X** to delete the character selected by the cursor, press **I** to insert the character at the selected position, run **dd** to delete the current line, and press **P** to paste the character.)
- **:wq!** + **Enter** // Save and exit.
- **:q!** + **Enter** // Exit without saving the settings.

crontab Editing Syntax

Run the **crontab -e** command to enter the edit mode and set a periodic task. The syntax is as follows:

- 1 2 3 4 5 /path/to/command arg1 arg2
- Where
- Field 1: minute (0-59)
- Field 2: hour (0-23)
- Field 3: date (0-31)
- Field 4: month (0-12), where 12 indicates December.
- Field 5: a day in a week (0-7), where 7 or 0 indicates Sunday.
- */path/to/command*: name of the script or command to be executed

A few simple crontab examples are as follows:

- Run the **backupscript.sh** script every five minutes.

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

- Run the **backupscript.sh** script at 01:00 every day.

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

- Run the **backupscript.sh** script at 3:15 on the first day of each month.

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

Operators

Operators are used to specify multiple values for a field. There are three operators available:

- Asterisk (*): This operator specifies all available values for a field. For example, in the hour field, an asterisk mean every hour; in the month field, an asterisk means every month.
- Comma (,): This operator specifies a list containing multiple values, for example, **1,5,10,15,20,25**.
- Hyphen (-): This operator specifies a value range, for example, **5-15**, which is the same as **5,6,7,8,9,10,11,12,13,14,15** entered using the comma (,) operator.
- Forward slash (/): This operator specifies a step value. For example, **0-23/** can be used in the hour field to specify that a command is executed every hour. The step value can also be followed by the asterisk (*) operator. If you want a command line to be executed every two hours, use ***/2**.

7.2 Network Management

7.2.1 OSI Protocol Stack (Extended)

OSI is short for Open System Interconnection. The International Organization for Standardization (ISO) has formulated the OSI model, which defines the standards for the interconnection of different computers and is the basic framework for designing and describing computer network communication. The OSI model divides network communication into seven layers: physical layer, data link layer, network layer, transport layer, session layer, presentation layer, and application layer.

Physical Layer

Data format: bit stream.

Main functions and connection modes: establishing, maintaining, and canceling physical connections.

Typical devices: optical fibers, coaxial cables, twisted pairs, NICs, repeaters, and hubs.

- Description: In the OSI reference model, the physical layer is the lowest layer of the reference model and the first layer of the OSI model. The main function of the physical layer is to use transmission media to provide physical connections for the data link layer and transparently transmit bit streams. The physical layer

transparently transmits bit streams between adjacent computer nodes and shields the differences between transmission media and physical devices. The data link layer does not need to consider the specific transmission media of the network. Transparent transmission of bit streams indicates that the bit streams transmitted through the actual circuit do not change. The circuit is invisible for the transmitted bit streams.

Data Link Layer

Data format: data frames encapsulated from bit streams.

Main functions and connection modes: establishing, canceling, and identifying logical links at the physical layer, multiplexing links, and checking errors, as well as addressing by using the hardware address or physical address of the receiving system.

Typical devices: bridges and switches.

- Description: The data link layer is the second layer of the OSI model and is responsible for establishing and managing links between nodes. The main function of this layer is to convert error physical channels into error-free data links that can reliably transmit data frames through various control protocols.

This layer is usually divided into two sublayers: Media Access Control (MAC) and Logical Link Control (LLC).

- The main task of the MAC sublayer is to solve the problem of multi-user channel competition in shared network and complete the access control of network media.
- The main task of the LLC sublayer is to establish and maintain network connections, and perform error check, flow control, and link control.

The data link layer receives bit streams from the physical layer, encapsulates the bit streams into data frames, and transmits the data frames to the upper layer. Similarly, the data link layer disassembles the data frames from the upper layer into bit streams and forwards the bit streams to the physical layer. In addition, it is also responsible for processing the information of the acknowledgment frame returned by the receive end, to provide reliable data transmission.

Network Layer

Data format: split and reassembled data packets.

Main functions and connection modes: path selection between different network systems based on network layer addresses (IP addresses).

Typical devices: gateways and routers.

- Description: The network layer is the third layer of the OSI model and is the most complex layer in the OSI reference model. Its main task is to select the most appropriate path for messages or packets through the communication subnet based on the route selection algorithm. This layer controls information forwarding between the data link layer and the transport layer, and establishes, maintains, and terminates network connections. Specifically, data at the data link layer is converted into data packets and then transmitted from one network device to another through control such as path selection, segment combination, sequencing, and incoming/outgoing routing. Generally, the data link layer is used for communication between nodes on the same network, and the network layer is used for communication between different subnets. For example, when communication is

performed between WANs, a route selection problem (that is, there may be multiple paths between two nodes) is inevitably encountered.

To implement network layer functions, the following problems need to be solved:

- Addressing: Physical addresses (such as MAC addresses) used at the data link layer solve only the addressing problem within the network. When different subnets communicate with each other, a unique address is assigned to each device on the subnet to identify and find the device on the network. Subnets may use different physical technologies. Therefore, this address should be a logical address (such as an IP address).
- Exchange: Different information exchange modes are specified. Common switching technologies include line switching and store-and-forward switching. The store-and-forward switching includes message switching and packet switching.
- Routing algorithm: When multiple paths exist between the source node and the destination node, this layer can select the optimal path for data packets through the network based on the routing algorithm, and transmit the information from the most appropriate path to the receive end.
- Connection service: Different from the data link layer which controls the traffic between adjacent nodes on the network, the network layer controls the traffic from the source node to the destination node. The purpose is to prevent blocking and detect errors.

Transport Layer

Data format: data segments.

Main functions and connection modes: using an addressing mechanism to identify a specific application (port number).

Typical devices: terminal devices (such as PCs, mobile phones, and tablets).

- Description: The main task of the lower three layers of the OSI model is data communication, and the task of the upper three layers is data processing. The transport layer is the fourth layer of the OSI model. This layer is the interface and bridge between the communication subnet and the resource subnet. The main task of this layer is to provide reliable end-to-end error and flow control for users to ensure correct packet transmission. It shields the details of the lower-layer data communication from the upper layer, to transparently transmit packets to users. Common protocols at this layer include Transmission Control Protocol (TCP) in TCP/IP, Sequenced Packet Exchange (SPX) used by Novell Netware, and Network Basic Input/Output System (NetBIOS) and NetBIOS Extended User Interface (NetBEUI) of Microsoft.

The transport layer provides the transmission service between the session layer and the network layer. The service obtains data from the session layer and segments the data if necessary. The transport layer then passes the data to the network layer and ensures that the data can be correctly transmitted to the network layer. Therefore, the transport layer provides reliable data transmission between two nodes. After the connection between the two nodes is determined, the transport layer will monitor the connection. In conclusion, the main functions of the transport layer are described as follows:

- Transmission connection management: establishing, maintaining, and removing transmission connections. Based on the network layer, the transport layer provides

connection-oriented and connectionless services for upper layers.

- Transmission error processing: reliable connection-oriented and less reliable connectionless data transmission services, error control, and flow control. When the connection-oriented service is provided, the data transmitted through this layer is acknowledged by the target device. If no acknowledgment is received within the specified time, the data is retransmitted.
- Service quality monitoring

Session Layer

Data format: DTPUs.

Main functions and connection modes: mapping from the session layer to the transport layer, flow control of session connections, data transmission, session connection recovery and release, session connection management, and error control.

Typical devices: terminal devices (such as PCs, mobile phones, and tablets).

- Description: The session layer is the fifth layer of the OSI model. It is the interface between user applications and the network. The main task is to provide the presentation layer of two entities with the method of establishing and using connections. The connection of the presentation layer between different entities is called a session. The task of the session layer is to organize and coordinate the communication between two session processes, and manage the data exchange. Users can set up sessions in half-duplex, simplex, or full-duplex mode. When establishing sessions, users must provide the remote addresses they want to connect to. Those addresses are different from MAC addresses or logical addresses at the network layer. They are specially designed for users and are easy to remember. A domain name (DN) is a remote address used on a network.

The functions of the session layer are described as follows:

- Session management: allowing users to establish, maintain, and terminate sessions between two entity devices, as well as data exchange between them. For example, it allows users to provide a unidirectional session or a bidirectional simultaneous session, and manage the sending sequence in the session and the duration of the session.
- Session traffic control: session traffic control and cross-session functions.
- Addressing: using a remote address to establish a session connection.
- Error control: Logically, the session layer is responsible for establishing, maintaining, and terminating data exchange. However, the actual work is to receive data from the transport layer and correct errors. Session control and remote procedure calls are functions of this layer. It should be noted, however, that the error checked at this layer is not a communication media error, but a high-level error such as the drive space error or printer paper shortage.

Presentation Layer

Data format: PTPUs.

Main functions and connection modes: data representation, data security, and data compression.

Typical devices: terminal devices (such as PCs, mobile phones, and tablets).

- Description: The presentation layer is the sixth layer of the OSI model. It interprets the commands and data from the application layer, assigns meanings to various syntaxes, and transmits the syntaxes to the session layer in a certain format. Its main function is to process the representation of user information, such as encoding, data format conversion, encryption, and decryption.

The functions of the presentation layer are as follows:

- Data format processing: negotiating and establishing the format for data exchange to resolve the differences in data format representation between applications.
- Data encoding: converting character sets and numbers. For example, a data type (integer or real, signed or unsigned, or the like) and a user identifier in a user program may have different representation manners. Therefore, a function of converting between different character sets or formats is required between devices.
- Compression and decompression: To reduce the amount of data to be transmitted, this layer is also responsible for data compression and restoration.
- Data encryption and decryption: improving network security.

Application Layer

Data format: ATPUs.

Main functions and connection modes: an interface between network services and user applications.

Typical devices: terminal devices (such as PCs, mobile phones, and tablets).

- Description: The application layer is the highest layer of the OSI reference model. It is the interface between computer users and various applications and the network. It directly provides services for users and completes the tasks that users want to complete on the network. Based on the work of the other six layers, it completes the connection between the application program and the network OS, establishing and terminating the connection between users, and completing various network services proposed by network users and various protocols such as supervision, management, and service required by applications. In addition, this layer coordinates the work between applications.

The application layer provides the following services and protocols for users: file service, directory service, file transfer service (FTP), remote login service (Telnet), email service, printing service, security service, network management service, and database service. The preceding network services are implemented by different application protocols and programs at this layer. Different network OSs differ greatly in terms of functions, interfaces, implementation technologies, hardware support, security and reliability, and application programming interfaces (APIs). The application layer provides the following functions:

- User interface: The application layer is a direct interface between users and the network and between applications and the network. It enables users to interact with the network.
- Implementing various services: Various applications at this layer can implement various services requested by users.

The seven-layer OSI model is too ideal and is seldom used in actual production environments. Most applications are designed and implemented based on the TCP/IP protocol stack. In the seven-layer model, each layer provides a special network function.

From the perspective of network functions, the lower four layers (physical layer, data link layer, network layer, and transport layer) mainly provide data transmission and switching functions, that is, communication between nodes. Layer 4 functions as a bridge between the upper and lower parts and is the most important part of the entire network architecture. The upper three layers (session layer, presentation layer, and application layer) mainly provide information and data processing functions between users and applications. In short, the lower four layers implement the functions of the communication subnet, and the upper three layers implement the functions of the resource subnet.

7.2.2 IPv4 Address

An IP address is a logical address, which is different from a MAC address. Its composition is as follows:

- IP address: network ID + host ID

An IPv4 address is a 32-bit number. It is typically written in decimal digits, formatted as four 8-bit octets (between 0 and 255) that are separated by dots. Therefore, the IPv4 address format is also called dot-decimal notation.

The value range of dot-decimal notation is as follows:

- 0.0.0.0–255.255.255.255

7.2.3 ifcfg and iproute

7.2.3.1 Network Interface Naming Conventions

Traditional Naming Conventions

Ethernet: eth*X*, where *X* ranges from [0,oo). For example, **eth0**, **eth1**, and so on.

Point-to-Point Protocol (PPP) network: ppp*X*, where *X* ranges from [0, ...]. For example, **ppp0**, **ppp1**, and so on.

Predictable Naming Conventions (openEuler)

Naming conventions based on the firmware and topology:

- If the index information provided by the firmware or BIOS for the devices integrated on the main board is available, the devices are named based on the index, such as **eno1**, **eno2**, and so on.
- If the index information provided by the firmware or BIOS for the PCIe slots is available and predictable, the PCIe slots are named based on the index, such as **ens1**, **ens2**, and so on.
- If the physical location information of the hardware interfaces is available, the hardware interfaces are named based on the information, for example, **enp2s0**.
- Users can explicitly define names based on the MAC addresses, for example, **enx122161ab2e10**.

If none of the preceding methods is available, use the traditional naming conventions.

7.2.4 Overview of NetworkManager

In openEuler 20.03 LTS, the default networking service is provided by NetworkManager, which is a dynamic network control and configuration daemon to keep network devices

and connections up and active when they are available. The traditional **ifcfg** type configuration files are still supported.

Table 7-1 Networking application and utility

Application or Utility	Description
NetworkManager	Default network connection daemon process
nmcli	A command-line utility that allows users and scripts to interact with NetworkManager

NetworkManager supports the following connection types: Ethernet, VLAN, bridge, bonding, group, Wi-Fi, mobile broadband (such as mobile network 3G), and IP-over-InfiniBand. NetworkManager can configure network aliases, IP addresses, static routes, DNS information, and VPN connections, as well as many connection-specific parameters. In addition, NetworkManager provides an API through D-bus, which allows applications to query and control network configuration and state.

The nmcli utility can be used by both users and scripts for controlling NetworkManager. The basic format of a nmcli command is as follows:

- nmcli OPTIONS OBJECT { COMMAND | help }

Where **OBJECT** can be **general**, **networking**, **radio**, **connection**, or **device**. The most commonly used optional **OPTIONS** are **-t**, **--terse** (for scripts), **-p**, **--pretty** (for users), and **-h**, **--help**. If you are not sure about the available command options, you can press **Tab** to view the available command options.

The nmcli utility has some built-in context-sensitive help. For example, run the following commands and pay attention to the differences:

```
[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>]
```

In the second example above, the help information is related to the **general** object.

The following shows how to use nmcli.

```
# Display the overall status of NetworkManager.
nmcli general status
# Control NetworkManager logging.
nmcli general logging
# Display all connections.
nmcli connection show
# Add -a/--active to display only the current active connections.
nmcli connection show --active
# Display the devices identified by NetworkManager and their status.
nmcli device status
# Simplify a command by omitting certain options. For example, the following is a sophisticated
command:
nmcli connection modify id 'MyCafe' 802-11-wireless.mtu 1350
It can be simplified as follows:
nmcli con mod MyCafe 802-11-wireless.mtu 1350
# The id option can be omitted because the connection ID (name) is clear for nmcli in this case. If
you are familiar with those commands, you can further simplify them. Example:
nmcli connection add type Ethernet

# You can use the nmcli utility to start and stop any network interface, including the main interface.
Example:
nmcli con up id bond0
nmcli con up id port0
nmcli dev disconnect iface bond0
nmcli dev disconnect iface ens3
```

The functions of many nmcli commands are implied by their names, but the following options need to be further explained.

type: connection type.

- Allowed values: adsl, bond, bond-slave, bridge, bridge-slave, bluetooth, cdma, ethernet, gsm, infiniband, olpc-mesh, team, team-slave, vlan, wifi, wimax.
- Each connection type has type-specific command options. Press **Tab** to display the

option list, or view the **TYPE_SPECIFIC_OPTIONS** list on the nmcli(1) man page. The **type** option can be used in the following commands: **nmcli connection add** and **nmcli connection edit**.

- **con-name**: name assigned to a connection profile.
- If no name is specified, a name is generated in the following format:

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

- A connection name is the name of a connection profile and cannot be confused with the name of a device such as **wlan0**, **ens3**, and **em1**. Although users can name connections based on interfaces, they are different. A device can have multiple connection profiles. This is helpful for mobile devices, or for repeatedly switching network cables between different devices. Instead of editing the profile, create another profile and apply it to the interface as needed. The **id** option also refers to the name of a connection profile.

id: identification string assigned by the user to a connection profile.

- **id** can be used in the **nmcli connection** commands to identify a connection. The **NAME** field in the command output always denotes the connection ID (name). It refers to the same connection profile name that the **con-name** does.
- **uuid**: unique identification string assigned by the system to a connection profile.
- **uuid** can be used in the **nmcli connection** commands to identify a connection.

7.2.5 Connecting to a Network Using nmcli

Adding an Ethernet connection means creating a configuration profile which is then assigned to a device. Before creating a new profile, review the available devices as follows:

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

Adding a Dynamic Ethernet Connection

To add an Ethernet configuration profile with dynamic IP configuration so that DHCP can assign the network configuration, run the following command:

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

NetworkManager sets the internal parameter **connection.autoconnect** to **yes**. It also saves the settings to the **/etc/sysconfig/network-scripts/ifcfg-my-office** file, in which the **ONBOOT** instruction is set to **yes**.

Run the following command to activate the Ethernet connection:

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

7.2.6 Configuring Static Routes Using nmcli

To use the nmcli tool to configure static routes, you can use either the nmcli command line or the nmcli interactive editor.

To configure a static route for an existing Ethernet connection using the command line, run the following command:

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

This will direct traffic for the 192.168.122.0/24 subnet to the gateway at 10.10.10.1.

7.2.7 Network Interface Configuration File

In the openEuler system, each interface has a file that stores the configuration of the network interface. The file is in the **/etc/sysconfig/network-scripts/** directory and is named in the **ifcfg-***interfacename* format. You can modify the network interface configuration by modifying the file. However, you need to reload the network configuration or restart the network interface for the modification to take effect.

The following uses the **enp4s0** network interface as an example to describe how to configure a static network by modifying the **ifcfg** file as the **root** user. Modify the parameter settings in the **ifcfg-enp4s0** file generated in the **/etc/sysconfig/network-scripts/** directory. The following is an example:

```
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
```

To configure a dynamic network for the **em1** interface through the **ifcfg** file, generate an **ifcfg-em1** file in the **/etc/sysconfig/network-scripts/** directory. The following is an example:

```
DEVICE=em1
BOOTPROTO=dhcp
ONBOOT=yes
```

To configure an interface to send different host names to the DHCP server, add the following content to the **ifcfg** file:

```
DHCP_HOSTNAME=hostname
```

To ignore the impact of the DHCP server on the **/etc/resolv.conf** file, add the following content to the **ifcfg** file:

```
PEERDNS=no
```

To configure an interface to use specific DNS servers, set **PEERDNS** to **no** and add the following content to the **ifcfg** file:

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

ip-address specifies the IP address of the DNS server. In this way, the network service uses the specified DNS servers to update the **/etc/resolv.conf** file.

7.2.8 Host Name Management

Each Linux system has a host name. There are three classes of host names: static, transient, and pretty.

- static: static host name, which can be set by the user and saved in the **/etc/hostname** file.
- transient: dynamic host name maintained by the kernel. It is initialized to the static host name by default, whose value defaults to **localhost**. It can be changed by **DHCP** or **mDNS** at runtime.
- pretty: free-form host name (including special characters and whitespace characters). Static and dynamic host names comply with the general restrictions of domain names.

7.2.8.1 Configuring Host Names Using hostnamectl

To view all the current host names, run the following command:

```
hostnamectl status
```

To set all the host names on a system, run the following command as the **root** user:

```
hostnamectl set-hostname name
```

To set a particular host name, run the following command as the **root** user:

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

- Where *option* is one or more of: **--pretty**, **--static**, and **--transient**.
- If **--static** or **--transient** is used together with **--pretty**, the static and transient host names will be simplified forms of the pretty host name. Spaces will be replaced with hyphens (-), and special characters will be removed.
- When setting a pretty host name, remember to use the appropriate quotation marks if the host name contains spaces or a single quotation mark. For example:

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

7.2.8.2 Configuring Host Names Using nmcli

To query the static host name, run the following command:

```
nmcli general hostname
```

To set the static host name to **host-server**, run the following command as the **root** user:

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

To enable hostnamectl to detect the change of the static host name, run the following command the **root** user to restart the hostnamed service:

```
systemctl restart systemd-hostnamed
```

7.3 Process Management

7.3.1 Process Overview

Generally speaking, a program is a static file that contains executable code. A process is an instance of a program in execution.

After a program is called by the system to the memory, the system allocates certain resources (such as memory and devices) to the program and performs a series of complex operations to convert the program into a process for the system to call.

7.3.1.1 Process Classification

Processes are classified into the following types based on their functions and running programs:

System Processes

System processes can perform management operations such as memory resource allocation and process switchover. In addition, the running of system processes is not affected by user intervention. Even the root user cannot interfere with the running of system processes.

User Processes

A user process is a process generated by executing a user program, an application program, or a system program outside the kernel. This type of process can be run or be shut down by the user.

User processes are classified into the following types:

- Interactive process: An interactive process is a process created by a shell terminal. During the execution, the process needs to interact with the user. The process can run in the foreground or background.
- Batch processing process: This is a set of processes and is responsible for starting other processes in sequence.
- Daemon process: A daemon process is a process that keeps running. It is often started when the Linux system is started and terminated when the system is shut

down. For example, the crond processes.

7.3.1.2 Process States

To make full use of resources, the system differentiates processes in different states. Generally, the OS classifies processes into five states:

- New: The process is being created.
- Running: The process is running.
- Blocked: The process is waiting for an event to occur.
- Ready: The process is awaiting execution on a CPU.
- Terminated: The process has finished execution and the system is reclaiming resources.

On Linux, a process has five states, which correspond to the states of common OSs:

- Running: The process is running or waiting in the running queue.
- Interruptible: The process is sleeping, blocked, waiting for a condition to form or receive a signal.
- Uninterruptible: The process does not wake up or run after receiving any signals until an interrupt occurs.
- Zombie: The process is terminated, but the process descriptor exists. It will not be released until the parent process calls wait4().
- Stopped: The process stops running after receiving the SIGSTOP, SIGSTP, SIGTIN, or SIGTOU signal.

7.3.1.3 Process ID and Parent and Child Processes

A program may have many processes, and each process may have many child processes. That is, a process can have many descendants.

To distinguish different processes, the system allocates an ID to each process. In Linux, a process ID (PID) uniquely identifies a process. A PPID identifies a parent process. All processes are descendants of the init process whose PID is 1. The kernel starts the init process in the last phase of system boot.

Generally, each process has a parent process. The parent process manages the child process. When the parent process stops, the child process disappears. However, when the child process stops, the parent process may not be terminated.

7.3.1.4 Zombie Processes

A process is in the zombie state after it ends and waits for the parent process to release its resources.

An orphan process is a process whose parent process has finished or terminated, though it remains running itself. If there is no corresponding processing mechanism, the orphan process will be in the zombie state and resources cannot be released. Such orphan process is a zombie process.

Find a process in the startup process as the parent process of the orphan process, or directly use the init process as the parent process of the orphan process, to release the resources occupied by the orphan process.

7.3.1.5 Threads

A thread is a lightweight process in Linux. A process has an independent memory address space, but a thread does not. Threads cannot exist independently. They are created by processes.

7.3.2 Process Monitoring

7.3.2.1 Monitoring System Processes Using ps

Short for process state, **ps** gives a snapshot of the current processes. The **ps** command accepts several kinds of options:

BSD options, which must not be used with a dash. For example, **ps aux**.

UNIX/Linux options, which must be preceded by a dash. For example, **ps -ef**.

Options of different types may be freely mixed. For example, **ps ax -f**. This section mainly discusses the UNIX-style syntax.

- Note that **ps aux** is distinct from **ps -aux**. For example, **-u** displays user processes, but **u** displays detailed information.

7.3.2.2 Common Options and Outputs of ps

- **-a**: selects all programs associated with a terminal.
- **-A**: selects all processes.
- **-u**: selects the processes of a valid user.
- **-N**: negate selection.
- **-f**: does full listing. It displays the execution paths of programs in detail.
- **-e**: selects all processes. Identical to **-A**.
- **-l**: long format.
- **-F**: extra full format.
- **-H**: shows the process hierarchy (forest).
- **-L**: shows threads, possibly with **LWP** and **NLWP** columns.
- **-m**: shows threads after processes.
- **-h**: no header.
- **-w**: wide output.
- **--lines***<Number of lines>*: number of lines displayed on each page.
- **--width***<Number of characters>*: number of characters displayed on each page.
- **--help**: displays the help information.
- **--version**: displays the version.
- **a**: selects all processes on a terminal, including those of other users.
- **x**: selects processes without controlling terminals.
- **r**: restricts the selection to only running processes on the current terminal.
- **c**: shows the true name of a process.
- **e**: shows the environment variable.

- **f**: shows the ASCII-art process hierarchy (forest).
- **T**: selects all processes associated with this terminal.
- **u**: shows all processes of a specified user.

Columns in the Output

- **F**: flags associated with the process.
- **S**: process state. Identical to **STAT**.
- **SID**: session ID.
- **USER**: user account to which the process belongs.
- **PID**: process ID.
- **PPID**: parent process ID.
- **%CPU**: CPU utilization of the process
- **%MEM**: percentage of the physical memory used by the process.
- **VSZ**: virtual memory size of the process (in KiB).
- **RSS**: non-swapped physical memory that a task has used (in kilobytes).
- **TTY**: terminal (tty) associated with the process. It specifies the terminal on which the process runs. If the process is irrelevant to the terminal, an question mark (?) is displayed. In addition, tty1 to tty6 are login programs on the local host. If the value is **pts/0**, it indicates that the process is connected to the host through the network.
- **STAT**: state of the program. The states are as follows:
- **D**: uninterruptible sleep.
- **R**: running.
- **S**: interruptible sleep.
- **T**: stopped
- **Z**: defunct (zombie)
- **START**: time when the process is triggered to start.
- **TIME**: cumulative CPU time of the process.
- **COMMAND**: command for starting the process.
- **LWP**: light weight process (thread) ID.
- **C**: processor utilization.
- **NLWP**: number of lwps (threads) in the process. (alias **thcount**).
- **PRI**: priority of the process.
- **NI**: nice value.
- **ADDR**: process address space (not displayed).
- **SZ**: total memory (code + data + stack) of the process, in KB.
- **PSR**: processor that the process is currently assigned to.
- **STIME**: start time.
- **CMD**: command line for starting a task (including parameters).
- **FLAGS**: numeric identifier related to the process.

7.3.2.3 Monitoring System Processes Using pstree

This command displays a tree of running processes.

Syntax: **pstree** *[OPTIONS]*

- Description: The **pstree** command automatically combines programs with the same name. For example, **|-httpd---8*[httpd]** indicates that the httpd process generates eight child processes in the system.

Options:

- **-a**: displays command line arguments. If the command line of a process is swapped out, that process is shown in parentheses. **-a** implicitly disables compaction. The same process is displayed in the form of *n*(process)*.
- **-c**: disables compaction of identical subtrees. Processes of the same name will be displayed one by one. By default, subtrees are compacted whenever possible.
- **-G**: uses VT100 line drawing characters.
- **-h**: highlights the current process and its ancestors. This is a no-op if neither the current process nor any of its ancestors are in the subtree being shown.
- **-H**: highlights the specified process and its ancestors. The syntax is **pstree -H** *PID*.
- **-l**: displays long lines. By default, lines are truncated to the display 132 bits. If the width exceeds 132 bits, the lines cannot be displayed normally.
- **-n**: sorts processes with the same ancestor by PID instead of by name.
- **-p**: displays all processes. The output contains the process name and PID.
- **-u**: displays the UID of the user. Whenever the UID of a process differs from the UID of its parent, the new UID is shown in parentheses after the process name.
- **-U**: uses UTF-8 (Unicode) line drawing characters.
- **-v**: displays version information.

7.3.2.4 Monitoring System Processes Using top

The **top** provides a dynamic real-time view of Linux processes, which is similar to the Windows task manager. The **top** command is a powerful tool for monitoring the system. It is especially important for the system administrator. Its disadvantage is that it consumes certain system resources.

Summary Display

Line 1: uptime

- Current time
- System uptime
- Number of current login users
- Load averages for the system. The load averages are the average number of process ready to run during the last 1, 5, and 15 minutes.

Line 2: processes

- Total number of processes
- Number of running processes
- Number of sleeping processes

- Number of stopped processes
- Number of zombie processes

Line 3: CPU states

- Percentage of CPU time in user mode
- Percentage of CPU time in system mode
- Percentage of CPU time for niced tasks
- Percentage of idle CPU time
- Percentage of CPU time for iowait
- Percentage of CPU time for hardware IRQs
- Percentage of CPU time for software interrupts

Line 4: Mem

- Total physical memory
- Total used physical memory
- Total free memory
- Memory used for buffers

Line 5: Swap

- Total swap space
- Used swap space
- Available swap space
- Swap space used for cache

Line 6: blank line

Line 7 and other: state of each process

- **PID**: process ID.
- **PPID**: parent process ID.
- **RUSER**: real user name.
- **UID**: user ID of the task's owner.
- **USER**: user name of the task's owner.
- **Group**: group name of the task's owner.
- **TTY**: name of the controlling terminal. However, a task does not need to be associated with a terminal, in which case you will see a question mark (?) displayed.
- **PR**: priority of the task.
- **NI**: nice value of the task. A negative value indicates a high priority, and a positive value indicates a low priority.
- **P**: last used CPU, which is valid only in a multi-CPU environment.
- **%CPU**: the task's share of the CPU time since the last screen update.
- **TIME**: total CPU time the task has used since it started, in seconds.
- **TIME+**: total CPU time the task has used since it started, in centiseconds
- **%MEM**: the task's share of the physical memory.

- **VIRT**: total amount of virtual memory used by the task, in KiB. VIRT = SWAP + RES
- **SWAP**: swapped size of the virtual memory, in KiB.
- **RES**: non-swapped size of the physical memory, in KiB. RES = CODE + DATA
- **CODE**: size of the physical memory occupied by the executable code, in KiB.
- **DATA**: physical memory occupied by non-executable code (data segments + stacks), in KiB.
- **SHR**: shared memory size, in KiB.
- **nFLT**: page errors.
- **nDRT**: number of pages that have been modified since they were last written to auxiliary storage.
- **S**: process state.
- **D**: uninterruptible sleep.
- **R**: running.
- **S**: sleeping.
- **T**: traced/stopped.
- **Z**: zombie.
- **COMMAND**: command name or command line.
- **WCHAN**: shows the name of the kernel function in which the task is currently sleeping.
- **Flags**: task flags. For details, see **sched.h**.

7.3.2.5 Killing a Process

kill Command

Syntax:

- **kill** [OPTION] [PID]

Options:

- **-l**: prints a list of signal names. If no signal number is added, **-l** lists all signal names.
- **-a**: does not restrict the command-name-to-PID conversion to processes with the same UID as the present process.
- **-p**: only prints PID of the named processes, but does not send any signals.
- **-s**: specifies the signal to send.
- **-u**: specifies a user.

Only the **9** (**SIGKILL**) signal can terminate a process unconditionally. Other signals can be ignored by processes. The following are common signals:

- HUP 1 Hangup
- INT 2 Terminal interrupt (same as **Ctrl+C**)
- QUIT 3 Terminal quit (same as **Ctrl+**)
- TERM 15 Termination
- KILL 9 Kill (cannot be caught or ignored)

- CONT 18 Continue executing, if stopped (contrary to **STOP** and **fg**/**bg** commands)
- STOP 19 Stop executing (same as **Ctrl+Z**)

Key processes, for example, the bash process, cannot be killed.

The init process is not allowed to terminate. As one of the indispensable programs in Linux system operations, the init process is a user-level process started by the kernel. After the kernel automatically starts (the kernel has been loaded into the memory, started to run, and initialized all device drivers and data structures), a user-level program init is started to complete the boot process. Therefore, init is always the first process (whose ID is always 1). All the other processes are descendants of the init process. The init process cannot be killed.

killall Command

killall kills processes (and their child processes) by name. To kill the SVN server process for example, run the following command:

```
killall svnserver
```

To forcibly kill a process, use **-9**. For example, **killall -9 cpusd**.

7.3.2.6 Process Priority

PRI (Priority) Value and NI (Nice) Value

A smaller **PRI** value indicates a higher process running priority. The **PRI** value is adjusted by the Linux kernel. If you want to adjust the priority of a process, you can set nice value (**NI**) of the process. Generally, the relationship between **PRI** and **NI** is as follows:

- PRI(new) = PRI(old) + NI
- **NI** ranges from **–20** to **19**, comprising 40 levels.
- If the value of **NI** is a negative number, the value of **PRI** decreases and the priority of the process increases.

7.4 Quiz

- Change the host name to openEuler.
- Add a NIC to the VM. Set the IP address of the NIC to 192.168.101.100/24 and the gateway to 192.168.101.254. Set the default route of the 192.168.0.0/16 network segment to pass through the NIC.
- Create a scheduled task to export the current time at 17:30.
- Create a periodic task to back up the **/etc** directory to the **/backup** directory at 02:00 every day. The name of the backup directory must be changed to *Current date*-**etc**, for example, **20201202-etc**.
- Create a periodic task to check the root partition usage every hour. If the root partition usage exceeds 50%, an alarm is generated and all data in the **/tmp** directory is deleted.

8 Shell Scripts

8.1 Shell Basics

8.1.1 Shell Overview

The shell is a program written in C language. It is a bridge for users to interact with Unix/Linux. It completes most of users' work. The shell is both a command language and a programming language. As a command language, it interactively interprets and executes commands entered by users. As a programming language, it defines various variables and arguments and provides many control structures that are available only in high-level languages, including loops and branches.

Although it is not a part of the Unix/Linux system kernel, it calls most functions of the system kernel to execute programs, create files, and coordinate the running of programs in parallel. Therefore, the shell is the most important utility for users. Understanding and mastering shell features and usage methods are the key to using the Unix/Linux system.

The shell accepts commands in either of the following ways:

- Interactive shell: interprets and executes user commands one by one.
- Batch shell: a shell script written in advance, composing multiple commands that will be executed in a batch.

Shell scripts, like most programming languages, also have variable assignment and control statements. However, they are interpreted and executed but not compiled. The shell program reads and executes commands line by line in the script, which is similar to a user entering commands line by line to the shell prompt for execution.

For shell beginners, it is recommended that you do not run the shell as the **root** user. As a common user, you cannot damage the system intentionally or unintentionally. As the **root** user, however, entering a few letters can cause catastrophic consequences.

8.1.2 Common Shells

Common shell script interpreters in Unix/Linux include bash, sh, csh, and ksh, which are called shells. When we talk about how many shells we have, actually we are talking about the shell script interpreters. The default login shell of openEuler is bash.

Bash

Bash is the default shell of Linux. This tutorial is also based on bash. Bash, short for Bourne Again Shell, is written by Brian Fox and Chet Ramey. It has 40 internal commands.

Linux uses it as the default shell because it has the following features:

- It can use the doskey function similar to DOS to view, quickly enter, and modify

commands by using arrow keys.

- It automatically searches for and matches the commands starting with a specific character string.
- It provides help information. You can enter **help** under the prompt to obtain related help.

sh

Developed by Steve Bourne, the Bourne shell (sh) is the default shell for Unix.

ash

The Almquist shell (ash) is written by Kenneth Almquist. It is a small shell that occupies the least system resources in Linux. It contains only 24 internal commands, which is inconvenient to use.

csh

The C shell (csh) is written by 47 authors represented by William Joy and has 52 internal commands. The shell points to **/bin/tcsh**, so csh is tcsh.

ksh

The KornShell (ksh) is written by David Korn and has 42 internal commands. The biggest advantage of the shell is that it is almost fully compatible with the ksh of the commercial edition. You can enjoy the performance of the commercial edition without paying for it.

- Note: Bash is the default shell of Linux. It is based on sh and absorbs some features of csh and ksh. Bash is fully compatible with sh. The scripts written in sh can be executed in bash without modification.

8.1.3 Differences Between Shell and Compiled Languages

In general, programming languages can be divided into two categories: compiled languages and interpreted languages.

Compiled Languages

Many traditional programming languages, such as Fortran, Ada, Pascal, C, C++, and Java, are compiled languages. For such languages, the source code needs to be converted into the object code in advance. This process is called compilation.

The object code is read to run a program. Because the compiled object code is close to the bottom layer of the computer, the execution efficiency is high, which is an advantage of compiled languages.

However, because compiled languages mostly operate at the bottom layer and process bytes, integers, floating point numbers, or other machine-level objects, a large amount of complex code is required to implement a simple function. For example, in C++, it is difficult to perform simple operations such as copying all files in one directory to another.

Interpreted Languages

Interpreted languages are also called scripting languages. When executing such programs, an interpreter needs to read the source code and convert it into the object code for the computer to run. The efficiency decreases because the compilation process is required each time the program is executed.

The advantage of using scripting languages is that they run at a higher level than compiled

languages and can easily process objects such as files and directories. The disadvantage is that they are usually less efficient than compiled languages. However, it is worthwhile to use scripting languages. It takes an hour to write a simple script. It may take two days to compile and implement the same function in C or C++. In addition, the script execution speed is high enough to ignore its performance problem. Common scripting languages are AWK, Perl, Python, Ruby, and Shell.

8.1.4 When to Use Shell

The shell is a common function in Unix systems and has been standardized by POSIX. Therefore, shell scripts written with a good style can be reused in other systems. The advantages of shell scripts are as follows:

- Simplicity: The shell is a high-level language that can express complex operations concisely.
- Portability: With the functions defined by POSIX, scripts can be executed on different systems without modification.
- Easy development: A powerful and easy-to-use script can be completed in a short time.

However, considering the command restrictions and efficiency, shell scripts are not used in the following cases:

- Resource-intensive tasks, especially when efficiency is an important factor (for example, sorting and hash).
- Large-scale mathematical operations, especially floating-point operations, accurate operations, and complex arithmetic operations (C++ or Fortran is recommended).
- Cross-platform (operating system) porting (C or Java is recommended).
- Complex applications that require structured programming (with variable type checking, function prototype, and so on).
- Mission-critical applications that affect the overall system performance.
- Tasks that have high security requirements. For example, you need a robust system to prevent intrusion, cracking, and malicious damage.
- A project consisting of a series of dependent parts.
- Large-scale file operations.
- Multi-dimensional arrays.
- Data structures, such as linked lists and numbers.
- A graphical user interface (GUI) to be generated or operated.
- Operations on system hardware.
- I/O or socket interfaces.
- Interfaces that need to use libraries or legacy code.
- Private, closed-source applications (shell scripts put code in text files that can be seen worldwide).

If your application complies with any of the above, consider a more powerful language - perhaps Perl, Tcl, Python, Ruby - or a higher-level compiled language such as C, C++, or Java. Even so, you will find that using shell to prototype your application is very useful in

development.

8.2 Shell Programming Basics

8.2.1 Shell Script Specifications and Running

Generally, a shell script has a file name extension **.sh**, for example, **test.sh**. A shell script starts with **#!** in the first line. The shebang line tells the system which interpreter should be used to execute the script. For example, **#!/bin/bash** means Bash is used to interpret the script.

To run a script, the execute permission on the script is required. You can run the **chmod** command to grant the execute permission. You can enter an absolute path, for example, **/root/test.sh**, or go to the **/root** directory and enter **./test.sh** to execute the script on which you have the execute permission.

To comment out a single line in a shell script, you can add a hashtag symbol with no white spaces (#) at the beginning of the line.

8.2.2 Shell Variables

Shell supports user-defined variables.

Defining Variables

When defining a variable, do not add the dollar sign ($) to the variable name. Example:

```
variableName="value"
```

Note that there is no space between the variable name and the equal sign, which may not be the same as any programming language you are familiar with. In addition, variable names must comply with the following rules:

- The first character must be a letter (a-z or A-Z).
- No space is allowed. You can use underscores (_) instead.
- Punctuation is not allowed.
- Keywords in Bash cannot be used. You can run the **help** command to view the reserved keywords.

Using Variables

To use a defined variable, you only need to add the dollar sign ($) before the variable name. Example:

```
your_name="mozhiyan"
echo $your_name
echo ${your_name}
```

The braces outside the variable name are optional. They are used to help the interpreter identify the boundary of the variable. For example:

```
for skill in Ada Coffe Action Java
```

```
do
    echo "I am good at ${skill}Script"
done
```

If you do not add braces to the **skill** variable and write it as **echo "I am good at $skillScript"**, the interpreter regards **$skillScript** as a variable (the value is empty). As a result, the code execution result is not as expected.

It is recommended that all variables be enclosed in braces, which is a good style in programming.

Redefining Variables

Defined variables can be redefined. Example:

```
myUrl="http://see.xidian.edu.cn/cpp/linux/"
echo ${myUrl}
myUrl="http://see.xidian.edu.cn/cpp/shell/"
echo ${myUrl}
```

Note that **$myUrl="http://see.xidian.edu.cn/cpp/shell/"** is invalid for redefining a variable. Add the dollar sign ($) only when you intend to use a variable.

Read-only Variables

You can run the **readonly** command to define a variable as a read-only variable. The value of a read-only variable cannot be changed. In the following example, the value of a read-only variable is changed.

```
#!/bin/bash

myUrl="http://see.xidian.edu.cn/cpp/shell/"
readonly myUrl
myUrl="http://see.xidian.edu.cn/cpp/danpianji/"
```

When the script is run, an error is reported.

```
/bin/sh: NAME: This variable is read only.
```

Deleting Variables

You can run the **unset** command to delete a variable. Syntax:

```
unset variable_name
```

A deleted variable cannot be used again. The **unset** command cannot delete a read-only variable.

Example:

```
#!/bin/sh
myUrl="http://see.xidian.edu.cn/cpp/u/xitong/"
unset myUrl
echo $myUrl
```

The preceding script has no output.

Variable Type

When the shell is run, three types of variables exist at the same time:

- Local variables: Local variables are defined in scripts or commands and are valid only in the current shell instance. Programs started by other shells cannot access local variables.
- Environment variables: All programs, including those started by shells, can access environment variables. Some programs require environment variables to ensure their normal running. If necessary, the shell script can also define environment variables.
- Shell variables: A shell variable is a special variable set by a shell program. Some shell variables are environment variables, and some are local variables. These variables ensure the normal running of the shell.

8.2.3 Special Shell Variables and Location Parameters

Special Variables

Some special variables are involved during script running, as described in Table 8-1.

Table 8-1 Special variables

Variable	Description
$0	Name of the current script.
$n	Parameter passed to a script or function. n is a number, indicating the sequence number of a parameter. For example, the first parameter is $1, and the second parameter is $2.
$#	Number of parameters passed to a script or function.
$*	All parameters passed to a script or function.
$@	All parameters passed to a script or function. When enclosed in double quotation marks, it is slightly different from $*, which will be described later.
$?	Exit status of the previous command or the return value of the function.
$$	ID of the current shell process. For shell scripts, the value is the ID of the process where the scripts are located.

Location Parameters

The parameters passed to the script when the script is run are called command line parameters. Command line parameters are represented by **$*n***. For example, **$1** indicates the first parameter, **$2** indicates the second parameter, and so on. Example:

```
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 : $#"
Output:

$./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 Replacement

Character Replacement

If an expression contains special characters, the shell will replace them. For example, enclosing variables in double quotation marks is a replacement, and using escape characters is also a replacement.

Example:

```
#!/bin/bash
a=10
echo -e "Value of a is $a \n"
```

Output:

```
Value of a is 10
```

-e replaces escape characters. If **-e** is not used, the output is as follows:

```
Value of a is 10\n
```

The following escape characters can be used in **echo**.

Table 8-2 Escape characters

Escape Character	Description
\	Escape
\a	Alert
\b	Backspace
\f	Form feed (new page)
\n	New line
\r	Carriage return

\t	Horizontal tab
\v	Vertical tab

Command Replacement

Command replacement means that the shell can run the command first, save the output temporarily, and output the output in a proper place.

Syntax: `command`

The command is enclosed in back quotes (the key under the **Esc** key) instead of single quotation marks.

The following example shows how to save the command output in a variable.

```
#!/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"
```

Output:

```
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
```

Variable Replacement

Variable replacement refers to change the value of a variable based on the variable status (whether the variable is empty or defined).

The following table describes variable replacement syntax.

Table 8-3 Variable replacement syntax

Syntax	Description
${var}	Original value of a variable.
${var:-word}	If the variable *var* is empty or has been deleted (**unset**), *word* is returned, but the value of *var* is not changed.
${var:=word}	If the variable *var* is empty or has been deleted (**unset**), *word* is returned and the value of *var* is changed to *word*.
${var:?message}	If the variable *var* is empty or has been deleted (**unset**), a *message* is sent to the standard error output to check whether the variable *var* can be assigned a value normally. If the replacement appears in a shell script, the script stops

	running.
${var:+word}	If the variable *var* is defined, *word* is returned, but the value of *var* is not changed.

Example:

```
#!/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}"
```

Output:

```
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 Operators

Bash supports many operators, including arithmetic operators, relational operators, Boolean operators, string operators, and file test operators. The native Bash does not support simple mathematical operations, which can be implemented using other commands, such as **awk** and **expr** (commonly used).

Arithmetic Operators

Example:

```
#!/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
```

Output:

```
a + b : 30
a - b : -10
a * b : 200
b / a : 2
b % a : 0
a is not equal to b
```

Table 8-4 Arithmetic operators

Operator	Description	Example
+	Addition	The result of **`expr $a + $b`** is 30.
-	Subtraction	The result of **`expr $a - $b`** is 10.
*	Multiplication	The result of **`expr $a \* $b`** is 200.
/	Division	The result of **`expr $b / $a`** is 2.
%	Remainder	The result of **`expr $b % $a`** is 0.
=	Value assignment	**a=$b** means to assign the value of variable **b** to variable **a**.
==	Equal to. It compares two numbers. If they are the same, true is returned.	[$a == $b] returns false.

Operator	Description	Example
!=	Not equal to. It compares two numbers. If they are different, true is returned.	[$a != $b] returns true.

Note: The condition expression must be enclosed by square brackets and must contain spaces. For example, **[$a==$b]** is incorrect and must be written as **[$a == $b]**.

Relational Operators

Relational operators support only digits and do not support character strings unless the value of a character string is a number.

Table 8-5 Relational operators

Operator	Description
-eq	Checks whether two numbers are equal. If yes, **true** is returned.
-ne	Checks whether two numbers are equal. If they are not equal, **true** is returned.
-gt	Checks whether the number on the left is greater than that on the right. If yes, **true** is returned.
-lt	Checks whether the number on the left is less than that on the right. If yes, **true** is returned.
-ge	Checks whether the number on the left is greater than or equal to that on the right. If yes, **true** is returned.
-le	Checks whether the number on the left is less than or equal to that on the right. If yes, **true** is returned.

Example:

```
#!/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
```

Output:

```
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
```

Boolean Operators

Example:

```
#!/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
```

Output:

```
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
```

Table 8-6 Boolean operators

Operator	Description	Example
!	NOT operation. If the expression is true, **false** is returned. Otherwise, **true** is returned.	[! false] returns true.
-o	OR operation. If one expression is true, **true** is returned.	[$a -lt 20 -o $b -gt 100] returns true.
-a	AND operation. If both expressions are true, **true** is returned.	[$a -lt 20 -a $b -gt 100] returns false.

String Operators

Example:

```
#!/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
```

Output:

```
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
```

Table 8-7 String operators

Operator	Description	Example
=	Checks whether two strings are equal. If they are equal, **true** is returned.	[$a = $b] returns false.
!=	Checks whether two strings are equal. If they are not equal, **true** is returned.	**[$a != $b]** returns **true**.
-z	Checks whether the length of a character string is 0. If the length is 0, **true** is returned.	**[-z $a]** returns **false**.
-n	Checks whether the length of a character string is 0. If the length is not 0, **true** is returned.	**[-z $a]** returns **true.**

Operator	Description	Example
str	Checks whether a character string is empty. If no, **true** is returned.	**[$a]** returns **true**.

File Test Operators

File test operators are used to detect various attributes of Unix files.

For example, the **/var/www/tutorialspoint/unix/test.sh** file, whose size is 100 bytes, has the rwx permission. The following code checks the attributes of the file:

```
#!/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 special 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
```

Output:

```
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
```

Table 8-8 File test operators

Operator	Description	Example
-b file	Checks whether a file is a block device file. If yes, **true** is returned.	**[-b $file]** returns **false**.
-c file	Checks whether a file is a character device file. If yes, **true** is returned.	**[-b $file]** returns **false**.
-d file	Checks whether a file is a directory. If yes, **true** is returned.	**[-d $file]** returns **false**.
-f file	Checks whether a file is a common file (neither a directory nor a device file). If yes, **true** is returned.	**[-f $file]** returns **true**.
-g file	Checks whether the SGID bit is set for a file. If yes, **true** is returned.	**[-g $file]** returns **false**.
-k file	Checks whether a sticky bit is set for a file. If yes, **true** is returned.	**[-k $file]** returns **false**.
-p file	Checks whether a file is a named pipe. If yes, **true** is returned.	**[-p $file]** returns **false**.
-u file	Checks whether the SUID bit is set for a file. If yes, **true** is returned.	**[-u $file]** returns **false**.

Operator	Description	Example
-r file	Checks whether a file is readable. If yes, **true** is returned.	**[-r $file]** returns **true**.
-w file	Checks whether a file is writable. If yes, **true** is returned.	**[-w $file]** returns **true**.
-x file	Checks whether a file is executable. If yes, **true** is returned.	**[-x $file]** returns **true**.
-s file	Checks whether a file is empty (whether the file size is greater than 0). If no, **true** is returned.	**[-s $file]** returns **true**.
-e file	Checks whether a file (including a directory) exists. If yes, **true** is returned.	**[-e $file]** returns **true**.

8.2.6 Shell Character Strings

Strings are the most commonly used data type in shell scripting. Strings can be enclosed in single quotation marks or double quotation marks, or does not need to be enclosed in quotation marks.

Single Quotation Marks

```
str='this is a string'
```

Restrictions: Any characters enclosed in single quotation marks are output without any changes. Variables in the string enclosed in single quotation marks are invalid. Strings enclosed in single quotation marks cannot contain extra single quotation marks (even if escape characters are used for the single quotation marks).

Double Quotation Marks

```
your_name='openEuler'
str="Hello, I know your are \"$your_name\"! \n"
```

Advantages: Strings enclosed in double quotation marks can contain variables and support escape characters.

Concatenate strings.

```
your_name="openEuler"
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"

echo $greeting $greeting_1
```

Obtain the length of a character string.

```
string="abcd"
echo ${#string} #Outputs 4.
```

Extract substrings.

```
string="huawei is a great company"
echo ${string:1:4}
```

Search for a substring.

```
string="huawei is a great company"
echo `expr index "$string" is`
```

8.2.7 If-Else Statement

The if statement uses relational operators to determine whether an expression is true or false to determine which branch to execute. There are three types of if-else statements in the shell:

- if...fi
- if...else...fi
- if...elif...else...fi

if...fi

Syntax

```
if [ expression ]
then
    Statement(s) to be executed if expression is true
fi
```

if...else...fi

Syntax

```
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...else...fi

Syntax

```
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 Statement

Similar to the switch...case statement in other languages, case...esac is a multi-branch selection structure.

The case statement matches a value or a pattern. If the matching is successful, the matched command is executed. Syntax:

```
case value in
Pattern 1
    command1
    command2
    command3
    ;;
Pattern 2
    command1
    command2
    command3
    ;;
*)
    command1
    command2
    command3
    ;;
esac
```

The patterns of the case statement are shown in the above code. The *value* must be followed by the keyword **in**, and each pattern must end with a right parenthesis. The *value* can be a variable or a constant. If the *value* complies with a pattern, all commands in front of **;;** will be executed. Similar to the **break** in other languages, **;;** means to jump to the end of the entire case statement.

The *value* tries to match each pattern. Once a pattern is matched, other patterns will not be continued after the corresponding commands of the matched pattern are executed. If no pattern is matched, the asterisk (*) is used to capture the value and the commands followed will be executed.

8.2.9 For Loop

Similar to other programming languages, the shell supports the for loop.

Syntax:

```
for variable in list
do
    command1
    command2
    ...
```

```
    commandN
done
```

A list is a sequence of values (digits, strings, etc.) separated by space. Each time a loop occurs, the next value in the list is assigned to the variable. The in list is optional. If it is not used, the for loop uses the location parameter of the command line.

Output the numbers in the current list in ascending order.

```
for loop in 1 2 3 4 5
do
    echo "The value is: $loop"
done
```

Output:

```
The value is: 1
The value is: 2
The value is: 3
The value is: 4
The value is: 5
```

Output characters in a character string in sequence.

```
for str in 'This is a string'
do
    echo $str
done
```

Output:

```
This is a string
```

Display the files whose names start with **.bash** in the home directory.

```
#!/bin/bash

for FILE in $HOME/.bash*
do
   echo $FILE
done
```

Output:

```
/root/.bash_history
/root/.bash_logout
/root/.bash_profile
/root/.bashrc
```

8.2.10 While Loop

The while loop continuously executes a series of commands and reads data from the input

file. Commands are usually test conditions. Syntax:

```
while command
do
    Statement(s) to be executed if command is true
done
```

After the command is executed, the control returns to the beginning of the loop until the test condition is false.

8.2.11 Script Instances

if

- Number guessing game

```
#!/bin/bash
a=3
b=$1
if [ $a == $b ]
then
 echo "You win!"
else
 echo "Please guess again."
fi
```

while + if

- Check whether the location parameter is a file. If yes, the file content is displayed. If no, a message is displayed.

```
#!/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

- Read each line of information from the file and output the information.

```
#!/bin/bash
for Name in $(cat ./namefile)
do
        echo $Name
done
```

case

- Input data from the keyboard and selectively output data based on the input data.

```
#!/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 Quiz

- Create a **userlist** file and write 10 user names (a user name per line) in the file. Create a **useradd.sh** script that is able to automatically create users written in the **userlist** file, with the password **openEuler12#$**. The home directory of the new user contains a **hello.txt** file, and the owner and group of the file is the user.

9 Samba File Sharing Server Management

9.1 Samba Overview

In 1987, Microsoft and Intel jointly formulated the Server Messages Block (SMB) protocol, which aims to solve the problem of sharing resources such as files or printers on a local area network (LAN). This makes it easier to share files among multiple hosts. In 1991, Tridgwell, who was a college student, developed the open source SMBServer daemon based on the SMB protocol to solve the problem of file sharing between Linux and Windows. After simple configuration, files can be shared between Linux and Windows using SMBServer. At that time, Tridgwell wanted to register the software's name SMBServer as a trademark, but was rejected by the trademark office because the word SMB was "meaningless". Later, Tridgwell kept looking through the dictionary and suddenly saw the Samba dance, which happened to contain "SMB". This was how the Samba daemon was born. Samba has now become the optimal choice for sharing files between Linux and Windows systems.

The original purpose of Samba is to connect Windows with Unix-like systems. The capabilities of Samba include:

- Sharing files and printers
- Providing identity authentication when users log in to the Samba host to present different data to different users
- Host name resolution (NetBIOS name) on Windows networks
- Sharing devices (for example, Zip and CD-ROMs)

9.1.1 Basic Configuration of the Samba Server

9.1.1.1 Samba Introduction

The method of configuring the Samba daemon is similar to that of configuring many services. You need to install the Samba daemon through DNF. The name of the Samba daemon is also the name of the software package. Samba includes the following software packages:

- samba: contains daemons (smbd and nmbd), files, logrotate configuration files, and default startup option files required by the Samba server.
- samba-client: provides tool commands required when running Samba client on Linux, for example, mount.cifs for mounting the Samba file format and smbtree for obtaining the tree diagram similar to network neighborhood.

- samba-common: provides data used by both the server and the client, including the main configuration file **smb.conf** and the syntax check command **testparm**.

9.1.1.2 Samba Configuration Files

Samba configuration files include:

- **/etc/samba/smb.conf**: This is the main and the mandatory configuration file of Samba. This configuration file is also a detailed description file. You can run the **vim** command to view the content. The main settings include server-related settings (the **global** section), such as the workgroup, NetBIOS name, and password level, and shared directory settings, such as the actual directory, shared resource name, and permissions.
- **/etc/samba/lmhosts**: In earlier versions of Samba, the NetBIOS name needs to be set additionally. Therefore, the **lmhosts** file containing the IP addresses corresponding to the NetBIOS names is required. The file functions similar to **/etc/hosts**. The only difference is that in **lmhosts**, the host names are NetBIOS names. Samba can now uses your local host name (hostname) as your NetBIOS name by default, so **lmhosts** is not optional.
- **/etc/sysconfig/samba**: records service parameters that you want to add for running smbd and nmbd.
- **/etc/samba/smbusers**: The administrator and guest user names are different in Windows and Linux, for example, administrator (Windows) and root (Linux). You can use this file to map the user names.
- **/var/lib/samba/private/{passdb.tdb,secrets.tdb}**: database archive used for managing Samba user accounts and passwords.
- **/usr/share/doc/samba-<Version>**: This directory contains the documents of Samba. After Samba is installed, a rich and complete SAMBA user manual is generated in this directory.

9.1.2 Samba Server Configuration

9.1.2.1 smb.conf File Configuration

[global] Parameters

In the **[global]** section, you can set the global parameters of the server, including the workgroup, NetBIOS name of the host, character encoding, login file settings, whether to use passwords, and password authentication mechanisms. The **[Shared Resource Name]** section is used to set permissions for the directory that you want to expose, including who can browse the directory and whether the directory can be read and written. The parameters related to the host name in the **[global]** section are as follows:

workgroup = Name of the workgroup. Note that the workgroup across the host group must be the same.

netbios name = NetBIOS name of the host. Each host has a unique NetBIOS name.

server string = Brief description of the host. Enter any value.

In addition, there is information about log files, including the following parameters:

log file = The log file. The file name can contain variables.

max log size = Maximum size of the log file, in KB. When this value is exceeded, the earliest logs are

rotated.

There are also password parameters for security purposes, including:

security = The value can be **share**, **user**, or **domain**. The values indicate the following:

share: The shared data does not require a password and can be used by all users (unsafe).

user: The password database of the Samba server is used. The password database is related to the passdb backend.

domain: The password database of an external server is used. That is, Samba functions as a client. If this parameter is set, you need to configure the value of **password server** too.

[Shared Resource Name] Parameters

Common parameters in the **[Shared Resource Name]** section are as follows:

[Shared Resource Name]: This parameter is the name of the shared resource and is mandatory.

comment: description of the directory.

path: Linux file system (directory) to which **[Shared Resource Name]** points to.

browseable: whether to allow all users to view this resource.

writable: indicates whether data can be written.

writelist = User, @Group: specifies users who can access this resource. If the format is **@Group**, all users in the group have the permission.

The shared resources are mainly related to file permissions of Linux. Therefore, the parameters in the file are related to permissions.

smb.conf Variables

To simplify configuration, Samba provides the following variables:

- **%S**: indicates the current item value.
- **%m**: indicates the NetBIOS host name of the client.
- **%M**: indicates the Internet host name of the client.
- **%L**: indicates the NetBIOS host name of the Samba host.
- **%H**: indicates the home directory of the user.
- **%U**: indicates the name of the current login user.
- **%g**: indicates the group name of the login user.
- **%h**: indicates the host name of the current Samba host.
- **%I**: indicates the IP address of the client.
- **%T**: current date and time

9.2 Samba File Sharing Server Example

The following is an example of how to set up a Samba server:

- Install the Samba service and related components.

```
[root@openEuler ~]# dnf -y install samba samba-client samba-common
```

- Start the Samba service and enable it to start upon system startup.

```
[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.
```

- Check the listening status of the server. The server is listening on TCP ports 139 and 445.

```
[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
```

- Check whether the firewall is enabled. If yes, disable the firewall.

```
[root@openEuler ~]# systemctl stop firewalld
[root@openEuler ~]# setenforce 0     # Temporarily disable seLinux.
```

- Configure a share access user, for example, smb.

```
[root@openEuler ~]# useradd -s /sbin/nologin -M smb
```

- Set the Samba server password for the **smb** user, for example, **Huawei12#$**.

```
[root@openEuler02 samba]# smbpasswd -a smb
New SMB password:
Retype new SMB password:
Added user smb.
```

- Create a shared directory.

```
[root@openEuler ~]# mkdir /var/share /var/smb
[root@openEuler ~]# chmod 777 /var/share /var/smb
[root@openEuler ~]# chown smb:smb /var/smb
```

- Configure a shared directory.

```
[root@openEuler ~]# vim /etc/samba/smb.conf
Add the following information to the [global] section:
[global]
        workgroup = SAMBA
        security = user
        map to guest = Bad User     # Add this line.
        passdb backend = tdbsam

        printing = cups
        printcap name = cups
        load printers = yes
        cups options = raw
```

- Add the **share** directory for public sharing and allow anonymous access to the directory.

```
[share]
        comment = share
        path = /var/share
        guest ok = yes
        writeable = yes
        browseable = yes
```

- Add the **smb** directory and allow the share access user to access the directory.

```
[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
```

- Save the configuration and exit. Then, restart the Samba service.

```
[root@openEuler ~]# systemctl restart smb
```

- Now, you can use Windows or other Linux systems with the CIFS access client installed to access the shared resources.

9.3 Quiz

- How do you configure Samba to ensure privacy when a user accesses shared resources?

Tip: You can configure the home directory.

10 Reference

https://www.openeuler.org/en/
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