EXAM CRAM

CompTIA®

Linux+

XK0-005


Cram Sheet


Practice Tests


Exam Alerts

WILLIAM "BO" ROTHWELL

EXAM CRAM

CompTIA® Linux+® XK0-005 Exam Cram

William "Bo" Rothwell

CompTIA® Linux+® XK0-005 Exam Cram

ISBN-13: 978-0-13-789855-8

ISBN-10: 0-13-789855-X

Library of Congress Control Number: 2022910969

ScoutAutomatedPrintCode

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson IT Certification cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

This book is designed to provide information about the CompTIA® Linux+® (XK0-005) certification. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact

governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact

intlcs@pearson.com.

Editor-in-Chief
Mark L. Taub

Director, ITP Product Management
Brett Bartow

Executive Editor
Nancy Davis

Development Editor
Christopher A. Cleveland

Managing Editor
Sandra Schroeder

Project Editor
Mandie Frank

Copy Editor
Kitty Wilson

Indexer
Erika Millen

Proofreader
Donna E. Mulder

Technical Editor
Casey Boyles

Publishing Coordinator
Cindy Teeters

Designer
Chuti Prasertsith

Compositor
codeMantra

This page intentionally left blank

Contents at a Glance

Part IV: Troubleshooting

Table of Contents

Part III: Scripting, Containers, and Automation

Figure Credits

Figure	Credit
Figures 1.3, 1.4, 2.2	GNU Project
Figures 2.3, 4.2, 5.2, 6.2, 14.1–14.8, 19.1, 19.2, 20.1, 20.2	Linux Kernel Organization, Inc
Figure 5.1	Wireshark
Figure 11.1	Mozilla.org

About the Author

At the impressionable age of 14, **William "Bo" Rothwell** crossed paths with a TRS-80 Micro Computer System (affectionately known as a "Trash 80"). Soon after the adults responsible for Bo made the mistake of leaving him alone with the TSR-80, he dismantled it and held his first computer class, showing his friends what made this "computer thing" work.

Since that experience, Bo's passion for understanding how computers work and sharing this knowledge with others has resulted in a rewarding career in IT training. His experience includes Linux, Unix, IT security, DevOps, cloud technologies, and programming languages such as Perl, Python, Tcl, and BASH. He is the founder and lead instructor of One Course Source, an IT training organization.

Dedication

As I close out what will become my 14th book in print (and my 10th with Pearson Publishing), I find myself writing YAD (yet another dedication). I honestly didn't know who I was going to dedicate this book to until just yesterday, when my family had to make one of the most difficult decisions of my life. We needed to end the suffering of our amazing, faithful, and lovable dog, Midnight, a black lab/golden retriever mix.

I was reminded, in a very emotionally painful way, how our furry family members mean so much to us. Midnight brought so much joy and happiness to our family and asked only simple things in return: affection, the opportunity to be close to the members of his pack, and, of course, treats.

He made my world a bit brighter, and while the world is a bit dimmer today, I know that my memory of him will forever enrich my life.

I will miss you, Midnight.

Acknowledgments

To everyone at Pearson who helped make this book come to life, I thank you. I know that this is a team effort, and I appreciate everyone's hard work.

Special thanks go to Nancy, Chris, and Casey for helping me complete this book ahead of schedule!

About the Technical Reviewer

Casey Boyles started working in the IT field more than 30 years ago and quickly moved into systems automation, distributed applications, and database development. Casey later moved into technical training and course development, where he specializes in Layer 0–7 software development, database architecture, systems security, telecommunications, and cloud computing. Casey typically spends his time smoking cigars while "reading stuff and writing stuff."

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: community@informit.com

Reader Services

Register your copy of *CompTIA Linux+ XK0-005 Exam Cram* at www.pearsonitcertification.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to www.pearsonitcertification.com/register and log in or create an account.* Enter the product ISBN **9780137898558** and click **Submit**. When the process is complete, you will find any available bonus content under Registered Products.

*Be sure to check the box indicating that you would like to hear from us to receive exclusive discounts on future editions of this product.

Introduction

Welcome to *CompTIA Linux+ XK0-005 Exam Cram*. This book prepares you for the CompTIA Linux+ XK0-005 certification exam. Imagine that you are at a testing center and have just been handed the passing scores for this exam. The goal of this book is to make that scenario a reality. My name is Bo Rothwell, and I am happy to have the opportunity to help you in this endeavor. Together, we can accomplish your goal to attain the CompTIA Linux+ certification.

Target Audience

The CompTIA Linux+ exam measures the necessary competencies for an entry-level Linux professional with the equivalent knowledge of at least 12 months of hands-on experience in the lab or field.

This book is for persons who have experience working with Linux operating systems and want to cram for the CompTIA Linux+ certification exam—*cram* being the key word.

Linux can be a challenging topic for individuals who are not used to command-line environments. If you don't already have a lot of experience running commands in Linux, I highly recommend trying out the commands presented in this book. Install Linux on a virtual machine and get to practicing!

This book focuses very specifically on the CompTIA Linux+ certification exam objectives. I point this out because you might consider exploring other topics if you want to become proficient. I avoided any non-testable topics because I didn't want to add any confusion as to what you need to study to pass the exam. You might find that some topics that are not exam-testable, like installing Linux and using man pages (to view documentation), will be useful for your understanding of the Linux operating system.

About the CompTIA Linux+ Certification

This book covers the CompTIA Linux+ XK0-005 exam, which you will need to pass to obtain the CompTIA Linux+ certification. This exam is administered by Pearson Vue and can be taken at a local test center or online.

Passing the certification exam proves that you have a solid understanding of the essentials of the Linux operating system, as well as associated Linux topics.

Before doing anything else, I recommend that you download the official CompTIA Linux+ objectives from CompTIA's website. The objectives are a comprehensive bulleted list of the concepts you should know for the exams. This book directly aligns with those objectives, and each chapter specifies the objective it covers.

For more information about how the Linux+ certification can help you in your career or to download the latest objectives, access CompTIA's Linux+ web page at https://www.comptia.org/certifications/linux.

About This Book

This book covers what you need to know to pass the CompTIA Linux+ exam. It does so in a concise way that allows you to memorize the facts quickly and efficiently.

We organized this book into four parts comprising 22 chapters, each chapter pertaining to a particular objective covered on the exams. Each part of the book matches up exactly with one of the four Linux+ exam domains.

A note about studying for the exam: The chapters in this book are in exactly the same order as the corresponding objectives on the Linux+ exam. This provides you with a very clear understanding of where to find content for a specific exam objective, but this does not necessarily mean that you should read the book from cover to cover. For example, Chapter 1, "Linux Fundamentals," does not cover "entry-level" Linux topics. The chapter title matches the Linux+ objective, but if you review the topics, you will discover that they are more "foundational" in nature, not the fundamental topics that an entry-level person would learn. So, where are these fundamental topics in the book? They start in Chapter 2, "Manage Files and Directories."

I mention this because if you are a novice Linux learner and are trying to learn Linux from the ground up using this book, you will likely become overwhelmed within the first chapter. With that said, this really isn't a "learn from the ground up book" but rather a book designed to fill in a bunch of gaps that Linux users often find they have when preparing for the Linux+ exam.

Chapter Format and Conventions

Every chapter of this book follows a standard structure and contains graphical clues about important information. Each chapter includes the following:

- **Opening topics list:** This list defines the CompTIA Linux+ objective covered in the chapter.

- **Topical coverage:** The heart of the chapter, this text explains the topics from a hands-on and theory-based standpoint. In-depth descriptions, tables, and figures are geared toward helping you build your knowledge so that you can pass the exam.
- **Cram Quiz questions:** At the end of each chapter is a brief quiz, along with answers and explanations. The quiz questions and ensuing explanations are meant to help you gauge your knowledge of the subjects you have just studied. If the answers to the questions don't come readily to you, consider reviewing individual topics or the entire chapter. You can also find the Cram Quiz questions on the book's companion web page, at www.pearsonitcertification.com.
- **ExamAlerts and Notes:** These are interspersed throughout the book. Watch out for them!

ExamAlert

This is what an ExamAlert looks like. ExamAlerts stress concepts, terms, hardware, software, or activities that are likely to relate to one or more questions on the exam.

Additional Elements

Beyond the chapters, we have provided some additional study aids for you:

- **CramSheet:** The tear-out CramSheet is located in the beginning of the book. It jams some of the most important facts you need to know for each exam into one small sheet, allowing for easy memorization. It is also available in PDF format on the companion web page. If you have an e-book version, the CramSheet might be located elsewhere in the e-book; run a search for the term "cramsheet," and you should be able to find it.
- **Online Practice Exams:** If you want more practice on the exam objectives, remember that you can access all of the Cram Quiz questions on the Pearson Test Prep software online. You can also create a custom exam, by objective, with the Online Practice Test. Note any objective you struggle with and go to that objective's material in the corresponding chapter. Download the Pearson Test Prep Software online at http://www.pearsonitcertification.com/content/downloads/pcpt/engine.zip.

To access the book's companion website and the software, simply follow these steps:

Step 1. Register your book by going to **PearsonITCertification.com/register** and entering the ISBN **9780137898558**.

Step 2. Answer the challenge questions.

Step 3. Go to your account page and click the **Registered Products** tab.

Step 4. Click the **Access Bonus Content** link under the product listing.

Step 5. Click the **Install Pearson Test Prep Desktop Version** link under the Practice Exams section of the page to download the software.

Step 6. After the software finishes downloading, unzip all the files on your computer.

Step 7. Double-click the application file to start the installation and follow the onscreen instructions to complete the registration.

Step 8. After the installation is complete, launch the application and click the **Activate Exam** button on the My Products tab.

Step 9. Click the **Activate a Product** button in the Activate Product Wizard.

Step 10. Enter the unique access code found on the card in the sleeve in the back of your book and click the **Activate** button.

Step 11. Click **Next** and then click **Finish** to download the exam data to your application.

Step 12. Start using the practice exams by selecting the product and clicking the **Open Exam** button to open the exam settings screen.

You can also use the online version of this software on any device with a browser and connectivity to the Internet including desktop machines, tablets, and smartphones. Follow the directions on the companion website for the book. Note that the offline and online versions will sync together, so saved exams and grade results recorded in one version will be available to you in the other as well.

The Hands-On Approach

As mentioned previously, hands-on experience is very important for understanding Linux. Before taking the exam, you should practice using each command that is listed in this book. Explore the different options that are provided in this book to gain a better understanding of each topic.

Use a virtual machine! It is possible that when you perform some of the administration tasks (partitioning, using firewalls, and so on), you could end up making the operating system unusable. If you use a virtual machine and mess up the original, you can just install a new one (or make use of a cool feature called a snapshot, which allows you to return your operating system to a previous state).

Goals for This Book

Clearly, the primary goal of this book is to prepare you to pass the Linux+ certification exam. With that goal in mind, I did my best to include all relevant exam topics, commands, and information in a very condensed format.

The secondary goal of this book is the help you broaden your understanding of Linux. The folks who developed the objectives for the Linux+ exam did an excellent job of including a wide variety of Linux-related topics. I've done my best to ensure that you have a good understanding of each of these topics, within the bounds of what is testable on the exam.

Linux is a truly remarkable topic, which includes a wide range of capabilities. After achieving your goal of passing the Linux+ exam, I highly encourage you to explore this topic further.

Good luck with the exam and please feel free to reach out to me on LinkedIn, at https://www.linkedin.com/in/bo-rothwell/.

I look forward to hearing about your journey toward passing the Linux+ exam!

—William "Bo" Rothwell

CHAPTER 1

Linux Fundamentals

This chapter covers the following Linux+ XK0-005 exam objective:

- **1.1:** Summarize Linux fundamentals.

Welcome to the first chapter of the book, where you will learn about some of the fundamental features of Linux. In this chapter you will learn about the common locations where Linux files are stored by exploring the Filesystem Hierarchy Standard (FHS). You will also explore the boot process, including BIOS, UEFI, and GRUB2.

Later in this chapter you will learn about device types and how to perform a basic package compilation from source code. The chapter ends with coverage of storage concepts and commands that are used to list hardware information.

This chapter provides information on the following topics: the Filesystem Hierarchy Standard (FHS), the basic boot process, kernel panic, device types in **/dev**, basic package compilation from source, storage concepts, and hardware information.

Filesystem Hierarchy Standard (FHS)

The Filesystem Hierarchy Standard (FHS) defines where files and directories are supposed to be placed on Unix and Linux operating systems. Table 1.1 provides a summary of some of the most important locations.

TABLE 1.1 **FHS Locations**

Location	Description/Contents
/	The root or top-level directory
/bin	Critical binary executables
/boot	Files related to booting the system
/dev	Files that represent physical devices (See the section "Device Types in **/dev**," later in this chapter, for more details.)
/etc	Configuration files for the system
/home	Regular user home directories
/lib	Critical system libraries
/media	Mount points for removable media
/mnt	Temporary mounts
/opt	Optional software packages
/proc	Information related to kernel data and process data (in a virtual file-system, not a disk-based filesystem)
/root	Home directory for the root user account
/sbin	Critical system binary executables
/sys	Files that contain system-related information
/tmp	Temporary files
/usr	Many subdirectories that contain binary executables, libraries, and documentation
/usr/bin	Nonessential binary executables
/usr/lib	Libraries for the executables in the **/usr/bin** directory
/usr/sbin	Nonessential system binary executables
/usr/share	Data that is architecture independent
/var	Data that is variable (that is, that changes in size regularly)
/var/mail	Mail logs
/var/log	Spool data (such as print spools)
/var/tmp	Temporary files

ExamAlert

For the Linux+ XK0-005 exam, you should know where files are stored in Linux. Review Table 1.1 prior to taking the exam.

Basic Boot Process

A *bootloader* is a piece of software that is designed to handle the initial booting of the operating system (OS). Figure 1.1 provides an overview of the boot process and the bootloader's place in this process.

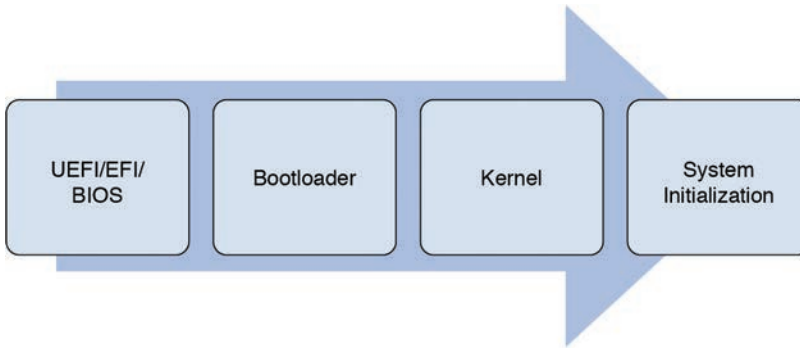

FIGURE 1.1 **Overview of the Boot Process**

UEFI/EFI/BIOS performs sanity checks and then loads the bootloader. See the "Basic Input/Output System (BIOS)/Unified Extensible Firmware Interface (UEFI)" section, later in this chapter, for more details.

The standard Linux bootloader is the Grand Unified Bootloader (GRUB or GRUB2). It is responsible for loading the kernel and associated kernel modules (or *libraries*) stored in a file referred to as the **initramfs** file.

The **initramfs** file contains a mini-root filesystem that has the kernel modules necessary when the system is booting. It is located in the **/boot** filesystem, and there is a unique **initramfs** file for each kernel. The **initramfs** file is created by using the **mkinitrd** command (see the "**mkinitrd**" section, later in this chapter, for more information).

The kernel is loaded from the hard disk, performs some critical boot tasks, and then passes control of the boot process to the system initialization software.

The three different system initialization systems in Linux are SysVinit (the oldest), Upstart, and Systemd (currently the most widely used). The system initialization is responsible for starting system services.

Basic Input/Output System (BIOS)/ Unified Extensible Firmware Interface (UEFI)

Basic input/output system (BIOS), Unified Extensible Firmware Interface (UEFI), and Extensible Firmware Interface (EFI) are all similar in that they are used to provide connections between a system's firmware and the operating system. These programs are provided by the system's manufacturer and are able to start the boot process.

BIOS is only mentioned here in passing. It is older software that has not been officially supported since 2020. However, many UEFI and EFI systems are often referred to as "BIOS," and it is important that you understand this.

UEFI is the successor to EFI and considered the standard in most modern systems.

For the Linux+ XK0-005 exam, you should be aware that UEFI/EFI is the software that starts the boot process. It is the component that starts the bootloader. In addition, it is configurable; for example, you can specify which devices (hard disk, CD/DVD, and so on) to boot from and in which order to attempt to find a bootloader on these devices.

Commands

The sections that follow focus on the commands related to boot software.

mkinitrd

The **initrd** file is created by the **mkinitrd** command, which in turn calls the **dracut** utility:

```
[root@localhost ~]# mkinitrd /boot/initrd-5.17.4.x86_64.img 5.17.4
```

The first argument to the **mkinitrd** command is the name of the **initrd** file that you want to create. The second argument is the version of the kernel.

Note that you rarely use the **dracut** utility directly; however, it is listed as a Linux+ XK0-005 exam objective, so be aware that **mkinitrd** executes the **dracut** command behind the scenes.

See the section "**initrd.img**," later in this chapter, for information on how this file is generated.

grub2-install

Typically the bootloader is installed during the boot process, but it is possible that the bootloader could become corrupt and need to be reinstalled. To install the bootloader, execute the **grub-install** command and provide the device where you want to install GRUB. For example, the following command installs GRUB on the first SATA hard drive:

```
[root@localhost ~]# grub2-install /dev/sda
```

grub2-mkconfig

grub2-mkconfig, which is used only for GRUB2, generates GRUB2 configuration files from the user-editable files located in the **/etc** directory structure. This command converts data from the **/etc/default/grub** file and the files in the **/etc/grub.d** directory into the GRUB2 configuration file (either **/boot/grub/grub.cfg** or **/boot/grub/menu.lst**).

Figure 1.2 provides a visual example.

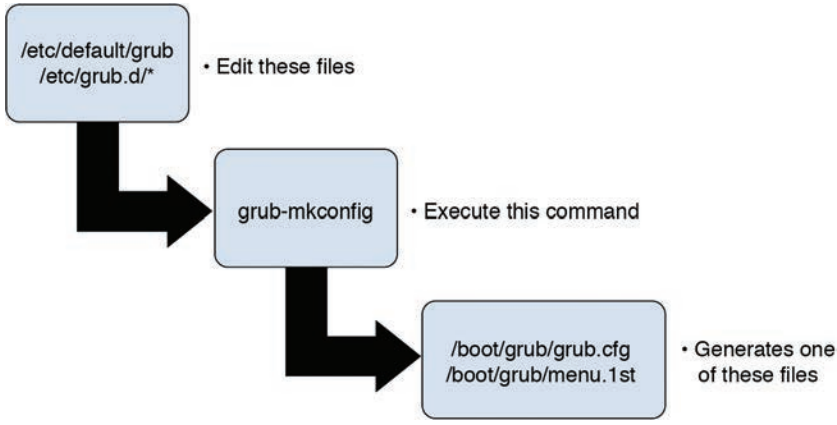

FIGURE 1.2 **The grub2-mkconfig Command**

> **Note**
>
> On some systems, the command is **grub-mkconfig**.

grub2-update

The **grub2-update** command provides another way of running the **grub2-mkconfig** utility. It exists mostly for backward compatibility to some systems that utilized this command to update the GRUB2 configuration files. By default it runs the command **grub-mkconfig -o /boot/grub/grub.cfg**. See the "**grub2-mkconfig**" section, earlier in this chapter, for details about that command.

dracut

Refer to the "**mkinitrd**" section, earlier in this chapter, for more information about **dracut**.

initrd.img

Typically the Linux kernel is configured with few kernel modules enabled by default. Additional modules are normally required during the boot process before the filesystems are mounted. These additional modules are stored within a compressed file called **initrd.img**. See the "**mkinitrd**" section, earlier in this chapter, for information on how this file is generated.

vmlinuz

The **vmlinuz** file is stored in the **/boot** directory. Each version of the kernel has a different **vmlinuz** file, typically with a name that includes the version of the kernel. Here is an example:

```
vmlinuz-4.17.8-200.fc32.x86_64
```

Grand Unified Bootloader Version 2 (GRUB2)

The Grand Unified Bootloader (GRUB), also called Legacy GRUB, is an older bootloader that is rarely used on modern Linux systems. Most of the configuration files and commands on the Linux+ XK0-005 exam focus on GRUB2, which is an improved version of GRUB.

GRUB2 is designed as a replacement for Legacy GRUB. There are several differences between the two, including the following:

- They use different configuration files.

- GRUB2 supports more devices to boot from, including LVM (Logical Volume Management) and software RAID devices.
- GRUB2 supports UEFI and EFI. See the section "Basic Input/Output System (BIOS)/Unified Extensible Firmware Interface (UEFI)," earlier in this chapter, for more details.

Expect Linux+ XK0-005 exam questions to focus on GRUB2, as Legacy GRUB is rarely used in modern Linux distributions.

Boot Sources

GRUB2 allows you to boot from different media. This section focuses on which media you can boot from as well as how to boot from the different media sources.

During the boot process, you can interact with the bootloader. This is normally useful for the following reasons:

- To boot to an alternative stanza
- To modify the existing boot parameters

This interaction starts with the boot menu screen, as shown in Figure 1.3.

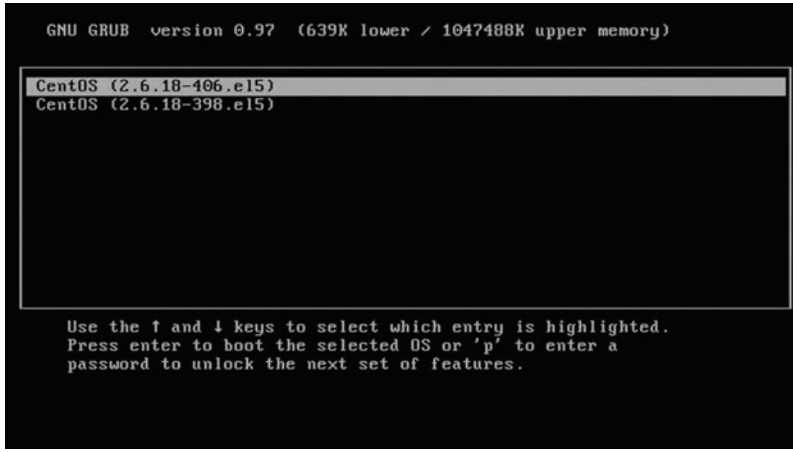

FIGURE 1.3 **The GRUB Boot Menu Screen**

Table 1.2 describes the commands available on the GRUB boot menu screen.

TABLE 1.2 **Commands Available on the GRUB Boot Menu Screen**

Command	Description
Arrow keys	Used to select a stanza.
e	Used to edit the currently selected stanza.
c	Used to enter a GRUB command prompt.
p	Only visible when a password is required to edit a stanza; use **p** to enter the required password.

If you edit a stanza, a new screen with different menu options is provided (see Figure 1.4).

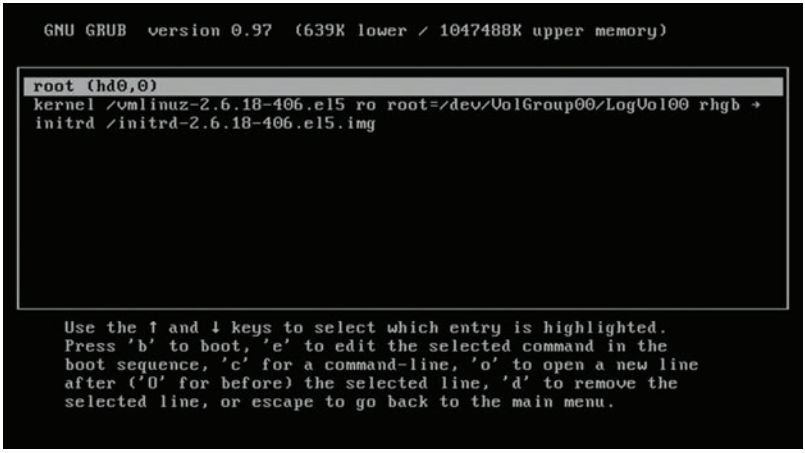

FIGURE 1.4 The GRUB Stanza-Editing Screen

Table 1.3 describes the commands available on the GRUB stanza-editing screen.

TABLE 1.3 **Commands Available on the GRUB Boot Stanza-Editing Screen**

Command	Description
Arrow keys	Used to select a stanza.
e	Used to edit the currently selected line.
c	Used to enter a GRUB command prompt.
o	Used to open (create) a new line below the current line.
O	Used to open (create) a new line above the current line.
d	Used to remove the selected line.

Command	Description
b	Used to boot the current stanza.
[ESC]	Returns you to the main menu.

The rest of this section describes the different boot sources that GRUB2 can boot from.

Preboot eXecution Environment (PXE)

Preboot eXecution Environment (PXE) allows you to boot a system across the network, assuming that a boot server has been created on the network. PXE uses a Dynamic Host Configuration Protocol (DHCP) server to obtain network configuration information, such as an IP address and subnet address.

The boot server listens for PXE boot requests and then provides an operating system to the client system. Typically this operating system calls the client machine to perform an installation, so further PXE boots are not required.

Note that PXE boots are initiated from BIOS/UEFI/EFI software.

Booting from ISO/Universal Serial Bus (USB)

There are several Live Linux distributions that allow you to boot directly from a CD, DVD, or USB device. This technique is referred to as "booting from an ISO" because the file format used to store the operating system on this media is called an *ISO image*.

There are several advantages to and reasons for booting from an ISO:

- The system might be a thin client with no hard drive.
- Booting from a security-based ISO image can be helpful in resolving issues (virus, worms, or other security compromises) on the host OS.
- Some Live Linux distributions can fix problems with booting the host OS.
- An ISO image can also be used to install a new distribution on the system.

In most cases, booting from an ISO image requires instructing BIOS/UEFI/EFI to boot from the drive that contains the ISO media. However, it is possible to configure GRUB to boot from ISO media as well (although this tends to be rarely done).

Kernel Panic

A *kernel panic* occurs when something goes wrong with the kernel and the system crashes. Typically when this happens, data is stored using a feature called **kdump**. A kernel expert can view this data to determine the cause of the crash.

ExamAlert

Using a **kdump** file is a specialized skill and beyond the scope of the Linux+ XK0-005 exam.

Device Types in /dev

The **/dev** filesystem contains device files, which are used to access physical devices (such as hard drives, keyboards, and CPUs) and virtual devices (such as LVM devices, pseudo-terminals, and software RAID devices). The **/dev** filesystem is memory based, not stored on the hard drive.

Table 1.4 describes the key files in **/dev**.

TABLE 1.4 **Key Files in /dev**

File	Description
/dev/sd*	Devices that begin with **sd** in the **/dev** directory are either SATA, SCSI, or USB devices. The device name **/dev/sda** refers to the first device, **/dev/sdb** refers to the second device, and so on. If a device has partitions, they are numbered starting with 1. For example, **/dev/sda1** is the first partition of the first SATA, SCSI, or USB device.
/dev/hd*	Devices that begin with **hd** in the **/dev** directory are IDE-based devices. The device name **/dev/hda** refers to the first device, **/dev/hdb** refers to the second device, and so on. If a device has partitions, they are numbered starting with 1. For example, **/dev/hda1** is the first partition of the first IDE-based device.
/dev/cdrom	This is a symbolic link that points to the first CD-ROM on the system.
/dev/dm*	Devices that begin with **dm** in the **/dev** directory are either software RAID or LVM devices. The device name **/dev/dm-0** refers to the first device, **/dev/dm-1** refers to the second device, and so on.
/dev/tty*	Devices that begin with **tty** in the **/dev** directory are terminal devices. The device name **/dev/tty0** refers to the first device, **/dev/tty1** refers to the second device, and so on.

ExamAlert

Be prepared for Linux+ XK0-005 exam questions related to specific device files described in this section.

Block Devices

A *block device* is any device that is designed to read and write data in chunks (that is, blocks). Block devices are typically storage devices, like USB drives, hard drives, CD-ROMs, and DVDs.

Character Devices

A *character device* is a device that is designed to read and write data in single bits (that is, single characters). An example of a character device is a keyboard, which sends one character to the system at a time.

Special Character Devices

Special character devices are device files that don't represent real physical devices. This includes the **/dev/null**, **/dev/zero**, and **/dev/urandom** files, as described in the sections that follow.

/dev/null

In some cases, you may not want to see either stdout or stderr of a command. For example, consider the following **find** command:

```
[root@OCS ~]$ find /etc -name "hosts"
find: '/etc/named': Permission denied
find: '/etc/polkit-1/localauthority': Permission denied
find: '/etc/polkit-1/rules.d': Permission denied
find: '/etc/pki/rsyslog': Permission denied
find: '/etc/pki/CA/private': Permission denied
find: '/etc/sudoers.d': Permission denied
find: '/etc/grub.d': Permission denied
find: '/etc/phpMyAdmin': Permission denied
/etc/hosts
```

```
find: '/etc/selinux/targeted/modules/active': Permission denied
find: '/etc/webmin/lpadmin': Permission denied
find: '/etc/webmin/iscsi-target': Permission denied
```

Notice the large number of "Permission denied" messages. These messages result from not having permissions to view the contents of specific directories that are located in the search path. You often don't care about such messages and would rather not see them. In such cases, you can redirect these messages to the **/dev/null** device file. This file is called the "bit bucket" and acts as a trash can that never needs to be emptied. Anything sent to the **/dev/null** device is immediately discarded.

Because the error messages in the previous **find** command were sent to stderr, they can be discarded by using the following command:

```
[root@OCS ~]$ find /etc -name "hosts" 2> /dev/null
/etc/hosts
```

See Chapter 13, "Create Simple Shell Scripts to Automate Common Tasks," for details regarding the **2>** symbol, stdout, and stderr.

/dev/zero

The **/dev/zero** file is a special file in Linux that returns null characters. It is often used with utilities like the **dd** command to create large files. See the "**dd**" section in Chapter 2, "Manage Files and Directories," for more details about the **dd** command and how it uses the **/dev/zero** file.

/dev/urandom

The Linux kernel has a feature that can be used to generate random numbers. This feature can be accessed by reading content from the **/dev/urandom** file. This file returns random numbers, which can be useful in software development as well as with some tools that need to create unique values, such as encryption tools.

ExamAlert

Using **/dev/urandom** is beyond the scope of the Linux+ XK0-005 exam.

Basic Package Compilation from Source

Build tools are used to create software that will be used by others. This section explores some of the most useful build tools.

./configure

After you download the source code of a software package, you need to know how to build it and install it. In a real-world situation, you would probably modify the source code first, but that normally requires programming skills that are beyond the scope of this book and the Linux+ XK0-005 exam.

The first step in building the source code is to execute the **configure** script. This may be directly in the top-level directory of the source code, but you may need to look for it, as in the following output:

```
[root@OCS unix]# pwd
/tmp/source/zip/unix
[root@OCS unix]# ls
configure  Makefile  osdep.h  Packaging  README.OS390  unix.c  zipup.h
```

You need to change the permissions of the configure file to make it executable and then run the script as shown in the following example:

```
[root@OCS unix]# su - bo
[bo@OCS unix]$ chmod u+x configure
[bo@OCS unix]$ ./configure
Check C compiler type (optimization options)
  GNU C (-O3)
Check bzip2 support
  Check for bzip2 in bzip2 directory
  Check if OS already has bzip2 library installed
-- Either bzlib.h or libbz2.a not found - no bzip2
Check for the C preprocessor
Check if we can use asm code
Check for ANSI options
Check for prototypes
```

```
Check the handling of const

Check for time_t

Check for size_t

Check for off_t

Check size of UIDs and GIDs

(Now zip stores variable size UIDs/GIDs using a new extra field. This
tests if this OS uses 16-bit UIDs/GIDs and so if the old 16-bit
storage should also be used for backward compatibility.)

  s.st_uid is 4 bytes

  s.st_gid is 4 bytes

-- UID not 2 bytes - disabling old 16-bit UID/GID support

Check for Large File Support

  off_t is 8 bytes

-- yes we have Large File Support!

Check for wide char support

-- have wchar_t - enabling Unicode support

Check for gcc no-builtin flag

Check for rmdir

Check for strchr

Check for strrchr

Check for rename

Check for mktemp

Check for mktime

Check for mkstemp

Check for memset

Check for memmove

Check for strerror

Check for errno declaration

Check for directory libraries

Check for readlink

Check for directory include file

Check for nonexistent include files

Check for term I/O include file

Check for valloc

Check for /usr/local/bin and /usr/local/man
```

```
Check for OS-specific flags
Check for symbolic links
```

The result of the **configure** script is the **Makefile** file. The **Makefile** file provides directions on how to build and install the software. The "**make**" section that follows provides details regarding the next steps in building the software package.

make

The **make** command uses a file named **Makefile** to perform specific operations. The **make** command is a utility for building and maintaining programs and other types of files from source code. A primary function of the **make** utility is to determine which pieces of a large program need to be recompiled and to issue the commands necessary to recompile them. Each **Makefile** is written by the software developer to perform operations related to the software project. Typically this includes the following types of functions:

- An operation to compile the software
- An operation to install the software
- An operation to "clean" previous versions of the compiled code

The following is an example of a simple **Makefile** that has only an install operation:

```
# more /usr/lib/firmware/Makefile
# This file implements the GNOME Build API:
# http://people.gnome.org/~walters/docs/build-api.txt

FIRMWAREDIR = /lib/firmware

all:

install:
        mkdir -p $(DESTDIR)$(FIRMWAREDIR)
        cp -r * $(DESTDIR)$(FIRMWAREDIR)
        rm -f $(DESTDIR)/usbdux/*dux $(DESTDIR)/*/*.asm
        rm $(DESTDIR)$(FIRMWAREDIR)/{WHENCE,LICENSE.*,LICENCE.*}
```

make install

The **install** option for the **make** command is designed to install code from source on the system. It may also include a compile process, depending on how the software developer created the **Makefile**. See the preceding "**make**" section for more details.

Storage Concepts

This section covers some of the essential concepts related to storage devices.

File Storage

With file storage, a remote storage device is used by the local system. The remote storage device is shared to the local system via a network filesystem structure, and users can place files and directories on the remote system. Common network filesystems include the following:

- **nfs:** This network-based filesystem originated on Unix systems. While it is an older filesystem, it has provided a standard way of sharing directory structures between Unix and Linux systems. Newer versions of this filesystem include modern security features and performance improvements.
- **smb:** This filesystem, which is also known as the Samba filesystem, is based on cifs and designed to provide network-based sharing.
- **cifs:** This filesystem is used on Microsoft Windows systems to share folders across the network. Samba utilities on Linux are used to connect to cifs shares.

Block Storage

A block storage device is a physical storage device and typically provides back-end storage for Linux storage systems. Examples of block storage devices include the following:

- Traditional SATA drives
- SSDs
- RAID drives

- Storage area networks (SANs), which can include Fibre Channel Protocol (FCP), Internet Small Computer System Interface (iSCSI), ATA over Ethernet (AoE), and Fibre Channel over Ethernet (FCoE)
- Optical discs (CD-ROM and DVD devices)

Data on block storage devices is accessible via a filesystem. Linux has a variety of local filesystem types, including the following:

- **ext3:** This filesystem, which is an extension of the ext2 filesystem, is designed to be placed on disk-based devices (partitions). While there are several differences between ext2 and ext3, the big change in ext3 was the introduction of journaling. Journaling helps prevent filesystem corruption by creating a log (journal) of changes made to files. In the event of a system crash, the recovery time of an ext3 filesystem should be relatively quick, as the journal can be used to quickly fix corrupted file metadata.
- **ext4:** The ext4 filesystem is a replacement for the ext3 filesystem. It supports larger filesystems and individual file sizes. Performance was improved in this version as well.
- **xfs:** This is another disk-based filesystem that is known for high performance and for handling larger file sizes.

Object Storage

Object storage is actually not a Linux concept; rather, it is typically found in cloud infrastructures. However, the object storage in a cloud infrastructure can be used by Linux, and this storage type is becoming a very popular way to store and share files.

Object storage is a feature in which objects (unstructured data like emails, videos, graphics, text, or any other type of data) can be stored in a cloud environment. Object storage doesn't use traditional filesystem storage features but rather organizes the data into "groups" (similar to a folder in a filesystem). Data is typically accessed using a URL much like one that you would use to access a web page. Object storage is durable and highly available, supports encryption, and can be used in a flexible manner that supports different backup and archiving features. Examples include AWS S3 (Simple Storage Service), Google Cloud Storage, and IBM Cloud Object Storage.

Partition Type

Partitions are used to separate a hard disk into smaller components. Each component can be treated as a different storage device, and a separate filesystem (btrfs, xfs, etx4, and so on) can be created on each partition.

There is a limit to the number of traditional PC-based partitions you can create. Originally only four partitions, referred to as *primary partitions*, were permitted. As more partitions were needed, a technique was created that makes it possible to convert one of the primary partitions into an extended partition. Within an extended partition, you can create additional partitions called *logical partitions*.

In Figure 1.5, **/dev/sda1**, **/dev/sda2**, and **/dev/sda3** are primary partitions. The **/dev/sda4** partition is an extended partition that is used as a container for the **/dev/sda5**, **/dev/sda6**, and **/dev/sda7** logical partitions.

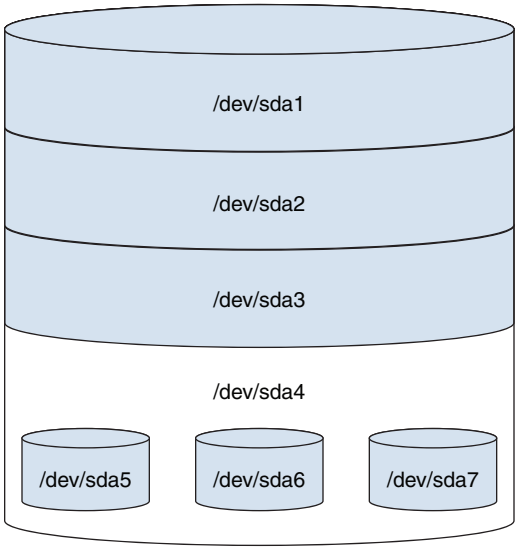

FIGURE 1.5 **Traditional Partition Structure**

On most Linux distributions that use traditional partitions, you are limited to a total of 15 partitions (though a kernel tweak can increase this number to 63).

Traditional partition tables are stored on the master boot record (MBR). A newer partition table, called the GUID partition table (GPT), does not face the same limitations or have the same layout as an MBR partition table.

Several different tools can be used to create or view partitions, including **fdisk**, **parted**, and the GUI-based tool provided by the installation program (which can vary based on the distribution).

Both **fdisk** and **parted** support command-line options, and both of them can be executed as interactive tools. These tools are covered in greater detail in Chapter 3, "Configure and Manage Storage Using the Appropriate Tools."

A *raw device* is a device file that is associated with a block device file (partition, hard disk, and so on). When you create this association, direct access to the block device is available. To create a raw device, use the **raw** command:

```
# raw /dev/raw/raw1 /dev/vda
/dev/raw/raw1: bound to major 252, minor 0
```

Once a raw device is created, you can use commands like the **dd** command to perform actions on the corresponding block file. The **dd** command is often used to create a copy of an entire hard disk.

Master Boot Record (MBR)

MBR partition tables are often referred to as "traditional" partitions, as opposed to newer partition tables such as the GUID partition table. An MBR partition table has the restriction of only permitting four partitions by default. This is an extremely limiting factor for operating systems such as Linux.

However, one of the primary partitions in an MBR partition table can be converted into an extended partition. Additional partitions, called *logical partitions*, can be added within this extended partition. See Figure 1.6 for a visual example.

A note regarding hard disk device names: Hard disks are referred to via device names in the **/dev** directory. IDE-based devices have names that start with **/dev/hd**, whereas SATA, SCSI, and USB devices have names that start with **/dev/sd**. The drives on a system are named starting from **a**, so the first SATA device would be **/dev/sda**, the second SATA device would be **/dev/sdb**, and so on. Partitions are numbered sequentially, starting from **1**, such as **/dev/sda1**, **/dev/sda2**, and **/dev/sda3**.

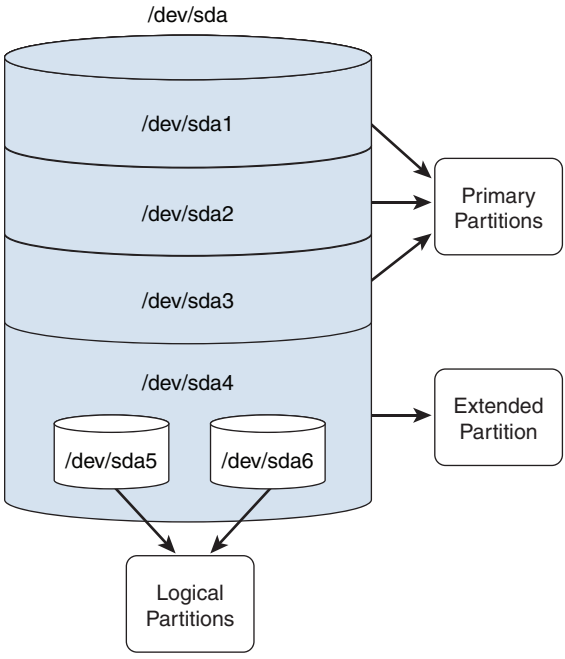

FIGURE 1.6 **Partitions**

GUID (Globally Unique Identifier) Partition Table (GPT)

The GUID partition table (GTP) is a partitioning scheme that is designed to overcome the limitations of MBR (see the "Master Boot Record (MBR)" section, earlier in this chapter). Unlike the MBR, the GPT doesn't have the limitation of four primary partitions. There also isn't a need for extended or logical partitions. The GPT supports up to 128 partitions per hard disk device.

Filesystem in Userspace (FUSE)

By default, only the root user can create filesystems. Filesystem in Userspace (FUSE) is a feature of the Linux kernel that allows regular users to create Linux filesystems.

ExamAlert

Setting up, configuring, and using FUSE is beyond the scope of the Linux+ XK0-005 exam.

Redundant Array of Independent (or Inexpensive) Disks (RAID) Levels

A RAID device is used to provide redundancy. Two or more physical devices can be combined to create a single device that stores data in a way that mitigates data loss in the event that one of the physical storage devices fails.

Note that RAID is covered in more detail in Chapter 3.

Striping

Striping, or RAID 0, is the process of writing to multiple drives as if they were a single device. The writes are performed using striping, in which some data is written to the first drive, then some data is written to the second drive, and so on. See Chapter 3 for more details.

Mirroring

With mirroring, or RAID 1, two or more disk drives appear to be one single storage device. Data that is written to one disk drive is also written to all of the other drives. If one drive fails, the data is still available on the other drives. See Chapter 3 for more details.

Parity

Parity data is a value that is derived from the data stored on the other devices in the RAID. Suppose there are three devices in the RAID, two of which (called devices A and B in this example) store regular filesystem data and one of which (called device C) stores the parity data. If device A fails, then the data that it stored could be rebuilt using a comparison of the data in device B and the parity data in device C. See Chapter 3 for more details.

Listing Hardware Information

This section focuses on several different utilities designed to display information about hardware devices or configure these devices.

lspci

The **lspci** command displays devices attached to the PCI bus.

Syntax:

lspci [*options*]

Here are some of the key options for the **lspci** command:

- **-b** is "bus centric," meaning it displays IRQ (Interrupt Request Line) numbers.
- **-n** displays device numbers rather than names; names typically are stored in **/usr/share/hwdata/pci.ids** or **/usr/share/hwdata/pci.ids.gz**.
- **-nn** displays both device numbers and names.
- **-v** shows verbose messages.
- **-vv** shows even more verbose messages.
- **-vvv** shows the most verbose messages.

Example:

```
[root@OCS ~]# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX
 PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA
 [Natoma/ Triton II]
00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4
 IDE (rev 01)
00:02.0 VGA compatible controller: InnoTek Systemberatung
 GmbH VirtualBox Graphics Adapter
00:03.0 Ethernet controller: Intel Corporation 82540EM
 Gigabit Ethernet Controller (rev 02)
00:04.0 System peripheral: InnoTek Systemberatung GmbH
 VirtualBox Guest Service
```

```
00:05.0 Multimedia audio controller: Intel Corporation
 82801AA AC'97 Audio Controller (rev 01)
00:06.0 USB controller: Apple Inc. KeyLargo/Intrepid USB
00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:0b.0 USB controller: Intel Corporation 82801FB/FBM/FR
 /FW/FRW (ICH6 Family) USB2 EHCI Controller
00:0d.0 SATA controller: Intel Corporation 82801HM/HEM
 (ICH8M/ICH8M-E) SATA Controller [AHCI mode] (rev 02)
```

ExamAlert

The Linux+ XK0-005 exam should not ask specific questions on the output of the **lspci** command, but be ready to answer questions regarding the purpose of the command and the options described here.

lsusb

The **lsusb** command displays devices that are attached to the PCI bus.

Syntax:

lsusb [*options*]

Here are some of the key options for the **lsusb** command:

- **-D** displays a specific USB device (specified as an argument) rather than probing the **/dev/bus/usb** directory and displaying all USB devices.
- **-t** displays USB devices in a tree-like format.
- **-v** shows verbose messages.

Example:

```
[root@OCS Desktop]# lsusb
Bus 001 Device 002: ID 1221:3234 Unknown manufacturer Disk(Thumb
drive)
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

dmidecode

The **dmidecode** command is used to display a description of hardware components. In the following example, the **head** command was used to reduce the large amount of output:

```
[root@OCS ~]# dmidecode | head
# dmidecode 3.1
Getting SMBIOS data from sysfs.
SMBIOS 2.5 present.
10 structures occupying 450 bytes.
Table at 0x000E1000.

Handle 0x0000, DMI type 0, 20 bytes
BIOS Information
        Vendor: innotek GmbH
        Version: VirtualBox
```

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. According to the FHS, where is information related to kernel data and process data stored?
   - ❍ **A. /tmp**
   - ❍ **B. /var**
   - ❍ **C. /usr/lib**
   - ❍ **D. /proc**

2. What is the third stage of the boot process?
   - ❍ **A.** BIOS
   - ❍ **B.** Kernel
   - ❍ **C.** Bootloader
   - ❍ **D.** System initialization

3. PXE uses a _____ server to obtain network configuration information, such as an IP address and subnet address.
   - ❍ **A.** DNS
   - ❍ **B.** NTP

- ❍ **C.** DHCP
- ❍ **D.** SAMBA

4. The **/dev/zero** file is a special file in Linux that returns _____ characters.
   - ❍ **A.** binary
   - ❍ **B.** zero-sized
   - ❍ **C.** null
   - ❍ **D.** None of these answers are correct.

Cram Quiz Answers

1. **D.** The **/proc** directory stores information related to kernel data and process data. The **/tmp** directory is the location for temporary files. The **/var** directory stores data that is variable in size. The **/usr/lib** directory stores libraries for the executables.
2. **B.** The kernel stage is the third stage. BIOS is the first, bootloader is the second, and system initialization is the fourth.
3. **C.** PXE uses a Dynamic Host Configuration Protocol (DHCP) server to obtain network configuration information, such as an IP address and subnet address. The other answers do not provide this configuration information.
4. **C.** The **/dev/zero** file is a special file in Linux that returns null characters.

This page intentionally left blank

CHAPTER 2

Manage Files and Directories

This chapter covers the following Linux+ XK0-005 exam objective:

- **1.2:** Given a scenario, manage files and directories.

The focus of this chapter is on tools and concepts related to managing files and directories. The first section explores a variety of text editors. Many configuration files in Linux are in plaintext format, and knowing how to use text editors is critical for a Linux system administrator.

This chapter also covers how to archive and compress files, as well as how to copy files between different computer systems. This chapter explores a collection of file and directory commands that allow you to perform tasks such as moving, copying, creating, deleting, and displaying information about files and directories.

This chapter provides information on the following topics: file editing; file compression, archiving, and backup; file metadata; soft and hard links; copying files between systems; and file and directory operations.

File Editing

Most configuration files on Linux systems are in plaintext format. This makes it critical to know how to edit text files. This section focuses on common Linux editors: **sed**, **awk**, **printf**, **nano**, and the vi editor.

sed

Use the **sed** utility to make automated modifications to files. The basic format for the **sed** command is **sed** *'s/RE/string/' file*, where *RE* refers to the term *regular expression*, a feature that uses special characters to match patterns.

Here is an example of the **sed** command:

```
[student@OCS ~]$ head -n 5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
[student@OCS ~]$ head -n 5 /etc/passwd | sed 's/bin/----/'
root:x:0:0:root:/root:/----/bash
----:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/s----:/sbin/nologin
adm:x:3:4:adm:/var/adm:/s----/nologin
lp:x:4:7:lp:/var/spool/lpd:/s----/nologin
```

sed is a very powerful utility with a large number of features. Table 2.1 describes some of the most useful **sed** functions.

TABLE 2.1 **sed Functions**

Feature	Description
'/RE/d'	Deletes lines that match the RE from the output of the **sed** command.
'/RE/c*string***'**	Changes lines that match the RE to the value of *string*.
'/RE/a*string***'**	Adds *string* on a line after all lines that match the RE.
'/RE/i*string***'**	Adds *string* on a line before all lines that match the RE.

The **sed** command has two important modifiers (that is, characters added to the end of the **sed** operation):

- **g:** Means "global." By default, only the first RE pattern match is replaced. When the **g** modifier is used, all replacements are made. Figure 2.1 shows an example.
- **i:** Means "case-insensitive." This modifier matches an alpha character regardless of its case. So, the command **sed 's/a/-/i'** would match either **a** or **A** and replace it with the **-** character.

The **sed** command can also change the original file (instead of displaying the modified data to the screen). To change the original file, use the **-i** option.

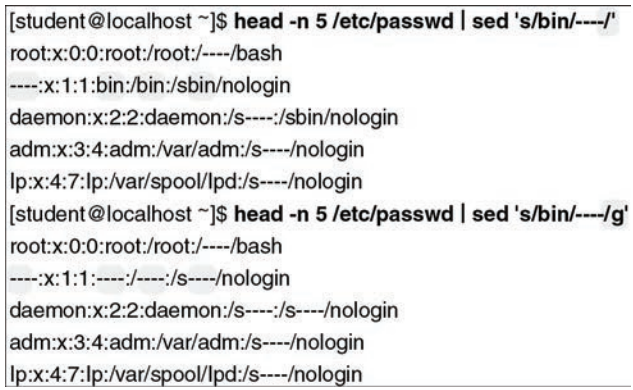

FIGURE 2.1 **The g Modifier**

ExamAlert

The sed command is an extremely powerful tool with many features. For the Linux+ XK0-005 exam, focus on the features described in this section.

awk

The **awk** command is used to modify text that is in a simple database format. Consider the following example:

```
[student@OCS ~]$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

With the **awk** command, you can break apart the data shown in this example into different fields and then perform actions on that data, as shown here:

```
[student@OCS ~]$ head /etc/passwd | awk -F: '{print $1,$7}'
root /bin/bash
bin /sbin/nologin
daemon /sbin/nologin
adm /sbin/nologin
lp /sbin/nologin
sync /bin/sync
shutdown /sbin/shutdown
halt /sbin/halt
mail /sbin/nologin
operator /sbin/nologin
```

In this example, the **-F** option is used as a field separator, and each field is assigned to a variable, as described in Table 2.2.

TABLE 2.2 **awk Field Variables**

Variable	Description
$1, **$2**, and so on	The field variables
$0	The entire line
NF	The number of fields on the line (Do not use the **$** character for this variable.)
NR	The current line number

Table 2.3 lists some important options of the **awk** command.

TABLE 2.3 **Important awk Command Options**

Option	Description
-F	Used to specify the field separator.
-f	Used to specify a file that contains the **awk** commands to execute.

printf

The **printf** command is sometimes used by BASH scripters to format data before displaying it to the user who is running the script. This command can help you format data such as digits and strings. For example, compare the

output of the following two commands (where the **echo** command is a simpler text-printing command):

```
[student@OCS ~]$ total=10.000
[student@OCS ~]$ echo "$total"
10.000
[student@OCS ~]$ printf "%5f \n" $total
10.000000
```

Note how **%5f** changed the precision of the number to be five places after the decimal point. Also note that **\n** produced a newline character, which, unlike with the **echo** command, is not automatically printed.

nano

The **nano** editor is a non-GUI editor that provides a handy "cheat sheet" at the bottom of the screen. See Figure 2.2 for an example.

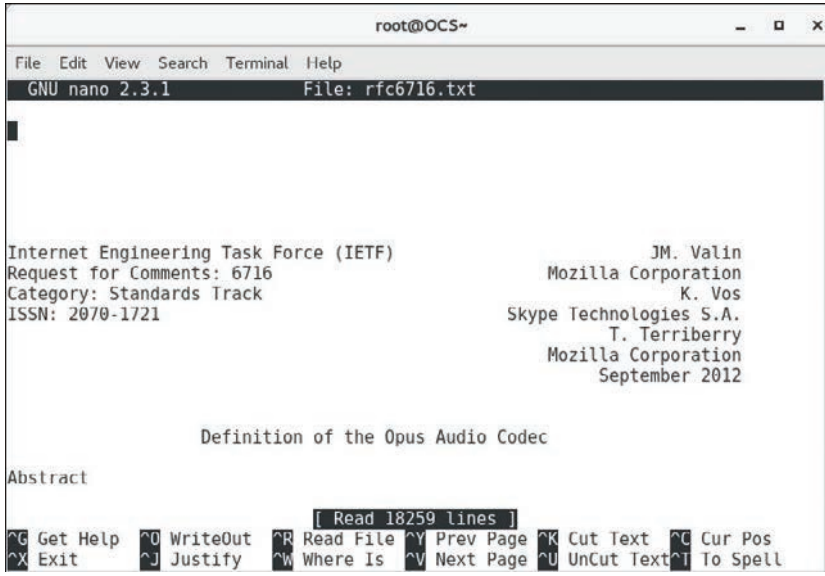

FIGURE 2.2 The nano Editor

You can find commands in **nano** by holding down the Ctrl key and then pressing another key. For example, **Ctrl-x** exits the **nano** editor. In the list of commands at the bottom of the screen, the **^** symbol represents the Ctrl key. Also note that **^X** doesn't mean **Ctrl-Shift-x**; rather, it means just **Ctrl-x**.

You should be aware of the commonly used **nano** commands in Table 2.4.

TABLE 2.4 **Commonly Used nano Commands**

Command	Description
^G	Displays a help document. Note that when the help document appears, different commands are displayed at the bottom of the screen.
^O	Saves a file.
^X	Exits the editor.
^K	Cuts a line (that is, deletes the line and places it in memory). **^K** can be used to cut multiple lines into memory.
^U	Uncuts a line. Note that this isn't an undo operation but rather a paste operation.
^^	Marks the beginning and ending text. Marked text can be copied using **Alt-^**. Use one **^^** to start the marking, use arrow keys to select the text to mark, and then use a final **^^** to end the marking. Note that in **^^**, the second **^** is an actual **^** character, and the first **^** represents the control character.
^W	Searches for text in the current document.
^F	Moves forward one screen.
^B	Moves back one screen.
^n	Moves to a specific line number. For example, **^5** moves to line 5.
^C	Displays the current position in the document.

vi

The vi editor is a standard text editor for both Linux and Unix environments. Although it may not be as user friendly as other editors, it has a few important advantages:

- The vi editor (or vim, which is an improved version of the vi editor) is on every Linux distribution (and all Unix flavors). This means if you know how to edit files with the vi editor, you can always edit a file, regardless of which distribution you are working on.
- Because the vi editor is a command-line-only editor, it does not require a graphical user interface (GUI). This is important because many Linux servers do not have a GUI installed, which means you can't use GUI-based text editors.
- When you understand vi well, you will find that it is an efficient editor, and you can edit files very quickly using it compared to using most

other editors. All vi commands are short and keyboard based, and you don't have to spend time taking your hands off the keyboard to use the mouse.

To edit a new file with the vi editor, you can just type the command with no arguments or type **vi** *filename*.

> **Note**
>
> The vim editor is an improved text editor that has additional features that are not available in the vi editor. Many Linux distributions use the vim editor by default. One advantage of the vim editor is that it includes all the features and commands of the vi editor. So, if you learned to use the vi editor 30 years ago, your knowledge will still apply in the vim editor. All of the commands in this chapter work in both the vi and vim editors.

The vi editor was designed to only use a keyboard, which poses a challenge because sometimes a key should execute a command and sometimes a key should represent a character to insert into the document. To allow the keys to perform different tasks, vi has three modes of operation:

- **Command mode:** This is the default mode, and when you open vi, you are placed in the command mode. In this mode, you can perform commands that can move around the screen, delete text, and paste text.
- **Insert mode:** While you're in insert mode, any key typed appears in your document as new text. When you are finished adding new text, you can return to the default mode (that is, the command mode) by pressing the Esc key.
- **Last line mode:** This mode, also called the *ex mode*, allows you to perform more complex operations, such as saving a document to a file with a different name. To enter last line mode from the command mode, press the **:** key. After you enter a command and press Enter, the command is executed, and normally you are returned to the command mode. In some cases, you may need to press the Esc key to return to the command mode.

> **Note**
>
> You cannot move from the insert mode to the last line mode or vice versa. To move to the insert mode or the last line mode, you first must be in the command mode. Pressing the Esc key places you in the command mode.

To search for files while in the vi editor, you can use either the **/** or the **?** character when you are working in the command mode. When you type either the **/** or the **?** character, a prompt appears in the bottom-left portion of the screen, as shown in Figure 2.3.

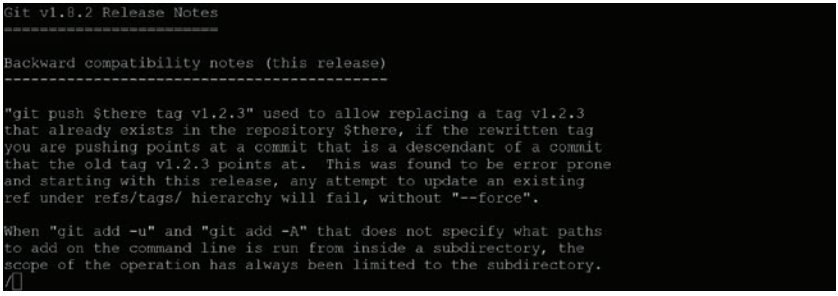

FIGURE 2.3 **Searching for Files in the vi Editor**

At the **/** or **?** prompt, type the value you want to search for. You can use regular expressions within this value. For example, if you type **/^the**, the vi editor searches for the next occurrence of the string **"the"** that appears at the beginning of a line.

The **/** character performs a forward search, and **?** searches backward in the document. If you don't find what you are searching for, you can use the **n** character to find the next match. When the search is started with a **/** character, the **n** character continues the search forward in the document. When the search is started with a **?** character, the **n** character continues the search backward in the document.

To reverse the current search, use the **N** character. For example, suppose you start the search by typing **?^the**. Using the **N** character results in searching forward in the document.

While in the command mode, you can move around the screen by using the keys described in Table 2.5.

TABLE 2.5 **vi Command Mode Navigation Keys**

Key	Description
h	Moves one character to the left.
j	Moves one line down.
k	Moves one line up.
l	Moves one character to the right.

While in the command mode, you can enter the insert mode by using one of the keys described in Table 2.6.

TABLE 2.6 **vi Insert Mode Navigation Keys**

Key	Description
i	Enters the insert mode before the character the cursor is on.
I	Enters the insert mode before the beginning of the current line.
o	Opens a new line below the current line and enters the insert mode.
O	Opens a line above the current line.
a	Enters the insert mode after the character the cursor is on.
A	Enters the insert mode after the end of the current line.

While in the command mode, you can modify text by using the keys described in Table 2.7.

TABLE 2.7 **vi Command Mode Keys to Modify Text**

Key	Description
c	The **c** key is combined with other keys to change data. For example, **cw** changes the current word, and **c$** changes from the cursor position to the end of the line. When finished making your changes, press the Esc key.
d	The **d** key is combined with other keys to delete data. For example, **dw** deletes the current word, and **d$** deletes from the cursor position to the end of the line. All of the deleted data is stored in a buffer and can be pasted back into the document with the **p** or **P** key.
p	After cutting text with the **d** key or copying text with the **y** key, you can paste with the **p** or **P** key. A lowercase **p** pastes after the current cursor, whereas an uppercase **P** pastes before the cursor.
y	The **y** key is combined with other keys to copy data. For example, **yw** copies the current word, and **y$** copies from the cursor position to the end of the line. All of the copied data is stored in a buffer and can be pasted back into the document with the **p** or **P** key.
dd	The **dd** command deletes the current line.
yy	The **yy** command copies the current line.

While in the command mode, you can save and/or quit a document by using the keys described in Table 2.8.

TABLE 2.8 **vi Command Mode Keys to Save/Quit a Document**

Key	Description
ZZ	Saves and quits the document. This is equivalent to **:wq**.
:w!	Forces the vi editor to write changes in the document to the file.

Key	Description
:q!	Forces the vi editor to quit, even if changes in the file have not been saved.
:e!	Opens a new file to edit and forgets all changes in the document since the last write. This requires a filename argument as an option (for example, **:e! myfile.txt**).

ExamAlert

Expect the Linux+ XK0-005 exam to include more questions about the vi editor than about the nano editor.

File Compression, Archiving, and Backup

When disaster strikes, such as a hard disk failure or a natural disaster, data may become corrupted. Using archive and restore utilities helps limit the risks involved with data loss and makes it easier to transfer information from one system to another. This section focuses on these utilities.

ExamAlert

Many of the options for compression commands **(gzip, bzip2**, and **zip)** are the same. This makes them easier to remember for the Linux+ XK0-005 exam.

gzip

Use the **gzip** command to compress files, as shown here:

```
[student@OCS ~]$ ls -lh juju
-rwxr-xr-x 1 vagrant vagrant 109M Jan 10 09:20 juju
[student@OCS ~]$ gzip juju
[student@OCS ~]$ ls -lh juju.gz
-rwxr-xr-x 1 vagrant vagrant 17M Jan 10 09:20 juju.gz
```

Note that the **gzip** command replaces the original file with the compressed file.

Table 2.9 details some important **gzip** options.

TABLE 2.9 **gzip Command Options**

Option	Description
-c	Writes output to stdout and does not replace the original file. You can use redirection to place output data into a new file (for example, **gzip -c juju > juju.gz**).
-d	Decompresses the file. (You can also use the **gunzip** command for decompression.)
-r	Compresses recursively (that is, compresses all files in the directory and its subdirectories).
-v	Enters verbose mode and displays a percentage of compression.

ExamAlert

The **gzip**, **xz**, and **bzip2** commands are very similar to one another. The biggest difference is the technique used to compress files. The **gzip** command uses the Lempel-Ziv coding method, whereas the **bzip2** command uses the Burrows-Wheeler block-sorting text-compression algorithm and Huffman coding. The **xz** command uses the LZMA and LZMA2 compression methods.

Use the **gunzip** command to decompress gzipped files, as shown here:

```
[student@OCS ~]$ ls -lh juju.gz
-rwxr-xr-x 1 vagrant vagrant 17M Jan 10 09:20 juju.gz
[student@OCS ~]$ gunzip juju
[student@OCS ~]$ ls -lh juju
-rwxr-xr-x 1 vagrant vagrant 109M Jan 10 09:20 juju
```

bzip2

Use the **bzip2** command to compress files, as shown here:

```
[student@OCS ~]$ ls -lh juju
-rwxr-xr-x 1 vagrant vagrant 109M Jan 10 09:20 juju
[student@OCS ~]$ bzip2 juju
[student@OCS ~]$ ls -lh juju.bz2
-rwxr-xr-x 1 vagrant vagrant 14M Jan 10 09:20 juju.bz2
```

Note that the **bzip2** command replaces the original file with the compressed file.

Table 2.10 details some important **bzip2** options.

TABLE 2.10 **bzip2 Command Options**

Option	Description
-c	Writes output to stdout and does not replace the original file. You can use redirection to place output data into a new file (for example, **bzip2 -c juju > juju.bz**).
-d	Decompresses the file. (You can also use the **bunzip2** command.)
-v	Enters verbose mode and displays a percentage of compression.

ExamAlert

The **gzip**, **xz**, and **bzip2** commands are very similar to one another. The biggest difference is the technique used to compress files. The **gzip** command uses the Lempel-Ziv coding method, whereas the **bzip2** command uses the Burrows-Wheeler block-sorting text-compression algorithm and Huffman coding. The **xz** command uses the LZMA and LZMA2 compression methods.

zip

The **zip** command is used to merge multiple files into a single compressed file. To create a compressed file named **mail.zip** that contains all the files in the **/etc/mail** directory, use the following format:

```
[student@OCS ~]$ zip mail /etc/mail*
 adding: etc/mail/ (stored 0%)
 adding: etc/mailcap (deflated 53%)
 adding: etc/mailman/ (stored 0%)
 adding: etc/mail.rc (deflated 49%)
```

Table 2.11 details some important **zip** options.

TABLE 2.11 **zip Command Options**

Option	Description
-d	Decompresses the file. (You can also use the **unzip** command.) Note that the zipped file is not deleted.
-v	Enters verbose mode and displays percentage of compression.
-u	Updates a **.zip** file with new content.
-r	Zips recursively, meaning you can specify a directory, and all of the contents in that directory (including all subdirectories and their contents) are zipped.
-x *file(s)*	Excludes the specified *file(s)* from the **.zip** file.

ExamAlert

Remember that **zip** merges multiple files together, whereas **bzip2** and **gzip** do not.

tar

The purpose of the **tar** command is to merge multiple files into a single file. To create a tar file named **sample.tar**, execute the following:

```
tar -cf sample.tar files_to_merge
```

To list the contents of a **.tar** file, execute the following:

```
tar -tf sample.tar
```

To extract the contents of a **.tar** file, execute the following:

```
tar -xf sample.tar
```

Table 2.12 details some important **tar** options.

TABLE 2.12 **tar Command Options**

Option	Description
-c	Creates a **.tar** file.
-t	Lists the contents of a **.tar** file.
-x	Extracts the contents of a **.tar** file.
-f	Specifies the name of the **.tar** file.
-v	Enters verbose mode and provides more details about what the command is doing.
-A	Appends new files to an existing **.tar** file.
-d	Compares a **.tar** file and the files in a directory and determines the differences between them.
-u	Updates by appending newer files to an existing **.tar** file.
-j	Compresses/uncompresses the **.tar** file using the **bzip2** utility.
-J	Compresses/uncompresses the **.tar** file using the **xz** utility.
-z	Compresses/uncompresses the **.tar** file using the **gzip** utility.

xz

Use the **xz** command to compress files, as shown here:

```
[student@OCS ~]$ ls -lh juju
-rwxr-xr-x 1 vagrant vagrant 109M Jan 10 09:20 juju
[student@OCS ~]$ xz juju
[student@OCS ~]$ ls -lh juju.xz
-rwxr-xr-x 1 vagrant vagrant 11M Jan 10 09:20 juju.xz
```

Table 2.13 details some important **xz** options.

TABLE 2.13 **xz Command Options**

Option	Description
-c	Writes output to stdout and does not replace the original file. You can use redirection to place output data into a new file (for example, **xz -c juju > juju.xz**).
-d	Decompresses the file. (You can also use the **unxz** command.)
-l	Lists information about an existing compressed file (for example, **xz -l juju.xz**).
-v	Enters verbose mode and displays the percentage of compression.

ExamAlert

The **gzip**, **xz**, and **bzip2** commands are very similar to one another. One noticeable difference is the technique used to compress files. The **gzip** command uses the Lempel-Ziv coding method, whereas the bzip2 command uses the Burrows-Wheeler block-sorting text-compression algorithm and Huffman coding. The **xz** command uses the LZMA and LZMA2 compression methods.

cpio

The purpose of the **cpio** command is to create archives. You can create an archive of files by sending the filenames into the command as stdin, as in the following example:

```
[student@OCS ~]$ find /etc -name "*.conf" | cpio -ov > conf.cpio
```

Table 2.14 details some important **cpio** options.

TABLE 2.14 **cpio Command Options**

Option	Description
-d	Used with the **-i** option to extract the directory structure as well as the files in the **cpio** file.
-i	Extracts data from a **cpio** file; the file should be provided via STDIN (for example, **cpio -i < conf.cpio**).
-o	Creates an archive (output file).
-t	Lists the table of contents of a **cpio** file.
-v	Enters verbose mode.

dd

The **dd** command can perform multiple operations related to backing up data and creating files. One common use is to make a backup of an entire drive; for example, the following command backs up the entire **/dev/sdb** device to the **/dev/sdc** device:

```
[student@OCS ~]$ dd if=/dev/sdb of=/dev/sdc bs=4096
```

Another use of the **dd** command is to create a large file that can be used as a swap file:

```
[student@OCS ~]$ dd if=/dev/zero of=/var/swapfile bs=1M count=50
```

Table 2.15 details some important **dd** options.

TABLE 2.15 **dd Command Options**

Option	Description
if=	Specifies the input file.
of=	Specifies the output file.
bs=	Specifies the block size.
count=	Indicates the number of blocks to create/transfer.

File Metadata

File metadata is information about a file, other than the file contents. Two useful commands for displaying file metadata are covered in this section: the **stat** command and the **file** command.

stat

A file or directory consists of several components. Many of these components, such as the owner and permissions, are stored in a filesystem element called an inode.

Everything about a file except for the data in the file and the filename is stored in the inode. Each file is given an inode number that is unique for the filesystem in which the file resides.

The inode of a file contains the following information:

- Unique inode number
- User owner
- Group owner
- Mode (permissions and file type)
- File size
- Timestamps:
  - Last time the file contents were modified
  - Last time the inode data was modified
  - Last time the file was accessed
- Pointers (references to the data block locations that contain the file data)

You can see this inode information with the **stat** command, as shown here:

```
[root@OCS ~]$ stat /etc/passwd
 File: '/etc/passwd'
 Size: 2597 Blocks: 8 IO Block: 4096 regular file
Device: fc01h/64513d Inode: 33857 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid:
 ( 0/ root)
Access: 2018-10-12 12:54:01.126401594 -0700
Modify: 2018-09-08 12:53:48.371710687 -0700
Change: 2018-09-08 12:53:48.371710687 -0700
 Birth: -
```

file

The **file** command reports the type of contents in a file. Here are some examples:

```
[student@localhost ~]$ file /etc/hosts
/etc/hosts: ASCII text
[student@localhost ~]$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=aa7ff68f13de25936a098016243ce57c3c982e06, stripped
[student@localhost ~]$ file /usr/share/doc/pam-1.1.8/html/
sag-author.html
/usr/share/doc/pam-1.1.8/html/sag-author.html: HTML document,
UTF-8 Unicode text, with very long lines
```

Soft and Hard Links

There are two different types of link files: hard links and soft (also called symbolic) links. Understanding these link types is important when determining if you should link a file or make a file copy. This section covers the purposes of links and how to create links using the **ln** command.

Symbolic (Soft) Links

When you create a soft link, the original file contains the data, and the link file points to the original file. Any changes made to the original file also appear to be in the linked file because using the linked file always results in following the link to the target file. Deleting the original file results in a broken link, making the link file worthless and resulting in complete data loss.

Figure 2.4 demonstrates soft links.

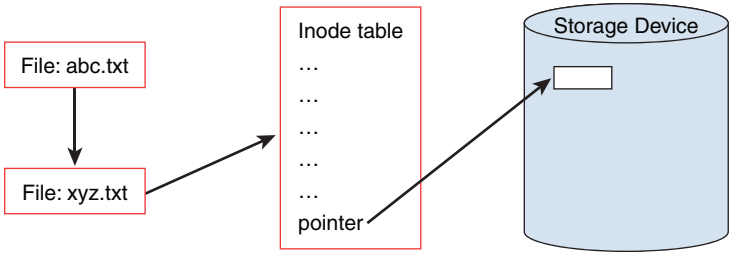

FIGURE 2.4 **Soft Links**

In Figure 2.4, the **abc.txt** file is soft-linked to the **xyz.txt** file. The **abc.txt** file points to the filename **xyz.txt**, not the same inode table. (Although not shown in this figure, the **abc.txt** file has its own inode table.) When the process that is accessing the link file follows the link, the data for the **xyz.txt** file is accessible via the **abc.txt** file.

Copying files results in a complete and separate copy of the data. Changes in the original file have no effect on the data in the copied file. Changes in the copied file have no effect on the data in the original file. Deleting one of these files has no impact on the other file.

To create a link, execute the **ln** command in the following manner: **ln [-s]** *target_file link_file*. You can create a soft link to any file or directory:

```
[root@OCS ~]$ ln -s /boot/initrd.img-3.16.0-30-generic initrd
```

The **ls** command can be used to view both soft and hard links. Soft links are very easy to see because the target file is displayed when executing the **ls -l** command, as shown here:

```
[root@OCS ~]$ ls -l /etc/vtrgb
lrwxrwxrwx 1 root root 23 Jul 11 2015
 /etc/vtrgb -> /etc/alternatives/vtrgb
```

Hard Links

When you create a hard link to a file, there is no way to distinguish the "original" file from the "linked" file. They are just two filenames that point to the same inode and, therefore, the same data. (See the "inodes" section in this chapter for more details.) If you have 10 hard-linked files and you delete any 9 of these files, the data is still maintained in the remaining file.

Figure 2.5 demonstrates hard links.

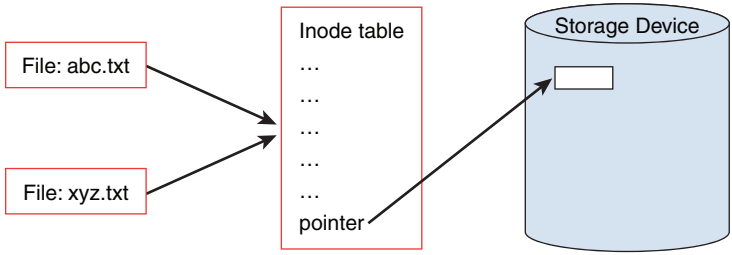

FIGURE 2.5 **Hard Links**

In Figure 2.5, the **abc.txt** and **xyz.txt** files are hard-linked together. This means that they share the same inode tables. An **…** in the inode table represents metadata—information about the file such as the user owner and permissions. Included with this metadata are pointers that refer to blocks within the storage device where the file data is stored.

To create a link, execute the **ln** command in the following manner: **ln [-s]** *target_file link_file*. For example, to create a hard link from the **/etc/hosts** file to a file in the current directory called **myhosts**, execute the following command:

```
[root@OCS ~]$ ln /etc/hosts myhosts
```

Hard-linked files share the same inode. You can only make a hard link to a file (not a directory) that resides on the same filesystem as the original file. Creating hard links to files on another filesystem or to directories results in errors, as shown here:

```
[root@OCS ~]$ ln /boot/initrd.img-3.16.0-30-generic initrd
ln: failed to create hard link 'initrd' =>
 '/boot/initrd.img-3.16. 0-30-generic': Invalid cross-device link
[root@OCS ~]$ ln /etc myetc
ln: '/etc': hard link not allowed for directory
```

The **ls** command can be used to view both soft and hard links. Hard links are more difficult because a hard link file shares an inode with another filename. For example, the value **2** after the permissions in the following output indicates that this is a hard link file:

```
[root@OCS ~]$ ls -l myhosts
-rw-r--r-- 2 root root 186 Jul 11 2015 myhosts
```

To view the inode number of a file, use the **-i** option to the **ls** command:

```
[root@OCS ~]$ ls -i myhosts
263402 myhosts
```

Then use the **find** command to search for files with the same inode:

```
[root@OCS ~]$ find / -inum 263402 -ls 2>/dev/null
263402 4 -rw-r--r-- 2 root root 186 Jul 11 2015 /root/myhosts
263402 4 -rw-r--r-- 2 root root 186 Jul 11 2015 /etc/hosts
```

ExamAlert

Know the difference between hard and soft links. You are likely to get a question on the Linux+ XK0-005 exam that tests your understanding of the differences.

Copying Files Between Systems

Any filesystem is bound to have thousands of files and directories that need to be managed. This section focuses on the Linux commands used to manage these filesystem objects.

rsync

The **rsync** command is useful in copying files remotely across the network. It is typically used in situations where changes from previous files need to be copied over because it handles this more efficiently than other remote copy methods.

Syntax:

```
rsync [options] source destination
```

Table 2.16 describes some important **rsync** options.

TABLE 2.16 **rsync Command Options**

Option	Description
-t	Preserves the original modification timestamp.
-v	Enters verbose mode.
-r	Copies recursively (that is, transfers entire directories).

Option	Description
-l	Maintains symbolic links.
-p	Preserves original permissions.

scp

The **scp** command is used to copy files to and from remote systems via the Secure Shell service. To copy a file from your local machine to a remote machine, use the following syntax:

```
scp filename user@machine:/directory
```

In this syntax, *user* is an account name on the remote system, *machine* is the remote system, and */directory* represents where you want to store the file.

Table 2.17 describes some important options of the **scp** command.

TABLE 2.17 **scp Command Options**

Option	Description
-P *port*	Specifies the port number to connect to. Typically SSH servers use port 22, and that is the default for the **scp** command.
-p	Attempts to preserve the timestamps and permissions of the original file.
-r	Recursively copies entire directories.
-v	Enters verbose mode.

nc

The man page of the **nc** command provides an excellent summary of the **nc** command:

> The nc (or netcat) utility is used for just about anything under the sun involving TCP or UDP. It can open TCP connections, send UDP packets, listen on arbitrary TCP and UDP ports, do port scanning, and deal with both IPv4 and IPv6. Unlike telnet(1), nc scripts nicely, and separates error messages onto standard error instead of sending them to standard output, as telnet(1) does with some.

There are quite a few uses for the **nc** command. For example, suppose you want to know if a specific port is being blocked by your company firewall before you bring online a service that makes use of this port. On the internal server, you can have the **nc** command listen for connections on that port:

```
[root@server ~]# nc -l 3333
```

The result should be a blank line below the **nc** command. Next, to connect, on a remote system outside your network, you could run the following **nc** command (replacing *server* with the resolvable hostname or IP address of the local system):

```
[root@client Desktop]# nc server 3333
```

If the connection is established, you see a blank line under the **nc** command line. If you type something on this blank line and press the Enter key, then what you typed appears below the **nc** command on the server. Actually, the communication works both ways: What you type on the server below the **nc** command appears on the client as well.

The following are some useful options to the **nc** command:

- **-w:** This option is used on the client side to close a connection automatically after a timeout value is reached. For example, **nc -w 30 server 333** closes the connection 30 seconds after it has been established.
- **-6:** Use this option to enable IPv6 connections.
- **-k:** Use this option to keep server processes active, even after the client disconnects. The default behavior is to stop the server process when the client disconnects.
- **-u:** Use UDP connections rather than TCP connections (the default). This is important for correctly testing firewall configurations as a UDP port might not be blocked while the TCP port is blocked.

You can also use the **nc** command to display open ports, much the way you use the **netstat** command:

```
[root@onecoursesource Desktop]# nc -z localhost 1000-4000
Connection to localhost 3260 port [tcp/iscsi-target] succeeded!
Connection to localhost 3333 port [tcp/dec-notes] succeeded!
```

The **-z** option can also be used to port scan a remote host.

There is a useful way to use the **nc** command to transfer all sorts of data. The format you use, assuming that the transfer is from the client to the server, is shown below (where you replace *cmd* with an actual command):

On the server: **nc -l 3333 |** *cmd*

On the client: *cmd* **| nc server 3333**

For example, you can transfer an entire **/home** directory structure from the client to the server with the **tar** command by first executing the following on the server:

```
nc -l 333 | tar xvf -
```

Then, on the client command, you execute the following command:

```
tar cvf - /home | nc server 333
```

The client merges the contents of the **/home** directory structure into a tar ball. The **-** tells the **tar** command to send this output to standard output. The data is sent to the server via the client's **nc** command, and then the server's **nc** command sends this data to the **tar** command. As a result, the **/home** directory from the client is copied into the current directory of the server.

File and Directory Operations

mv

The **mv** command moves or renames a file.

Example:

```
mv /tmp/myfile ~
```

Table 2.18 describes some important options of the **mv** command.

TABLE 2.18 **mv Command Options**

Option	Description
-i	Provides an interactive prompt if the move process would result in overwriting an existing file.
-n	Prevents an existing file from being overwritten.
-v	Enters verbose mode and describes actions taken when moving files and directories.

cp

The **cp** command is used to copy files or directories. Here's the syntax for this command:

```
cp [options] file|directory destination
```

file|*directory* is the file or directory to copy. *destination* is where to copy the file or directory to. The following example copies the **/etc/hosts** file into the current directory:

```
[student@OCS ~]$ cp /etc/hosts .
```

Note that the destination *must* be specified (hence the **.** character that represents the current directory in this example).

Table 2.19 describes some important options of the **cp** command.

TABLE 2.19 **cp Command Options**

Option	Description
-i	Provides an interactive prompt if the copy process results in overwriting an existing file.
-n	Prevents an existing file from being overwritten.
-r	Recursively copies the entire directory structure.
-v	Enters verbose mode and describes actions taken when copying files and directories.

mkdir

The **mkdir** command creates a directory.

Example:

```
mkdir test
```

Table 2.20 describes some important **mkdir** options.

TABLE 2.20 **mkdir Command Options**

Option	Description
-m *perm*	Sets the permissions for the new directory rather than using the umask value.
-p	Creates parent directories, if necessary; for example, **mkdir /home/bob/data/january** creates all the directories in the path specified if they don't already exist.
-v	Enters verbose mode and prints a message for every directory that is created.

rmdir

The **rmdir** command is used to delete empty directories. This command fails if the directory is not empty. (Use **rm -r** to delete a directory and all the files within the directory.)

Example:

```
rmdir data
```

ls

The **ls** command is used to list files in a directory. Table 2.21 describes some important options of the **ls** command.

TABLE 2.21 **ls Command Options**

Option	Description
-a	Lists all files, including hidden files.
-d	Lists the directory name, not the contents of the directory.
-F	Appends a character to the end of the file to indicate its type; examples include * for an executable file, / for a directory, and @ for a symbolic link file.
-h	When used with the **-l** option, provides file sizes in human-readable format.
-i	Displays each file's inode value.
-l	Displays a long listing (refer to the figure).
-r	Reverses the output order of the file listing.
-S	Sorts by file size.
-t	Sorts by modification time (with newest files listed first).

The output of the **ls -l** command includes one line per file, as demonstrated in Figure 2.6.

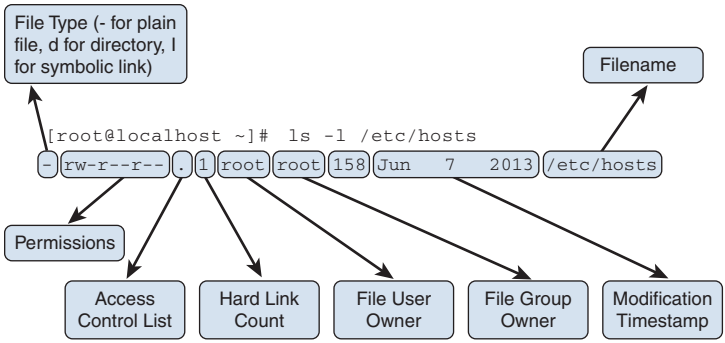

FIGURE 2.6 **The ls -l Command**

pwd

The **pwd** ("print working directory") command displays the shell's current directory, as in this example:

```
[student@localhost rc0.d]$ pwd
/etc/rc0.d
```

rm

The **rm** command is used to delete files and directories.

Example:

```
rm file.txt
```

Table 2.22 describes some important options of the **rm** command.

TABLE 2.22 **rm Command Options**

Option	Description
-i	Provides an interactive prompt before removing the file.
-r	Recursively deletes the entire directory structure.
-v	Enters verbose mode and describes actions taken when deleting files and directories.

cd

To move the shell's current directory to another directory, use the **cd** ("change directory") command. The **cd** command accepts a single argument: the

location of the desired directory. For example, to move to the **/etc** directory, you can execute the following command:

```
[student@localhost ~]$ cd /etc
[student@localhost etc]$
```

The **cd** command is "no news is good news" sort of command. If the command succeeds, no output is displayed (although the prompt changes). If the command fails, an error is displayed, as shown below:

```
[student@localhost ~]$ cd /etc
bash: cd: nodir: No such file or directory
[student@localhost ~]$
```

. (Current Directory)

One dot (period) represents the current directory. This isn't very useful with the **cd** command, but it is handy with other commands when you want to say "the directory I am currently in."

.. (Level Above the Current Directory)

Two dot characters represent one level above the current directory. So, if the current directory is **/etc/skel**, the command **cd ..** changes the current directory to the **/etc** directory.

~ (User's Home Directory)

The tilde character represents the user's home directory. Every user has a home directory (typically **/home/***username*) for storing their own files. The **cd ~** command returns you to your home directory.

tree

The **tree** command allows you to see a directory hierarchy, as in this example:

```
[student@localhost ~]$ tree /etc | head -20
/etc
|-- abrt
|   |-- abrt-action-save-package-data.conf
```

```
|   |-- abrt.conf
|   |-- abrt-harvest-vmcore.conf
|   |-- gpg_keys.conf
|   |-- plugins
|   |   |-- CCpp.conf
|   |   `-- python.conf
|   `-- xorg.conf
|-- acpi
|   |-- actions
|   |   `-- power.sh
|   `-- events
|       |-- powerconf
|       `-- videoconf
|-- adjtime
|-- aide.conf
|-- aliases
|-- aliases.db
```

cat

The **cat** command displays the contents of text files. Table 2.23 describes some important **cat** command options.

TABLE 2.23 **cat Command Options**

Option	Description
-A	Functions the same as **-vET**.
-e	Functions the same as **-vE**.
-E	Displays a $ character at the end of each line (to visualize trailing whitespace characters).
-n	Numbers all lines of output.
-s	Converts multiple blank lines into a single blank line.
-T	Displays ^I for each tab character (in order to show spaces instead of tabs).
-v	Displays “unprintable” characters (such as control characters).

ExamAlert

The **cat** command does not pause the display after one page of output.

touch

The **touch** command has two functions: to create an empty file and to update the modification and access timestamps of an existing file. To create a file or update an existing file's timestamps to the current time, use the following syntax:

```
touch filename
```

Table 2.24 describes some important options of the **touch** command.

TABLE 2.24 **touch Command Options**

Option	Description
-a	Modifies the access timestamp only, not the modification timestamp.
-d *DATE*	Sets the timestamp to the specified *DATE* (for example, **touch -d "2018-01-01 14:00:00"**).
-m	Modifies the modification timestamp only, not the access timestamp.
-r *file*	Uses the timestamp of *file* as a reference to set the timestamps of the specified file (for example, **touch -r /etc/hosts /etc/passwd**).

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. While of the following is a valid **sed** command?
   - ❍ **A. ls -l | sed 's~root~null~g'**
   - ❍ **B. ls -l | sed 's\root\null\g'**
   - ❍ **C. ls -l | sed 's-root-null-g'**
   - ❍ **D. ls -l | sed 's/root/null/g'**

2. Which of the following tools can be used to merge multiple files together? (Choose two.)
   - ❍ **A. zip**
   - ❍ **B. gzip**
   - ❍ **C. bzip**
   - ❍ **D. tar**

3. Which option to the **ln** command creates a hard link?
   - ❍ **A. -s**
   - ❍ **B. -h**
   - ❍ **C. -l**
   - ❍ **D.** None of these answers are correct.

4. Which option to the **ls** command displays all files, including hidden files?
   - ❍ **A. -l**
   - ❍ **B. -a**
   - ❍ **C. -d**
   - ❍ **D. -s**

Cram Quiz Answers

1. **D.** The **/** character should be used between the different values of a **sed** statement.
2. **A and D.** The **zip** and **tar** commands merge multiple files together. The **gzip** and **bzip2** commands compress files individually.
3. **D.** The **ln** command creates hard links without the need for any options. To create soft links, you must use the **-s** option.
4. **B.** Use the **-a** option to display all files, including hidden files.

CHAPTER 3

Configure and Manage Storage Using the Appropriate Tools

This chapter covers the following Linux+ XK0-005 exam objective:

- **1.3:** Given a scenario, configure and manage storage using the appropriate tools.

In this chapter you will learn how to manage storage devices. The first section covers disk partitioning, and you will learn about tools like **fdisk, parted**, and **partprobe**. Once you have learned how to create a partition, you will learn about how to place a filesystem on the partition and make the filesystem available to the operating system through a process called *mounting*.

This chapter explores two alternatives to partitioning: Logical Volume Manager and RAID. This chapter also covers the essentials of working with remote storage devices, such as NFS (Network File System) and SMB/CIFS (Server Message Block/Common Internet File System).

This chapter provides information on the following topics: disk partitioning, mounting of local and remote devices, filesystem management, monitoring of storage space and disk usage, creation and modification of volumes using Logical Volume Manager (LVM), inspection of RAID implementations, storage area network (SAN)/network-attached storage (NAS), and storage hardware.

Disk Partitioning

Partitions are used to separate a hard disk into smaller components. Each component can be treated as a different storage device, and a separate filesystem (btrfs, xfs, etx4, and so on) can be created on each partition.

There is a limit to the number of traditional PC-based partitions you can create. Originally only four partitions, referred to as *primary partitions*, were permitted. As more partitions were needed, a technique was created that makes it possible to convert one of the primary partitions

into an extended partition. Within an extended partition, you can create additional partitions called *logical partitions*.

In Figure 3.1, **/dev/sda1**, **/dev/sda2**, and **/dev/sda3** are primary partitions. The **/dev/sda4** partition is an extended partition that is used as a container for the **/dev/sda5**, **/dev/sda6**, and **/dev/sda7** logical partitions.

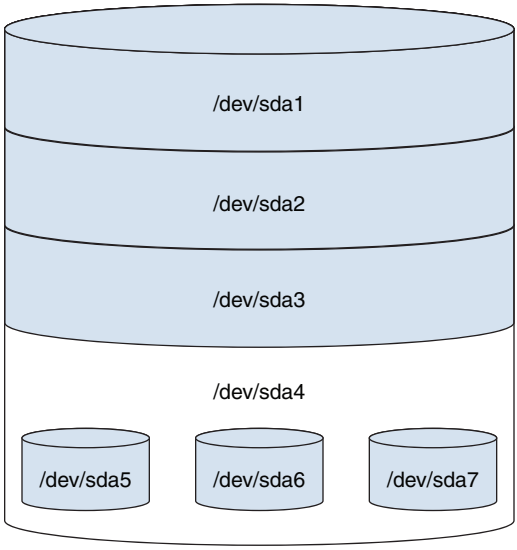

FIGURE 3.1 **Traditional Partition Structure**

On most Linux distributions that use traditional partitions, you are limited to a total of 15 partitions (though a kernel tweak can increase this number to 63).

Traditional partition tables are stored on the master boot record (MBR). A newer partition table, called the GUID partition table (GPT), does not face the same limitations or have the same layout as an MBR partition table.

Several different tools can be used to create or view partitions, including **fdisk**, **parted**, and the GUI-based tool provided by the installation program (which can vary based on the distribution).

Both **fdisk** and **parted** support command-line options, and both of them can be executed as interactive tools.

fdisk

The **fdisk** utility is an interactive tool that allows you to display and modify traditional (non-GUID) partition tables. To display a partition table, use the **-l** option (as the root user), like so:

```
# fdisk -l /dev/sda

Disk /dev/sda: 42.9 GB, 42949672960 bytes
4 heads, 32 sectors/track, 655360 cylinders, total 83886080 sectors
Units = sectors of 1 * 512 = 512 bytes
Sudo
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000c566f

      Device Boot Start End      Blocks    Id System
     /dev/sda1   * 2048 83886079 41942016  83 Linux
```

To modify the partition table of a drive, run the **fdisk** command without the **-l** option, as shown here:

```
# fdisk /dev/sda
Command (m for help):
```

There are several useful commands you can type at the **Command** prompt, including those listed in Table 3.1.

TABLE 3.1 **Partition-Related Commands**

Command	Description
d	Deletes a partition.
l	Lists partition types.
m	Prints a menu of possible commands.
n	Creates a new partition.
p	Prints the current partition table.
q	Quits without saving any changes.
t	Changes a partition table type.
w	Writes (saves) changes to the partition table on the hard drive.

parted

The **parted** utility is an interactive tool that allows you to display and modify both traditional and GUID partition tables. It can also create a filesystem on a partition.

To display a partition table, use the **-l** option and run the **parted** command as the root user, as in this example:

```
# parted -l /dev/sda
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sda: 42.9GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number Start   End    Size    Type File system Flags
 1      1049kB 42.9GB primary ext4     boot


Model: Linux device-mapper (thin) (dm)
Disk /dev/mapper/docker-8:1-264916-
f9bd50927a44b83330c036684911b54e494e4e48efbc2329262b6f0e909e3d7d:
107GB
Sector size (logical/physical): 512B/512B
Partition Table: loop

Number Start   End   Size File system Flags
 1       0.00B 107GB ext4


Model: Linux device-mapper (thin) (dm)
Disk /dev/mapper/docker-8:1-264916-
77a4c5c2f607aa6b31a37280ac39a657bfd7ece1d940e50507fb0c128c220f7a:
107GB
Sector size (logical/physical): 512B/512B
Partition Table: loop

Number Start   End   Size File system Flags
 1      0.00B  107GB ext4
```

To modify the partition table of a drive, run the **parted** command without the **-l** option, as shown here:

```
# parted /dev/sda
GNU Parted 2.3
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

Table 3.2 shows several useful commands you can type at the **(parted)** prompt.

TABLE 3.2 **Partition-Related Commands Entered at the (parted) Prompt**

Command	Description
rm	Deletes a partition.
? or **help**	Prints a menu of possible commands.
mkpart	Creates a new partition.
mkpartfs	Creates a new partition and filesystem.
print	Prints the current partition table.
quit	Quits without saving any changes.
w	Writes (saves) changes to the partition table on the hard drive.

ExamAlert

Remember that **parted** can manage GUID, but **fdisk** cannot.

partprobe

The **partprobe** command is normally needed only in situations where the partition table has changed and the system needs to be informed of the changes. The most common example is when you use the **fdisk** command to change a partition on a device that currently has mounted filesystems. The **fdisk** command attempts to inform the system of changes to the partition table by using a kernel call, which fails because of the "live" filesystem. To overcome this, just execute the **partprobe** command after exiting the **fdisk** utility.

Mounting Local and Remote Devices

Mounting is the process of making storage devices available to the Linux filesystem. This section focuses on the files and tools that are used to manage the mounting process.

systemd.mount

The **systemd.mount** configuration is used by Systemd to mount and unmount resources during the boot process. It utilizes the **/etc/fstab** file (which is described next). To learn more about Systemd, see Chapter 4, "Configure and Use the Appropriate Processes and Services."

/etc/fstab

The **/etc/fstab** file is used to specify which filesystems to mount, where to mount the filesystems, and what options to use during the mount process. This file is used during the boot process to configure filesystems to mount on bootup.

Each line of the **/etc/fstab** file describes one mount process. The following is an example of one of these lines:

```
/dev/sda1 / ext4 defaults 1 1
```

Each line is broken into six fields of data, separated by whitespace:

- The device to mount (in the preceding example, **/dev/sda1**).
- The mount point (**/**).
- The filesystem type (**ext4**).
- The mount options (**defaults**).
- Dump level (**1**). This field is related to the **dump** command and is rarely used.
- The **fsck** pass field (**1**). The value **0** means "do not run **fsck** on this filesystem during system boot," whereas a value of **1** or higher means "run **fsck** on this filesystem during system boot."

Note that a complete **/etc/fstab** file contains multiple entries as well as some comments that provide basic documentation. Here is an example:

```
#
# /etc/fstab
# Created by anaconda on Tue Jun 22 03:22:27 2021
#
# Accessible filesystems, by reference, are maintained under '/dev/
disk/'.
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more
info.
#
# After editing this file, run 'systemctl daemon-reload' to update
systemd
# units generated from this file.
#
```

```
/dev/mapper/cl-root     /               xfs     defaults        0 0
UUID=abc23ff6-7841-463b-bdac-45b3ae868cee /boot      xfs
defaults        0 0
/dev/mapper/cl-home     /home           xfs     defaults        0 0
/dev/mapper/cl-swap     none            swap    defaults        0 0
```

ExamAlert

It is a good idea to memorize the different fields of the **/etc/fstab** file because the Linux+ XK0-005 exam typically includes questions on the contents of this file.

mount

The **mount** command can display the currently mounted filesystems, as shown in this example:

```
# mount
/dev/sda1 on / type ext4 (rw)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/cgroup type tmpfs (rw)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs
 (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
none on /run/user type tmpfs
 (rw,noexec,nosuid,nodev,size=104857600, mode=0755)
none on /sys/fs/pstore type pstore (rw)
rpc_pipefs on /run/rpc_pipefs type rpc_pipefs (rw)
systemd on /sys/fs/cgroup/systemd type cgroup
 (rw,noexec,nosuid,nodev, none,name=systemd)
```

The **mount** command can also be used to manually mount a filesystem. Provide the device to mount as the first argument and the mount point (that is, mount directory) as the second argument and execute the following commands as the root user:

```
# mkdir /data
# mount /dev/sdb1 /data
```

Table 3.3 details important options for the **mount** command.

TABLE 3.3 **mount Command Options**

Option	Description
-a	Mounts all filesystems listed in the **/etc/fstab** file that have the mount option **auto**.
-o	Specifies a mount option (for example, **mount -o acl /dev/sdb1 /data**).
-t	Specifies the filesystem type to mount. This is typically not necessary because the **mount** command can determine the filesystem type by probing the partition.

Use the **umount** command to manually unmount a filesystem:

```
# mount | grep /data
/dev/sdb1 on /data type ext3 (rw)
# umount /data
```

Table 3.4 details important options for the **umount** command.

TABLE 3.4 **umount Command Options**

Option	Description
-r	Attempts to mount the filesystem as read-only if the unmount operation fails.
-f	Forces the unmount. This is typically used on NFS mounts when the NFS server is nonresponsive.

If you have just created the filesystem, it will likely be easy to remember which device file was used to access the filesystem. However, if you forget which device files are available, you can execute the **lsblk** command, as shown here:

```
# lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    252:0    0 254G 0  disk
| vda1  252:1  0 250G 0 part /
vda2    252:2  0   4G 0 part [SWAP]
```

This command was performed on a native virtual machine—hence the device names **vda**, **vda1**, and **vda2**.

You can see your label and UUIDs by using the **blkid** command:

```
# blkid
/dev/sda1: UUID="4d2b8b91-9666-49cc-a23a-1a183ccd2150" TYPE="ext4"
/dev/sda3: LABEL="mars" UUID="bab04315-389d-42bf-
  9efa-b25c2f39b7a0" TYPE="ext4"
/dev/sda4: UUID="18d6e8bc-14a0-44a0-b82b-e69b4469b0ad" TYPE="ext4"
```

Linux Unified Key Setup (LUKS)

An important technology related to Linux filesystem encryption is a specification called LUKS (Linux Unified Key Setup). As a specification, LUKS describes how filesystems are to be encrypted on Linux. It does not provide any software, and it is not an official standard (although specifications are commonly referred to as "unofficial standards").

Because it is a specification, LUKS does not force you to use any one specific software tool to encrypt a filesystem. Different tools are available, but for purposes of demonstration, this chapter shows a kernel-based implementation called DMCrypt.

DMCrypt is a kernel module that allows the kernel to understand encrypted filesystems. In addition to the DMCrypt module, you should be aware of two commands that you can use to create and mount an encrypted filesystem: **cryptsetup** and **cryptmount**. Note that you would use only one of these two commands (most likely **cryptsetup**) to configure an encrypted filesystem.

This section demonstrates how to create a new encrypted filesystem by using the **cryptsetup** command. To begin, you may need to load some kernel modules:

```
[root@onecoursesource ~]# modprobe dm-crypt
[root@onecoursesource ~]# modprobe aes
[root@onecoursesource ~]# modprobe sha256
```

Next, create a LUKS-formatted password on a new partition. Note that if you are using an existing partition, you first need to back up all data and unmount the partition. The following command overrides data on the **/dev/sda3** partition:

```
[root@onecoursesource ~]# cryptsetup --verbose --verify-passphrase
luksFormat /dev/sda3
WARNING!
========
This will overwrite data on /dev/sda3 irrevocably.
Are you sure? (Type uppercase yes): YES
Enter passphrase:
Verify passphrase:
Command successful.
```

Notice from this output of the **cryptsetup** command that you are prompted to provide a passphrase (a string of characters, such as a sentence or simple phrase). You will need to use this passphrase to decrypt the filesystem whenever you need to mount the filesystem.

Filesystem Management

Several sets of tools are used to manage filesystems and storage devices. This section covers the XFS, EXT, and Btrfs tools.

XFS Tools

XFS is a filesystem designed for high performance and for handling larger file sizes.

The **xfs_metadump** command dumps (copies) metadata from an unmounted XFS filesystem into a file to be used for debugging purposes. Table 3.5 details some important options for XFS tools.

TABLE 3.5 **XFS Tool Options**

Option	Description
-e	Stops the dump if a filesystem error is found.
-g	Shows the progress of the dump.
-w	Displays error messages if filesystem errors occur.

The **xfs_info** command is used to display the geometry of an XFS filesystem, similarly to the **dumpe2fs** command for ext2/ext3/ext4 filesystems. There are no special options for the **xfs_info** command.

ext4 Tools

The ext4 filesystem is a replacement for the ext3 filesystem. It supports larger filesystems and individual file sizes. ext4 provides better performance than ext3.

The **mkfs** command creates a filesystem on a partition. The basic syntax of the command is **mkfs -t** *fstype partition*, where *fstype* can be one of the types described in Table 3.6.

TABLE 3.6 **fstype Options**

Type	Description
ext2	Creates an ext2 filesystem (the default on most distributions).
ext3	Creates an ext3 filesystem.
ext4	Creates an ext4 filesystem.
bfs	Creates a btrfs filesystem.
vfat	Creates a VFAT (DOS) filesystem.
ntfs	Creates an NTFS (Windows) filesystem.
xfs	Creates an XFS filesystem.

Note that the **mkfs** command is a front-end utility to other commands. For example, if you run the command **mkfs -t ext4 /dev/sdb7**, the **mkfs.ext4 /dev/sdb7** command will actually be executed.

Each specific filesystem-creation utility has dozens of possible options that affect how the filesystem is created. These options are passed from the **mkfs** command to the specific filesystem-creation command that **mkfs** launches.

The **resize2fs** command is commonly used in conjunction with resizing a logical volume. Once the LV has been resized, the underlying ext3 or ext4 filesystem also must be resized.

If the plan is to make the LV larger, the **lvextend** command should be executed first, followed by the **resize2fs** command. No size value is needed for the **resize2fs** command, as it increases to the size of the LV. Here is an example:

```
lvextend -L+1G /dev/vol0/lv0
resize2fs /dev/vol0/lv0
```

If the plan is to make the LV smaller, you first have to resize the filesystem and then use the **lvreduce** command to decrease the size of the LV. If you reduced the LV first, the system would not be able to access the filesystem beyond the new LV size:

```
resize2fs /dev/vol0/lv0 2G
lvreduce -L2G /dev/vol0/lv0
```

The **fsck** utility is designed to find filesystem problems on unmounted filesystems. For example, you could run this command as the root user:

```
# fsck /dev/sdb1
fsck from util-linux 2.20.1
e2fsck 1.42.9 (4-Feb-2014)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/sdb1: 11/12544 files (0.0% non-contiguous), 6498/50176 blocks
```

This utility is fairly straightforward. It calls the correct filesystem check utility based on a probe of the filesystem and then prompts the user when errors are found. To fix an error, answer **y** or **yes** to each of the prompts. (Because **yes** is almost always the appropriate answer, the **fsck** utility supports a **-y** option, which automatically answers **yes** to each prompt.)

The **fsck** command executes filesystem-specific utilities. In the case of ext2/ext3/ext4 filesystems, the **fsck** command executes the **e2fsck** utility. See the "**fsck**" section, earlier in this chapter, for details regarding the **fsck** command.

The **tune2fs** command is used to display or modify specific metadata for an ext2/ext3/ext4 filesystem. For example, by default, 5% of an ext2/ext3/ext4 filesystem is reserved for the system administrator. You can run the following command as the root user:

```
# tune2fs -l /dev/sdb1 | grep block
Inode count:           12544
Block count:           50176
Reserved block count: 2508
Mount count:             0
Maximum mount count:  -1
```

Note that the reserved block count (2508) is 5% of the block count (50176). Use the following command to change this to a different percentage:

```
# tune2fs -m 20 /dev/sdb1
tune2fs 1.42.9 (4-Feb-2014)
Setting reserved blocks percentage to 20% (10035 blocks)
```

Table 3.7 details important options for the **tune2fs** command.

TABLE 3.7 **tune2fs Options**

Option	Description
-J	Modifies journal options.
-o	Modifies mount options.
-L	Modifies the filesystem label.
-l	Lists filesystem metadata.
-m	Modifies the percentage of the filesystem reserved for the root user.

To change the label of a filesystem, use the **e2label** command:

```
# e2label /dev/sda3 pluto
# blkid
/dev/sda1: UUID="4d2b8b91-9666-49cc-a23a-1a183ccd2150" TYPE="ext4"
/dev/sda3: LABEL="pluto" UUID="bab04315-389d-42bf-
 9efa-b25c2f39b7a0" TYPE="ext4"
/dev/sda4: UUID="18d6e8bc-14a0-44a0-b82b-e69b4469b0ad" TYPE="ext4"
```

Btrfs Tools

Btrfs, known as "butter FS," is a general-purpose Linux filesystem that uses a method called "B-trees" to manage the filesystem structure. To manage a Btrfs filesystem, use the **btrfs** utility.

A large number of subcommands are available with the **btrfs** utility. Table 3.8 details some of the important subcommands for the **btrfs** utility.

TABLE 3.8 **btrfs Utility Subcommands**

Subcommand	Description
filesystem	Manages filesystem tasks and displays filesystem information.
subvolume	Manages subvolumes and displays subvolume information.
rescue	Performs filesystem rescue (fix) operations.

You can use a subcommand as an argument to the **btrfs** utility. For example, the following would display filesystem information on the Btrfs filesystem on all devices:

```
btrfs filesystem show
```

Monitoring Storage Space and Disk Usage

Gathering storage space information can be useful when determining if there is enough space available to install a software package or database. It can also be helpful in determining which directories contain files that are using a large amount of space. The commands described in this section provide you with information needed in monitoring storage space and disk usage.

df

The **df** command displays usage of partitions and logical devices:

```
# df
Filesystem   1K-blocks   Used     Available Use% Mounted on
udev          2019204    12       2019192     1% /dev
tmpfs          404832    412      404420      1% /run
/dev/sda1    41251136    6992272  32522952   18% /
none         4           0         4          0% /sys/fs/cgroup
none         5120        0        5120        0% /run/lock
none         2024144     0        2024144     0% /run/shm
none         102400      0         102400     0% /run/user
```

Table 3.9 details important options for the **df** command.

TABLE 3.9 **df Command Options**

Option	Description
-h	Displays values in human-readable size.
-i	Displays inode information.

du

The **du** command provides an estimated amount of disk space usage in a directory structure. For example, the following command displays the amount of space used in the **/usr/lib** directory:

```
# du -sh /usr/lib
791M /usr/lib
```

Table 3.10 details important options for the **du** command.

TABLE 3.10 **du Command Options**

Option	Description
-h	Displays values in a human-readable size. (Instead of always displaying in bytes, it displays in more understandable values, such as megabytes or kilobytes, depending on the overall size of the file.)
-s	Displays a summary rather than the size of each subdirectory.

Creating and Modifying Volumes Using Logical Volume Manager (LVM)

LVM is designed to address a few issues with regular partitions, including the following:

- Regular partitions are not resizable. LVM provides the means to change the size of partition-like structures called *logical volumes*.
- The size of a regular partition cannot exceed the overall size of the hard disk on which the partition is placed. With LVM, several physical devices can be merged together to create a much larger logical volume.
- Active filesystems pose a challenge when you're backing up data because changes to the filesystem during the backup process could result in corrupt backups. LVM provides a feature called a "snapshot" that makes it easy to correctly back up a live filesystem.

LVM consists of one or more physical devices merged into a single container of space that can be used to create partition-like devices. The physical devices can be entire hard disks, partitions on a hard disk, removable media devices (USB drives), software RAID devices, or any other storage devices.

ExamAlert

LVM can be a challenging topic. For the Linux+ XK0-005 exam, make sure you understand the differences between physical volumes (PVs), volume groups (VGs), and logical volumes (LVs).

pvs

You can display PVs by using the **pvs** command, as demonstrated in the following example:

```
# pvs
  PV         VG    Fmt  Attr PSize  PFree
  /dev/sdb1  my_vg lvm2 a-   19.24G 18.14G
  /dev/sdc1  my_vg lvm2 a-   19.24G 18.06G
  /dev/sdd1  my_vg lvm2 a-   19.24G 18.14G
```

See the "**pvcreate**" section, later in this chapter, for more information about PVs.

vgs

You can display VGs by using the **vgs** command, as demonstrated in the following example:

```
# vgs
  VG            #PV #LV #SN Attr   VSize  VFree
  Vol00         3   2   0   wz--n- 33.22G 0
  Vol01         2   1   0   wz--nc 47.00G 8.00M
```

See the "**vgcreate**" section, later in this chapter, to learn more about VGs.

lvs

You can display LVs by using the **lvs** command, as demonstrated in the following example:

```
# lvs
  LV      VG        Attr     LSize  Pool Origin Data%  Move Log Copy%
Convert
  lv_root Vol00    -wi-ao--  12.22g
  lv_swap Vol00    -wi-ao--  3.12g
```

See the "**lvcreate**" section, later in this chapter, to learn more about LVs.

lvchange

The **lvchange** command allows you to change the attributes of an LV. One common use of this command is to change the state of an LV to active:

```
lvchange -ay /dev/vol0/lv0
```

The same command can be used to deactivate an LV:

```
lvchange -an /dev/vol0/lv0
```

lvcreate

The space within PVs is broken into small chunks called *extents*. Each extent is 4MB by default (but can be modified when creating the VG by using the **-s** option to the **vgcreate** command). To create an LV, execute the **lvcreate** command and either specify how many extents to assign to the LV or provide a size (which will be rounded up to an extent size), as shown here:

```
lvcreate -n lv0 -L 400MB vol0
```

The result of this command will be a device file named **/dev/vol0/lv0** that will have 400MB of raw space available. See Figure 3.2 for a visual example.

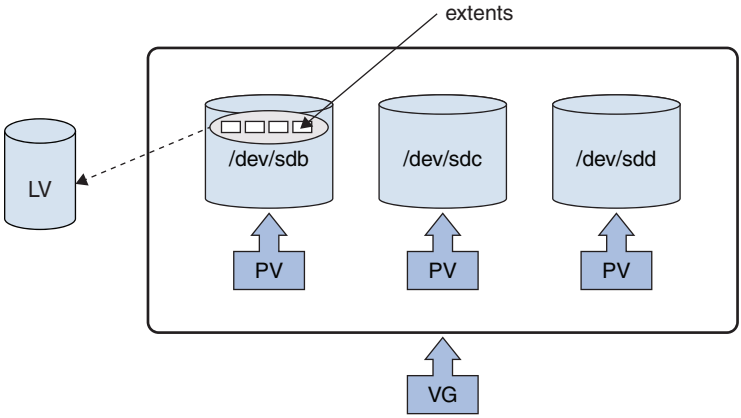

FIGURE 3.2 **Logical Volumes**

vgcreate

After creating PVs with the **pvcreate** command (see the "**pvcreate**" section, later in this chapter), place these PVs into a VG by executing the following command:

```
# vgcreate vol0 /dev/sdb /dev/sdc /dev/sdd
```

Consider a VG to be a collection of storage devices that you want to use to create partition-like structures called logical volumes (LVs). So, if **/dev/sdb** is a 60GB hard drive, **/dev/sdc** is a 30GB hard drive, and **/dev/sdd** is a 20GB hard drive, then the VG created by the previous command has 110GB of space available to create the LVs. You could create a single LV using all 110GB or many smaller LVs. See Figure 3.3 for a visual demonstration of volume groups.

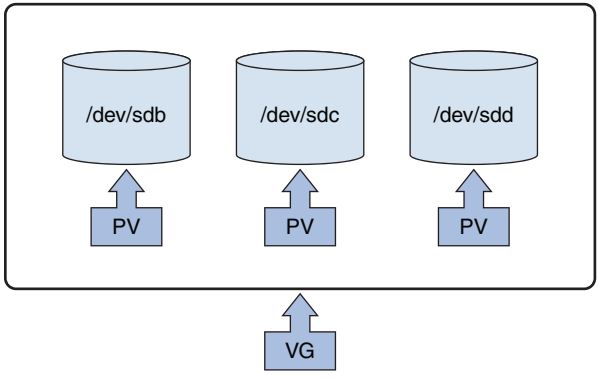

FIGURE 3.3 **Volume Groups**

lvresize

The **lvresize** command allows you to change the size of an LV. For example, the following command adds 10GB to the **/dev/vol0/lv0** LV, assuming that there is enough free space in the VG:

```
lvresize -L +10G dev/vol0/lv0
```

pvcreate

The first step in creating an LVM is to convert existing physical devices into PVs. This is accomplished by executing the **pvcreate** command. For example, if you have three hard drives, as shown in Figure 3.4, and you want to make them all PVs, you can execute the following command:

```
pvcreate /dev/sdb /dev/sdc /dev/sdd
```

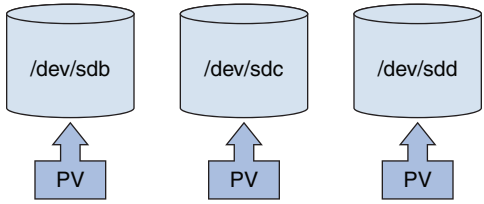

FIGURE 3.4 **Physical Volumes**

vgextend

When you want to add a new PV to a VG, you can use the **vgextend** command. For example, the following command adds the **/dev/sde** PV to the **vol0** VG:

```
vgextend vol0 /dev/sde
```

> **Note**
>
> The **/dev/sde** device must first be configured as a PV using the **pvcreate** command. See the **"pvcreate"** section, later in this chapter, for more information.

Inspecting RAID Implementations

A RAID device is used to provide redundancy. Two or more physical devices can be combined to create a single device that stores data in a way that mitigates data loss in the event that one of the physical storage devices fails.

There are several different types of RAID devices, called "RAID levels." While you can create many different types of software RAID levels, only a few of the levels really make sense for software RAID devices. When you configure a software RAID device, you should consider the RAID levels described in Table 3.11.

TABLE 3.11 **RAID Levels**

Level	Description
RAID 0	RAID 0 involves writing to multiple drives as if they were a single device. The writes are performed using "striping," in which some data is written to the first drive, then some data is written to the second drive, and so on. RAID 0 combines multiple smaller hard disk drives into a single storage space. So, if you have three 20GB hard disks, you could merge them together into a single 60GB storage device. Software RAID 0 is extremely rare for a couple of reasons. One reason is that this RAID level provides no redundancy (which makes it strange that it is considered RAID). Another reason is that LVM (Logical Volume Manager) also makes it possible to merge multiple devices together, and LVM has several advantages over software RAID 0 that make it the better choice. (LVM is discussed earlier in this chapter.)
RAID 1	With RAID 1, also called "mirroring," two or more disk drives appear to be a single storage device. Data that is written to one disk drive is also written to all of the others. If one drive fails, the data is still available on the other drives. Software RAID 1 is very popular because hard drives are fairly cheap, and the redundancy provided by RAID 1 limits data loss.
RAID 4	Implementing RAID 4 requires at least three drives. All but one drive is used to store filesystem data, and the last drive is used to store parity data—that is, a value derived from the data stored on the other devices in the RAID. Suppose there are three devices in the RAID; two of them (called devices A and B in this example) store regular filesystem data, and one (called device C) stores the parity data. If device A fails, then the data that it stored could be rebuilt using a comparison of the data in device B and the parity data in device C. How does this work? Consider the following formula: 1 + 1 = 2 (or, perhaps, A + B = C). If someone removed one of the values in that formula (for example, ___ + 1 = 2), you could recover that information by comparing the other two values. This is the basic idea of parity data.
RAID 5	RAID 5 is very similar to RAID 4. The difference is that RAID 5 doesn't use a single parity disk but rather spreads the parity data over multiple disks in a "round robin" approach.

Level	Description
RAID 10	Also called RAID 1+0, this RAID level combines the advantages of both RAID 1 and RAID 0. First, two or more sets of two devices are placed into multiple RAID 1 devices. This provides redundancy. Then they are merged together into a RAID 0 device to create a much larger storage container. In terms of software RAID, this RAID level is fairly rare, mostly because of the advantages of LVM over RAID 0. A much more common scenario would be a RAID 0 plus LVM combination. For hardware RAID, this level is not uncommon.

ExamAlert

Know these RAID levels for the Linux+ XK0-005 exam! The exam is likely to ask you to specify which RAID level you should use in a given scenario.

mdadm

To create a software RAID device, execute a command like the following:

```
# mdadm -C /dev/md0 -l 1 -n 2 /dev/sdb /dev/sdc
mdadm: array /dev/md0 started.
```

This command uses the following options:

- **-C:** Specifies the device name for the RAID device.
- **-l:** Specifies the RAID level.
- **-n:** Specifies the number of physical storage devices in the RAID array.

/proc/mdstat

After you create a RAID device, you can see information about the device by viewing the contents of the **/proc/mdstat** file, as in this example:

```
# more /proc/mdstat
Personalities : [raid1]
md0 : active raid1 sdc[1] sdb[0]
      987840 blocks [2/2] [UU]
unused devices: <none>
```

The **mdadm --detail** command can also be useful for displaying information about a software RAID device.

Storage Area Network (SAN)/ Network-Attached Storage (NAS)

This section provides information about SAN and NAS devices.

multipathd

Some storage devices are available only through the network. This creates a point of failure: the network itself. If you lose network access to a remote storage device, perhaps because a router went down or a new firewall rule was implemented, applications on your system might fail to function properly.

Multipathing involves creating different network paths to a remote storage device. This requires additional network setup, including configuring different routes to the network storage device. The multipath daemon (**multipathd**) is the process that manages these network paths.

> **ExamAlert**
>
> Details regarding configuring multipathing are beyond the scope of the Linux+ XK0-005 exam and are not covered in this book. However, multipathing is included as an exam objective, so be prepared to answer questions regarding the purpose of multipathing.

Network Filesystems

As with most other operating systems, Linux allows you to access filesystems that are shared across the network. This section describes two of the most common network filesystems for Linux: NFS and SMB.

Network File System (NFS)

Network File System (NFS) is a Distributed File System (DFS) protocol that has been in use for more than 40 years. NFS was originally created by Sun Microsystems in 1984 to provide an easy way for administrators to share files and directories from one Unix system to another.

Since its inception, NFS has been ported to several different operating systems, including Linux and Microsoft Windows. While it might not be as popular as SAMBA, there are many organizations that still use NFS to share files.

The primary configuration for the NFS server is the **/etc/exports** file. This is the file you use to specify what directories you want to share to the NFS clients. The syntax of this file is as follows:

```
Directory    hostname(options)
```

Directory should be replaced with the name of the directory that you want to share (for example, **/usr/share/doc**), and *hostname* should be a client hostname that can be resolved into an IP address. The *options* value is used to specify how the resource should be shared.

For example, the following entry in the **/etc/exports** file would share the **/usr/share/doc** directory to the NFS client **jupiter** as read/write and to the NFS client mars as read-only:

```
/usr/share/doc jupiter(rw) mars(ro)
```

Note that there is a space between **jupiter** and **mars**, but there is no space between each hostname and its corresponding option. A common mistake of novice administrators is to provide an entry like the following:

```
/usr/share/doc jupiter (rw)
```

This line would share the **/usr/share/doc** directory to the **jupiter** host with the default options, and all other hosts would have read/write access to this share.

When specifying a hostname in the **/etc/exports** file, the following methods are permitted:

- **hostname:** A hostname that can be resolved to an IP address.
- **netgroup:** An NIS netgroup using the designation @*groupname*.
- **domain:** A domain name using wildcards. For example, ***.onecoursesource.com** would include any machine in the onecoursesource.com domain.
- **Network:** A network defined by IP addresses using either VLSM (variable-length subnet masking) or CIDR (classless interdomain routing). Examples include 192.168.1.0/255.255.255.0 and 192.168.1.0/24.

There are many different NFS sharing options, including

- **rw:** Shares as read/write. Keep in mind that normal Linux permissions still apply. Important: This is a default option.
- **ro:** Shares as read-only.

- **sync:** Makes file data changes to disk immediately. This has an impact on performance but is less likely to result in data loss. On some distributions, this is the default.
- **async:** The opposite of sync. Initially makes file data changes to memory. This speeds up performance but is more likely to result in data loss. On some distributions, this is the default.
- **root_squash:** Maps the root user and group account from the NFS client to the anonymous accounts, typically either the nobody account or the nfsnobody account. Important: This is a default option.
- **no_root_squash:** Maps the root user and group account from the NFS client to the local root and group accounts.

Server Message Block(SMB)/Common Internet File System (CIFS)

One of the ways to share files between different systems is by using a protocol called SMB (Server Message Block). This protocol was invented in the mid-1980s by IBM to make it possible to share directories between hosts on a local area network (LAN). Distributed File System (DFS) is used to share files and directories across a network. In addition to SMB, Network File System (NFS) is a popular DFS for Linux. (NFS is covered earlier in this chapter.)

You may often hear the acronym CIFS (Common Internet File System) used in conjunction with SMB. CIFS is an SMB-based protocol that is popular on Microsoft Windows systems. Typically, the two abbreviations are used interchangeably (or together, as SMB/CIFS), but there are subtle differences between these protocols. This book uses the term SMB from this point on.

You can also share printers using SMB as well as share files between different operating system types. In fact, one common SMB task is to share printers between Linux and Microsoft Windows systems.

The Linux-based software that allows SMB sharing is called SAMBA. The configuration file for SAMBA is the **/etc/SAMBA/smb.conf** file.

To give you an idea of what a typical **smb.conf** file looks like, examine the following output, which demonstrates a typical default **smb.conf** file with all comment and blank lines removed:

```
[root@onecoursesource ~]# grep -v "#" /etc/SAMBA/smb.conf | grep -v
";" | grep -v "^$"
[global]
```

```
        workgroup = MYGROUP
        server string = SAMBA Server Version %v
        security = user
        passdb backend = tdbsam
        load printers = yes
        cups options = raw
[homes]
        comment = Home Directories
        browseable = no
        writable = yes
[printers]
        comment = All Printers
        path = /var/spool/SAMBA
        browseable = no
        guest ok = no
        writable = no
        printable = yes
```

Table 3.12 describes the common options for the SAMBA configuration file.

TABLE 3.12 **SAMBA Configuration File Options**

Option	Description
workgroup	This is the NetBIOS (Network Basic Input/Output System) workgroup or NetBIOS domain name. It enables you to group together a set of machines, which may be important if you want to communicate with Microsoft Windows systems via SMB.
server string	This is a description of the server and is useful when a remote system is attempting to connect to the server to determine what services the server provides. The value **%v** is replaced with SAMBA's version number. The value **%h** can be used to symbolize the server's hostname.
security	This determines what type of user authentication method is used. The value **user** means SAMBA user accounts will be used. If you have a DC (domain controller) on a Microsoft Windows system, you specify the domain. To authenticate via Active Directory, you specify **ads**.
passdb backend	This specifies how the SAMBA account data is stored. It is not typically something you change unless you are an expert.

Option	Description
load printers	This option, if set to **yes**, tells SAMBA to share all CUPS (Common UNIX Printing System, a common printing protocol in Linux and Unix) printers by default. While this can be handy, you don't always want to share all CUPS printers on the SAMBA server. Individual printer shares can be handled in separate sections. Note that for SAMBA to be able to load all CUPS printers automatically, the **[printers]** section needs to be properly configured.
cups options	These options regarding CUPS are modified only by experts.

Storage Hardware

Linux provides several commands that you can use to display information about storage hardware. This section covers the most commonly used of these commands.

lsscsi

A Small Computer System Interface (SCSI) device is a storage device (typically a hard drive, though it can also be a tape drive). These devices are fairly rare on Intel-based PCs, which is what most Linux systems are installed on. If your system is a SCSI device, you can list information about it by using the **lsscsi** command.

lsblk

If you have just created the filesystem, it will likely be easy to remember which device file was used to access the filesystem. However, if you forget which device files are available, you can execute the **lsblk** command. The following is an example of this command performed on a native virtual machine—hence the device names **vda**, **vda1**, and **vda2**:

```
# lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    252:0    0 254G 0  disk
| vda1  252:1  0 250G 0 part /
vda2    252:2  0   4G 0 part [SWAP]
```

blkid

You can see your label and UUIDs by using the **blkid** command:

```
# blkid
/dev/sda1: UUID="4d2b8b91-9666-49cc-a23a-1a183ccd2150" TYPE="ext4"
/dev/sda3: LABEL="mars" UUID="bab04315-389d-42bf-
  9efa-b25c2f39b7a0" TYPE="ext4"
/dev/sda4: UUID="18d6e8bc-14a0-44a0-b82b-e69b4469b0ad" TYPE="ext4"
```

fcstat

You can display information about storage devices that are attached to Fibre Channel by using the **fcstat** command. Fibre Channel storage devices are fairly rare for Linux devices, but you should know the common options for this command, as listed in Table 3.13.

TABLE 3.13 **fcstat Command Options**

Option	Description
link_stats	Displays link information in the event of connection errors.
device_map	Displays information about attached devices.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which utility can you use to create partitions on a GUID partition table?
   - ❍ **A. partprobe**
   - ❍ **B. parted**
   - ❍ **C. fdisk**
   - ❍ **D. mkfs**

2. Which field in the following example denotes the mount point?
   ```
   /dev/sda1 / ext4 defaults 1 1
   ```
   - ❍ **A. /dev/sda1**
   - ❍ **B. /**
   - ❍ **C. ext4**
   - ❍ **D. defaults**

3. Which command allows you to add a new PV to a VG?
   - ❍ **A. pvadd**
   - ❍ **B. pvextend**
   - ❍ **C. vgadd**
   - ❍ **D. vgextend**

4. Which command can be used to view the labels and UUIDs of devices?
   - ❍ **A. blkid**
   - ❍ **B. lsblk**
   - ❍ **C. lsscsi**
   - ❍ **D. fcstat**

Cram Quiz Answers

1. **B. parted** allows you to modify both MBR and GUID partition tables. The **fdisk** utility only allows you to modify MBR partition tables. The other answers are not partition tools.
2. **B.** The second field of the **/etc/fstab** file contains the mount point.
3. **D.** The **vgextend** command allows you to add a new PV to an existing VG, as in this example: **vgextend vol0 /dev/sde**. The rest of the answers are not valid commands.
4. **A.** The **blkid** command is used to display UUIDs and labels on storage devices.

CHAPTER 4

Configure and Use the Appropriate Processes and Services

This chapter covers the following Linux+ XK0-005 exam objective:

- **1.4:** Given a scenario, configure and use the appropriate processes and services.

On Linux systems, a service is a feature of the operating system or software that acts like a server. Administrators need to know how to manage these services by using the **systemctl** command, which is covered in this chapter.

Administrators often need to use the **crontab** and **at** utilities, which make it possible to schedule processes (or programs) in the future. Speaking of processes, it is also important to know how to list and control processes. These topics are also covered in this chapter.

This chapter provides information on the following topics: system services, scheduling of services, and process management.

System Services

Systemd is a feature of Linux that is used to control which services are started during the boot process. Systemd uses "targets," and each target has specific services that start. Figure 4.1 shows an example of a typical Systemd boot sequence.

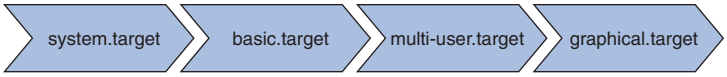

FIGURE 4.1 **Overview of the Systemd Boot Process**

Targets are defined in the **/usr/lib/systemd/system** directory. Consider the following example of a target file:

```
# cat /usr/lib/systemd/system/graphical.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License
 as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target
 display-manager.service
AllowIsolate=yes
```

The default target is defined by a symbolic link from **/etc/systemd/system/default.target** to the target in the **/usr/lib/systemd/system** directory, as in this example:

```
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 36 Jun 11 20:47
 /etc/systemd/system/default.target ->
 /lib/systemd/system/graphical.target
```

Use the following command to display the default target:

```
# systemctl get-default
multi-user.target
```

Use the **systemctl list-unit-files --type=target** command to list the available targets. This command provides a large amount of output, and the following example uses the **head** command to limit the output:

```
# systemctl list-unit-files --type=target | head
UNIT FILE STATE
basic.target              static
bluetooth.target          static
cryptsetup-pre.target     static
cryptsetup.target         static
ctrl-alt-del.target       disabled
cvs.target                static
default.target            enabled
emergency.target          static
final.target              static
```

Use the following command to set the default target:

```
# systemctl set-default graphical-user.target
rm '/etc/systemd/system/default.target'
ln -s '/usr/lib/systemd/system/graphical-
 user.target' '/etc/systemd/system/default.target'
```

systemctl

The **systemctl** command is used to administer a Systemd-based distribution. For example, to change to another target, execute the following command:

```
systemctl isolate multi-user.target
```

stop

The **stop** option is used with the **systemctl** command to stop a service that is currently running.

Syntax:

```
systemctl stop service
```

You can determine whether a service is currently running by using the command as follows:

```
# systemctl active cups
enabled
```

start

The **start** option is used with the **systemctl** command to start a service that is not currently running.

Syntax:

```
systemctl start service
```

You can determine whether a service is currently running by using the command as follows:

```
# systemctl active cups
enabled
```

restart

The **restart** option is used with the **systemctl** command to restart a service that is currently running.

Syntax:

```
systemctl restart process_name
```

You can determine whether a service is currently running by using the command as follows:

```
# systemctl active cups
enabled
```

status

The **status** option is used with the **systemctl** command to display the current status of a service.

Syntax:

```
systemctl status process_name
```

Here is an example of displaying the status of the CUPS service:

```
# sysctl status cups
 cups.service - CUPS Scheduler
 Loaded: loaded (/lib/systemd/system/cups.service; enabled;
 vendor preset: enabled)
 Active: active (running) since Mon 2019-01-07 00:10:18 PST;20h ago
 Docs: man:cupsd(8)
 Main PID: 4195 (cupsd)
 Tasks: 1 (limit: 4780)
 CGroup: /system.slice/cups.service
          4195 /usr/sbin/cupsd -l

Jan 07 00:10:18 student-VirtualBox systemd[1]: Started CUPS Scheduler.
```

enable

The **enable** option is used with the **systemctl** command to start a service at boot time.

Syntax:

```
systemctl enable service
```

You can determine if a service is currently enabled or disabled by using the command as follows:

```
# systemctl is-enabled cups
enabled
```

disable

The **disable** option is used with the **systemctl** command to change a service this is currently started at boot time so that it won't start automatically then.

Syntax:

```
systemctl disable service
```

You can determine if a service is currently enabled or disabled by using the command as follows:

```
# systemctl is-enabled cups
enabled
```

mask

To mask a service is to make it completely impossible to start or enable. This is commonly done when there is a conflicting service on a system that, for some reason, can't or shouldn't be removed from the system.

Syntax to mask a service:

```
systemctl mask service
```

> **ExamAlert**
>
> Understand the difference between masking a service and disabling a service before you take the Linux+ XK0-005 exam.

Scheduling Services

Sometimes you need to be able to execute commands in the future. The **cron** and **at** utilities described in this section allow you to schedule jobs (that is, commands) to be executed in the future.

cron

The **cron** service allows you to schedule processes to run at specific times. This service makes use of the **crond** daemon, which checks every minute to see what processes should be executed. This daemon checks both **crontab** and **at** jobs to determine what commands to execute and when to execute them. See the "**crontab**" and "**at**" sections, later in this chapter, for further details.

crontab

The **crontab** command allows you to view or modify your **crontab** file. This file allows you to schedule a command to be executed regularly, such as once an hour or twice a month.

Table 4.1 lists some important options for the **crontab** command.

TABLE 4.1 **crontab Command Options**

Option	Description
-e	Edits the **crontab** file.
-l	Lists the **crontab** file.
-r	Removes all entries from the **crontab** file.

Each line of the **crontab** file is broken into fields, separated by one or more space characters. Table 4.2 describes these fields.

TABLE 4.2 **crontab File Fields**

Field	Description
First field: **Minute**	The minute when the command should execute. Values can be 0–59. You can use a single value, a list of values (such as 0,15,30,45), or a range of values (such as 1–15). You can use the * character to indicate "all possible values."
Second field: **Hour**	The hour when the command should execute. Values can be 0–23. You can use a single value, a list of values (such as 0,6,12,18), or a range of values (such as 8–16). You can use the * character to indicate "all possible values."
Third field: **Day of the Month**	The day of the month the command should execute. Values can be 1–31. You can use a single value, a list of values (such as 1,15), or a range of values (such as 1–10). You can use the * character to indicate "not specified" unless the fifth field is also an * character, in which case the * character means "all possible values."
Fourth field: **Month**	The month that the command should execute. Values can be 1–12. You can use a single value, a list of values (such as 6,12), or a range of values (such as 1–3). You can use the * character to indicate "all possible values."
Fifth field: **Day of the Week**	The day of the week the command should execute. Values can be 0–7 (where 0=Sunday, 1=Monday,…6=Saturday, 7=Sunday). You can use a single value, a list of values (such as 1,3,5), or a range of values (such as 1–5). You can use the * character to indicate "not specified" unless the fifth field is also an * character, in which case the * character means "all possible values."
Sixth field: **Command Name**	The name of the command to execute.

For example, the following **crontab** entry executes the **/home/bob/rpt.pl** script every weekday (Monday–Friday), every month, starting at 8:00 in the morning and every half hour until 16:30 in the afternoon (4:30 p.m.):

```
0,30 8-16 * 1-12 1-5 /home/bob/rpt.pl
```

As the administrator, you can use configuration files to determine whether a user can use the **crontab** command. The **/etc/cron.deny** and **/etc/cron.allow** files are used to control access to the **crontab** command. The format of each of these files is one username per line. Here's an example:

```
[root@OCS ~]$ cat /etc/cron.deny
alias
backup
bin
daemon
ftp
games
gnats
guest
irc
lp
mail
man
nobody
operator
proxy
sync
sys
www-data
```

Table 4.3 describes how the **/etc/cron.deny** and **/etc/cron.allow** files work.

TABLE 4.3 **How the /etc/cron.deny and /etc/cron.allow Files Work**

Situation	Description
Only the **/etc/cron.deny** file exists.	All users listed in this file are denied access to the **crontab** command, whereas all other users can execute the **crontab** command successfully. Use this file when you want to deny access to a few users but allow access to most users.
Only the **/etc/cron.allow** file exists.	All users listed in this file are allowed access to the **crontab** command, whereas all other users cannot execute the **crontab** command successfully. Use this file when you want to allow access to a few users but deny access to most users.
Neither file exists.	On most Linux distributions, this means that only the root user can use the **crontab** command. However, on some platforms, this results in all users being allowed to use the **crontab** command.
Both files exist.	Only the **/etc/cron.allow** file is consulted, and the **/etc/cron.deny** file is completely ignored.

The **/etc/crontab** file acts as the system **crontab**. The system administrator edits this file to enable the execution of system-critical processes at specific intervals. The following is a sample **/etc/crontab** file:

```
[root@OCS ~]$ cat /etc/crontab
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin


#  m h dom mon dow user command
17 *      * * * root cd / && run-parts /etc/cron.hourly
```

Each configuration line describes a process to execute, when to execute it, and what username to execute the process as. Each line is broken into fields, separated by one or more space characters. Table 4.4 describes these fields.

TABLE 4.4 **Fields of a crontab Entry**

Field	Description
First field: **Minute**	The minute that the command should execute. Values can be 0–59. You can use a single value, a list of values (such as 0,15,30,45), or a range of values (such as 1–15). You can use the * character to indicate “all possible values.”
Second field: **Hour**	The hour that the command should execute. Values can be 0–23. You can use a single value, a list of values (such as 0,6,12,18), or a range of values (such as 8–16). You can use the * character to indicate “all possible values.”

Field	Description
Third field: **Day of the Month**	The day of the month that the command should execute. Values can be 1–31. You can use a single value, a list of values (such as 1,15), or a range of values (such as 1–10). You can use the * character to indicate "not specified" unless the fifth field is also an * character, in which case the * character means "all possible values."
Fourth field: **Month**	The month that the command should execute. Values can be 1–12. You can use a single value, a list of values (such as 6,12), or a range of values (such as 1–3). You can use the * character to indicate "all possible values."
Fifth field: **Day of the Week**	The day of the week that the command should execute. Values can be 0–7 (where 0=Sunday, 1=Monday,…6=Saturday, 7=Sunday). You can use a single value, a list of values (such as 1,3,5), or a range of values (such as 1–5). You can use the * character to indicate "not specified" unless the fifth field is also an * character, in which case the * character means "all possible values."
Sixth field: **Username**	The name of the user that the command should run as.
Seventh field: **Command Name**	The name of the command to execute.

ExamAlert

You will be expected to know the different fields of a **crontab** file for the Linux+ XK0-005 **exam.**

at

The **at** command is used to schedule one or more commands to be executed at one specific time in the future.

Syntax:

```
at time
```

where *time* indicates when you want to execute the command. For example, the following command allows you to schedule a command to run at 5 p.m. tomorrow:

```
at 5pm tomorrow
at>
```

When you see the **at>** prompt, you can enter a command to execute at the specified time. To execute multiple commands, press the Enter key to get another **at>** prompt. When this is complete, hold down the Ctrl key and press the d key. This results in an **<EOT>** message and creates the **at** job. Here's an example:

```
[root@OCS ~]$ at 5pm tomorrow
at> /home/bob/rpt.pl
at> echo "report complete" | mail bob
at> <EOT>
job 1 at Thu Feb 23 17:00:00 2017
```

Table 4.5 lists some important options for the **at** command.

TABLE 4.5 **at Command Options**

Option	Description
-m	Sends the user who created the **at** job an email when the job is executed.
-f *filename*	Reads commands from *filename*. This is useful when you are running the same **at** jobs on an infrequent basis.
-v	Displays the time and date when the **at** job will be executed.

The **atq** command lists the current user's **at** jobs:

```
[root@OCS ~]$ atq
1 Thu Feb 23 17:00:00 2017 a bob
```

The output includes a job number (**1** in this example), the date that the command will execute, and the user's name (**bob** in this example).

The **atq** command has no commonly used options.

To remove an **at** job before it is executed, run the **atrm** command followed by the job number to remove, as shown in this example:

```
[root@OCS ~]$ atq
1 Thu Feb 23 17:00:00 2017 a bob
[root@OCS ~]$ atrm 1
[root@OCS ~]$ atq
```

The **atrm** command has no commonly used options.

As the administrator, you can use configuration files to determine whether a user can use the command. The **/etc/at.deny** and **/etc/at.allow** files are used to control access to the **at** command.

The format of each of these files is one username per line. Here's an example:

```
[root@OCS ~]$ cat /etc/at.deny
alias
backup
bin
daemon
ftp
games
gnats
guest
irc
lp
mail
man
nobody
operator
proxy
sync
sys
www-data
```

Table 4.6 describes how the **/etc/at.deny** and **/etc/at.allow** files work.

TABLE 4.6 **How the /etc/at.deny and /etc/at.allow Files Work**

Situation	Description
Only the **/etc/at.deny** file exists.	All users listed in this file are denied access to the **at** command, and all other users can execute the **at** command successfully. Use this file when you want to deny access to a few users but allow access to most users.
Only the **/etc/at.allow** file exists.	All users listed in this file are allowed access to the **at** command, and all other users cannot execute the **at** command successfully. Use this file when you want to allow access to a few users but deny access to most users.

Situation	Description
Neither file exists.	On most Linux distributions, this means that only the root user can use the **at** command. However, on some platforms, this results in all users being allowed to use the **at** command.
Both files exist.	Only the **/etc/at.allow** file is consulted, and the **/etc/at.deny** file is completely ignored.

ExamAlert

Remember that **crontab** is for scheduling a process to run routinely, and **at** is used to schedule a process to run once.

Process Management

Process management includes listing running processes (that is, jobs or commands) and sending signals to the processes to have them modify their behavior (restart, stop, and so on). This section describes process management features.

Kill Signals

The **kill** command can be used to change the state of a process, including stopping (killing) it.

Syntax:

```
kill PID|jobnumber
```

To stop a process, first determine its process ID or job number and then provide that number as an argument to the **kill** command, as in this example:

```
[student@OCS ~]$ jobs
 [1]- Running                 sleep 999 &
 [2]+ Running sleep 777 &
 [student@OCS ~]$ kill %2
 [student@OCS ~]$ jobs
 [1]- Running                 sleep 999 &
```

```
[2]+ Terminated                 sleep 777
[student@OCS ~]$ ps -fe | grep sleep
student 17846 12540 0 14:30 pts/2 00:00:00 sleep 999
student 17853 12540 0 14:31 pts/2 00:00:00
grep --color=auto sleep
[student@OCS ~]$ kill 17846
[student@OCS ~]$ ps -fe | grep sleep
student 17856 12540 0 14:31 pts/2 00:00:00
grep --color=auto sleep
[1]+ Terminated                 sleep 999
```

Table 4.7 lists some important **kill** options.

TABLE 4.7 **kill Command Options**

Option	Description
-9	Forces a kill. Used when a process doesn't exit when a regular **kill** command is executed.
-l	Provides a list of other numeric values that can be used to send different kill signals to a process.

ExamAlert

For the Linux+ XK0-005 exam, keep in mind that kill -9 should only be used when all other attempts to stop a process have failed.

SIGTERM

A SIGTERM signal, also known as signal 15, is the default signal sent to a process when you use the **kill** command. This signal is a request for the process to stop, but the process can be programmed to ignore SIGTERM signals.

SIGKILL

A SIGKILL signal, also known as signal 9, is the signal sent to a process when you use the **kill** command with the **-9** option, as in this example:

```
kill -9 17846
```

The SIGKILL signal forces the process to stop without providing it the opportunity to shut down gracefully. If you use a SIGKILL signal on a process, you can lose data that the process had stored in memory.

SIGHUP

When a parent process is stopped, a hang-up (SIGHUP) signal is sent to all the child processes. This HUP signal is designed to stop the child processes. By default, a child process stops when sent an SIGHUP signal, but the process can be programmed to ignore SIGHUP signals. You can also have a process ignore a SIGHUP signal by executing the child process with the **nohup** command:

```
[student@OCS ~]$ nohup some_command
```

You typically use this technique when you remotely log in to a system and want to have some command continue to run even if you are disconnected. When you are disconnected, all of the programs you have running are sent HUP signals. Using the **nohup** command allows this specific process to continue running.

Listing Processes and Open Files

Several Linux commands can be used to list processes and the files that are opened by the processes. This section covers these commands.

top

The **top** command displays process information that is updated on a regular basis (by default, every 2 seconds). The first half of the output of the **top** command contains overall information, and the second half displays a select list of processes (by default, the processes that are using the CPU the most).

Figure 4.2 shows some typical output of the **top** command.

```
top - 16:09:10 up 2 days,  3:07,  2 users,  load average: 0.00, 0.07, 0.12
Tasks: 119 total,   2 running, 117 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.3 us,  1.0 sy,  0.0 ni, 97.0 id,  0.3 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem:   4048292 total,  3832140 used,   216152 free,   356468 buffers
KiB Swap:        0 total,        0 used,        0 free.  1610568 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
26159 root      20   0 2461400 1.243g  24040 S   2.7 32.2  44:59.94 java
  965 root       0 -20       0      0      0 S   0.3  0.0   0:35.58 loop0
27545 nobody    20   0   87524   3616    892 S   0.3  0.1   0:05.32 nginx
28770 root      20   0   12824    940    776 S   0.3  0.0   0:14.39 ping
    1 root      20   0   33604   2952   1476 S   0.0  0.1   0:00.98 init
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:05.72 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0.0  0.0   1:12.43 rcu_sched
    8 root      20   0       0      0      0 R   0.0  0.0   1:39.49 rcuos/0
    9 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
   10 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/0
```

FIGURE 4.2 **top Command Output**

Table 4.8 describes the output displayed in Figure 4.2.

TABLE 4.8 **top Command Output Walkthrough**

Output	Description
First line	Output derived from the **uptime** command.
Second line	A summary of processes running on the system.
Third line	CPU statistics since the last time **top** data was refreshed.
Fourth line	Physical memory statics. (Note: Type **E** while in the **top** command to change the value from kilobytes to another value.)
Fifth line	Virtual memory statics.
Remaining lines	A list of processes and associated information.

While the **top** command is running, you can use interactive commands to perform actions such as change display values, reorder the process list, and kill processes. These interactive commands are single characters. Table 4.9 describes the most important interactive commands.

TABLE 4.9 **Interactive Commands to Use with top**

Command	Description
h	Opens help. Displays a summary of interactive commands.
E	Changes the default value from kilobytes to another value; values "cycle" around back to kilobytes.
Z	Toggles color highlighting on; use lowercase **z** to toggle between color and non-color.

Command	Description
B	Toggles bold on and off.
< >	Moves the sort column to the left (<) or to the right (>).
s	Sets the update value to a different value than the default of 2 seconds.
k	Kills a process based on the process ID (PID).
q	Quits the **top** command.

The **top** command also supports several command-line options, including those listed in Table 4.10.

TABLE 4.10 **top Command-Line Options**

Option	Description
-d	Sets the time between data refresh.
-n *number*	Specifies the maximum number of data refreshes until the **top** command exits.
-u *username*	Displays only processes owned by *username*.

ps

The **ps** command is used to list processes that are running on the system. With no arguments, the command lists any child process of the current shell as well as the BASH shell, as shown here:

```
[student@OCS ~]$ ps
 PID TTY      TIME CMD
18360 pts/0   00:00:00 bash
18691 pts/0   00:00:00 ps
```

The **ps** command is unusual in that it supports older BSD options that normally don't have a hyphen (**-**) character in front of them.

Table 4.11 details some important options for the **ps** command.

TABLE 4.11 **ps Command Options**

Option	Description
-e	Displays all processes running on the system; the BSD method **ps ax** can also be used.
-f	Displays full information (that is, additional information about each process).

Option	Description
-u *username*	Displays all processes owned by *username*.
-forest	Provides a process hierarchy tree.

lsof

The **lsof** command is used to list open files. When used with no arguments, it lists all the open files for the OS, as shown here:

```
[root@OCS ~]# lsof | wc -l
25466
```

A more useful technique would be to list all files related to open network connections, as shown here:

```
[root@OCS ~]# lsof -i
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
avahi-dae 674 avahi 13u IPv4 15730 0t0
            UDP       *:mdns
avahi-dae 674 avahi 14u IPv4 15731 0t0
            UDP        *:49932
sshd 1411 root 3u IPv4 18771 0t0
            TCP       *:ssh (LISTEN)
sshd 1411 root 4u IPv6 18779 0t0
            TCP      *:ssh   (LISTEN)
master 2632 root 14u IPv4 20790 0t0
           TCP       localhost:smtp (LISTEN)
master 2632 root 15u IPv6 20791 0t0
            TCP      localhost:smtp (LISTEN)
dnsmasq 2739 nobody 3u IPv4 21518 0t0
            UDP      *:bootps
dnsmasq 2739 nobody 5u IPv4 21525 0t0
            UDP       192.168.122.1:domain
dnsmasq 2739 nobody 6u IPv4 21526 0t0
            TCP       192.168.122.1:domain (LISTEN)
cupsd 4099 root 12u IPv6 564510 0t0
            TCP       localhost:ipp (LISTEN)
```

```
cupsd 4099 root 13u IPv4 564511 0t0
            TCP        localhost:ipp (LISTEN)
dhclient 26133 root 6u IPv4 1151444 0t0
            UDP           *:bootpc
dhclient 26133 root 20u IPv4 1151433 0t0
               UDP       *:14638
dhclient 26133 root 21u IPv6 1151434 0t0
               UDP   *:47997
```

Table 4.12 describes common options for the **lsof** command.

TABLE 4.12 **lsof Command Options**

Option	Description
-i	Matches the Internet address; could also be used to display IP version (**-i4** or **-i6**) or port (**-i TCP:80**) or to display all open connections.
-u *user*	Lists files opened by *user*.
-p *pid*	Lists files opened by the process with the process ID *pid*.

htop

The **htop** command is much like the **top** command (see the "top" section, earlier in this chapter), but it has some additional features. For example, you can scroll through the output, both horizontally and vertically. The **htop** command also displays more information, such as user and kernel threads (where a *thread* is part of the execution of a process), which makes it a valuable command for software developers.

Setting Priorities

nice values are used to indicate to the CPU which process has the highest priority for access to the CPU. The values range from –20 (highest priority) to 19 (lowest priority). The default priority of any job created by a user is 0.

ExamAlert

nice values are often the subject of Linux+ XK0-005 exam questions. Remember that the lower the number, the higher the priority. Also remember that 0 is the default and that only the root user can run a process with a negative number. Finally, keep in mind that –20 is the lowest and 19 is the highest; you can expect to see confusing questions on the exam that provide numbers like –19 and 20 as possible answers.

The next section provides details on how to set a different priority when executing a command.

nice

To specify a different **nice** value than the default, execute the job via the **nice** command:

```
[student@OCS ~]$ nice -n 5 firefox
```

Note that a regular user cannot assign a negative **nice** value. These values can only be used by the root user. There are no additional useful options besides the **-n** option.

To view the **nice** value of a process, use the **-o** option with the **ps** command and include the value **nice**, as shown here:

```
[student@OCS ~] ps -o nice,pid,cmd
 NI PID CMD
 0 23865 -bash
 0 27969 ps -o nice,pid,cmd
```

renice

Use the **renice** command to change the **nice** value of an existing job. Here is an example:

```
[student@OCS ~] ps -o nice,pid,cmd
 NI PID CMD
 0 23865 -bash
 5 28235 sleep 999
 0 28261 ps -o nice,pid,cmd
[student@OCS ~] renice -n 10 -p 28235
28235 (process ID) old priority 5, new priority 10
```

```
[student@OCS ~] ps -o nice,pid,cmd
 NI PID CMD
 0 23865 -bash
 10 28235 sleep 999
 0 28261 ps -o nice,pid,cmd
```

> **Note**
>
> Regular (non-root) users can only change the priority of an existing process to a lower priority. Only the root user can alter a process priority to a higher priority.

Table 4.13 details some important options for the **renice** command.

TABLE 4.13 **renice Command Options**

Option	Description
-g *group*	Changes the priority of all files owned by *group*.
-u *user*	Changes the priority of all files owned by *user*.

Process States

Each process is assigned a state, depending on the current actions the process is taking (or if it is not taking any actions at all). This section describes the important process states.

Note that the **ps** and **top** commands can display the state a process is currently in. See the "ps" and "top" sections, earlier in this chapter, for more details.

Zombie

A *zombie process* is a process that has terminated but still has not been entirely cleared out of memory. Each process is started by another process, creating a "parent/child" relationship. When a child process ends, the parent process is responsible for telling the system that all details about the child process should be removed from memory.

In some rare cases, a child process may end without the parent being aware. This results in a zombie process. Zombie processes are fairly rare on modern Linux systems and typically indicate a bug that needs to be fixed.

Sleeping

A process that is in an uninterruptible sleep state is a process that is performing certain system calls that prevent it from being interrupted (that is, killed). Uninterruptible sleep state is fairly rarely seen on most modern Linux systems because these system calls are executed very quickly. If a process stays in uninterruptible sleep for a noticeable period of time, it is likely the result of a bug in the software.

A process that is in an interruptible sleep state is one that is performing some sort of I/O (input/output) operation, such as accessing a hard disk. This is a fairly common state, as I/O operations may take some time.

However, a process that is in interruptible sleep for a long period of time, especially if it is impacting the performance of the system, can indicate a problem. Either the device the process is attempting to access has an error (such as a bad data block on a hard disk) or the program has a bug.

Running

A running process is one that currently has operations taking place on the CPU or has operations on the CPU queue.

Stopped

A stopped process is no longer executing, but it might not have been cleared completely from memory.

Job Control

Job control is the ability to change the state of jobs. A job is a process that was started from the terminal window. This section describes commonly used commands for job control.

bg

A paused process can be restarted in the background by using the **bg** command:

```
[student@OCS ~]$ jobs
[1]+ Stopped                 sleep 999
[student@OCS ~]$ bg %1
[1]+ sleep 999 &
```

```
[student@OCS ~]$ jobs
[1]+ Running                 sleep 999 &
```

The **bg** command has no commonly used options.

> **Note**
>
> You can pause a process that is running in the foreground by holding down the Ctrl key and pressing z while in that process's window. See the "Ctrl+Z" section, later in this chapter, for more details.

fg

A paused process can be restarted in the foreground by using the **fg** command:

```
[student@OCS ~]$ jobs
[1]+ Stopped sleep 999
[student@OCS ~]$ fg %1
sleep 999
```

The **fg** command has no commonly used options.

> **Note**
>
> You can pause a process that is running in the foreground by holding down the Ctrl key and pressing z while in that process's window. See the "Ctrl+Z" section, later in this chapter, for more details.

jobs

The **jobs** command displays processes that are currently running and that were started from the shell in which you type the **jobs** command:

```
[student@OCS ~]$ jobs
[1]+ Stopped sleep 999
```

If you open another terminal window and type the **jobs** command, you don't see the processes that were displayed in your first terminal.

Ctrl+Z

When a process is running in the foreground, you can have a SIGTSTP signal sent to a process by holding down the Ctrl key and pressing the z key. A SIGTSTP signal is designed to pause a program. The program can then be restarted by using either the **bg** or **fg** command.

See the "**bg**" and "**fg**" sections, earlier in this chapter, for more details.

Ctrl+C

When a process is running in the foreground, you can have a SIGINT signal sent to a process by holding down the Ctrl key and pressing the c key. A SIGINT signal is designed to stop a program prematurely.

Ctrl+D

When you are running a process that accepts user input, such as the **at** command, you can send the process a signal that says "I am done providing input" by holding down the Ctrl key and pressing the d key (refer to the section "**at**," earlier in this chapter).

When you have finished entering **at** commands, hold down the Ctrl key and press the d key. This results in an **<EOT>** message and creates the **at** job. Here's an example:

```
[root@OCS ~]$ at 5pm tomorrow
at> /home/bob/rpt.pl
at> echo "report complete" | mail bob
at> <EOT>
job 1 at Thu Feb 23 17:00:00 2017
```

pgrep

Typically you use a combination of the **ps** and **grep** commands to display specific processes, like so:

```
[student@OCS ~]$ ps -e | grep sleep
25194 pts/0 00:00:00 sleep
```

However, the **pgrep** command can provide similar functionality:

```
[student@OCS ~]$ pgrep sleep
25194
```

Table 4.14 details some important options for the **pgrep** command.

TABLE 4.14 **pgrep Command Options**

Option	Description
-G *name*	Matches processes by group name.
-l	Displays the process name and PID.
-n	Displays the most recently started processes first.
-u *name*	Matches processes based on username.

pkill

When sending signals to a process using the **kill** command, you indicate which process by providing a process ID (PID). With the **pkill** command, you can provide a process name, a username, or another method to indicate which process or processes to send a signal to. For example, the following command sends a kill signal to all processes owned by the user sarah:

```
[student@OCS ~]$ pkill -u sarah
```

Table 4.15 details some important options for the **pkill** command.

TABLE 4.15 **pkill Command Options**

Option	Description
-G *name*	Matches processes by group name.
-u *name*	Matches processes by username.

pidof

The **pidof** command is useful when you know the name of a command that you want to control, but you don't know the command's PID (process ID). This command looks up a process by name and returns its PID:

```
[student@OCS ~]$ pidof dovecot
688
```

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which command can be used to control services?
   - ❍ **A. systemd**
   - ❍ **B. system**
   - ❍ **C. systemcfg**
   - ❍ **D. systemctl**

2. You run the **crontab -l** command and see the following output:

   ```
   15 12 * 3 1 /home/bob/rpt.pl
   ```

   When will this command execute?
   - ❍ **A.** At 3:15 on December 1
   - ❍ **B.** Every Monday of December at 3:15
   - ❍ **C.** Every Monday of March at 12:15
   - ❍ **D.** Never; this is an invalid **crontab** entry.

3. Which signal will be ignored when you run a process using the **nohup** command?
   - ❍ **A. -s**
   - ❍ **B. -h**
   - ❍ **C. -l**
   - ❍ **D.** None of these answers are correct.

4. Which of the following process priorities can a non-root user use?
   - ❍ **A. -20**
   - ❍ **B. -10**
   - ❍ **C. -1**
   - ❍ **D. 10**

Cram Quiz Answers

1. **D.** The **systemctl** command is used to control the state of services. The term **systemd** refers to the daemon that is running, not the command to control the services. The other answers are not valid commands for controlling services.

2. **C.** The time fields of the **crontab** output are in this order: Minute, Hour, Day of the Month, Month of the Year, and Day of the Week. Given this, the time the command would run is 12:15. No day of the month is specified (* means "not specified"), but **1** in the Day of the Week field means "every Monday," while the **3** value in the Month field means "March" (which is the third month of the year).
3. **A.** You can have a process ignore a SIGUP signal by executing the child process with the **nohup** command.
4. **D.** Non-root users cannot use negative number priorities.

This page intentionally left blank

CHAPTER 5

Use the Appropriate Networking Tools or Configuration Files

This chapter covers the following Linux+ XK0-005 exam objective:

- **1.5:** Given a scenario, use the appropriate networking tools or configuration files.

The goal of this chapter is to provide you with an understanding of different networking tools and configuration files. You will first learn about a variety of tools that are designed to display network information and allow you to make dynamic changes to network settings. Next, you will learn about which configuration files to edit to make network changes that will be persistent across a reboot.

This chapter also explores tools that allow you to monitor network traffic and verify that the network is behaving correctly. Finally, you will learn about some tools that allow you to connect to remote systems, either to log in to these systems or to transfer data.

This chapter provides information on the following topics: interface management, name resolution, network monitoring, and remote networking tools.

Interface Management

An *interface* (or, more specifically, a *network interface*) in Linux is a device that provides access to the network. This section covers the tools that allow you to manage network interfaces.

iproute2 Tools

iproute2 tools refers to a collection of tools that are designed to replace the legacy net tools. (See the section "net-tools," later in this chapter,

for more details about that collection.) The iproute2 tools include the **ip** and **ss** commands, which are covered in this section.

ip

The **ip** command is a newer command that is designed to replace a collection of commands related to network interfaces.

Syntax:

```
ip [options] object command
```

Table 5.1 describes some of the most important **ip** command objects.

TABLE 5.1 **ip Command Objects**

Object	Refers to
addr	IPv4 or IPv6 address
link	Network device
route	Routing table entry

Table 5.2 describes some of the most important **ip**-related commands that can be executed.

TABLE 5.2 **Commands to Add, Delete, and Analyze Objects**

Command	Description
add	Adds an object.
delete	Deletes an object.
show (or **list**)	Displays information about an object.

The following example displays network information for devices, much like the **ifconfig** command:

```
[root@OCS ~]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
     inet 127.0.0.1/8 scope host lo
     valid_lft forever preferred_lft forever
```

```
     inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
     link/ether 08:00:27:b0:dd:dc brd ff:ff:ff:ff:ff:ff
     inet 192.168.1.26/24 brd 192.168.1.255 scope global dynamic
enp0s3
     valid_lft 2384sec preferred_lft 2384sec
     inet 192.168.1.24/16 brd 192.168.255.255 scope global enp0s3
     valid_lft forever preferred_lft forever
     inet6 fe80::a00:27ff:feb0:dddc/64 scope link
     valid_lft forever preferred_lft forever
```

ExamAlert

The **ip** command was designed to replace utilities like **ipconfig**, **arp**, and **route**. This may come up in questions on the Linux+ XK0-005 exam.

ss

The **ss** command is used to display socket information. Think of a socket as an existing network connection between two systems.

Syntax:

```
ss [options]
```

Without any options, this command lists all open sockets. For example:

```
[root@OCS ~]# ss | wc -l
160
[root@OCS ~]# ss | head
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port
u_str ESTAB 0 0 /var/run/dovecot/anvil 23454966 * 23454965
u_str ESTAB 0 0 /var/run/dovecot/anvil 23887673 * 23887672
u_str ESTAB 0 0 /run/systemd/journal/stdout 13569 * 13568
u_str ESTAB 0 0                             * 13893 * 13894
u_str ESTAB 0 0                             * 13854 * 13855
u_str ESTAB 0 0                             * 13850 * 13849
```

```
u_str ESTAB 0 0                              * 68924 * 68925
u_str ESTAB 0 0                              * 17996 * 17997
u_str ESTAB 0 0     /var/run/dovecot/config 9163531 * 9163871
```

Table 5.3 describes some useful options for the **ss** command.

TABLE 5.3 **ss Command Options**

Option	Description
-lt	Lists listening TCP sockets.
-lu	Lists listening UDP sockets.
-lp	Lists the process ID that owns each socket.
-n	Specifies not to resolve IP addresses to hostnames or port numbers to port names.
-a	Displays all information.
-s	Displays a summary.

ExamAlert

Network essentials like TCP and UDP are not specific exam topics, so they are not covered in this book. CompTIA considers the Linux+ XK0-005 exam to be a higher level exam than the A+ exam. In other words, there is an assumption that you already have knowledge of network essentials. Not having that knowledge can affect your ability to understand questions on the Linux+ exam. If you don't have an understanding of networking basics, consider researching this topic before taking the Linux+ exam.

NetworkManager

The NetworkManager software is used to manage network devices in Linux. The primary command-line configuration tool for NetworkManager is **nmcli**, which is covered in this section.

nmcli

The **nmcli** command is used to configure NetworkManager, which is designed to detect and configure network connections.

Syntax:

```
nmcli [options] object [command]
```

Example:

```
[root@OCS ~]# nmcli device status
DEVICE      TYPE       STATE      CONNECTION
virbr0      bridge     connected  virbr0
enp0s3      ethernet   connected   enp0s3
lo          loopback   unmanaged   --
virbr0-nic  tun        unmanaged   --
```

object is one of the keywords listed in Table 5.4.

TABLE 5.4 **Objects to the nmcli Command**

Keyword	Description
connection	Manages network connections.
device	Manages a specific device.
general	Gets NetworkManager status information.
networking	Enables or disables networking or displays the current status.
radio	Finds radio (wireless) networking information and configuration.

Common **nmcli** commands are described in Table 5.5.

TABLE 5.5 **Commands That Can Be Used as Arguments to the nmcli Command**

Command	Description
status	Displays the current setting.
on|off	Turns a setting on (or off).
up|down	Brings an interface up (or down).
add|delete	Adds a new device or deletes an existing one.

net-tools

The term *net-tools* refers to a collection of Linux commands that display or modify network information. For example, **ifconfig**, **route**, **arp**, and **netstat** (which are covered in this section) are all part of the net-tools collection.

ifconfig

One of the commonly used commands for displaying network information is the **ifconfig** command. When executed with no arguments, it lists active network devices, as shown in the following example.

```
[root@onecoursesource ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.16  netmask 255.255.255.0  broadcast
192.168.1.255
        inet6 fe80::a00:27ff:fe52:2878  prefixlen 64  scopeid
0x20<link>
        ether 08:00:27:52:28:78  txqueuelen 1000  (Ethernet)
        RX packets 20141  bytes 19608517 (18.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2973  bytes 222633 (217.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 3320  bytes 288264 (281.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3320  bytes 288264 (281.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The output shows network information for two devices: the primary Ethernet network card (eth0) and the local loopback address (lo). If the system had additional Ethernet network cards, they would be displayed as eth1, eth2, and so on. The purpose of the loopback address is to allow software to communicate with the local system using protocols and services that would normally require the communication to occur on a network. In most cases, there is not much for you to administer or troubleshoot in regard to the loopback address.

ifcfg

ifcfg is a script that is specifically designed to replace the **ifconfig** functions of adding, deleting, and disabling IP addresses.

hostname

The **hostname** command can display or change the system hostname, as shown here:

```
[root@onecoursesource ~]# hostname
onecoursesource
[root@onecoursesource ~]# hostname myhost
[root@myhost ~]# hostname
myhost
```

arp

The **arp** command is used to view the ARP table or make changes to it. When executed with no arguments, the **arp** command displays the ARP table, as shown here:

```
# arp
Address      HWtype HWaddress          Flags Mask Iface
192.168.1.11 ether  30:3a:64:44:a5:02 C          eth0
```

If a remote system has its network card replaced, it may be necessary to delete an entry from the ARP table. This can be accomplished by using the **-d** option to the **arp** command:

```
# arp -i eth0 -d 192.169.1.11
```

Once the address has been removed from the ARP table, there should be no need to add the new address manually. The next time the local system uses this IP address, it sends a broadcast request on the appropriate network to determine the new MAC address.

route

The **route** command can be used to display the routing table:

```
[root@OCS ~]# route
Kernel IP routing table
Destination Gateway     Genmask      Flags Metric Ref Use Iface
default     192.168.1.1  0.0.0.0     UG    100     0   0  enp0s3
```

```
192.168.0.0 0.0.0.0      255.255.0.0  U    100      0    0  enp0s3
192.168.1.0 0.0.0.0      255.255.0.0  U    100      0    0  enp0s3
```

This information can also be displayed with the **ip** command:

```
[root@OCS ~]# ip route show
default via 192.168.1.1 dev enp0s3 proto static metric 100
192.168.0.0/16 dev enp0s3 proto kernel scope link src 192.168.1.24
metric 100
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.26
metric 100
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1
```

The **route** command can also be used to modify the default router:

```
route add default gw 192.168.1.10
```

To add a new router, execute the following command:

```
route add -net 192.168.3.0 netmask 255.255.255.0 gw 192.168.3.100
```

This command sends all network packets destined for the 192.168.3.0/24 network to the 192.168.3.100 router.

> **Note**
>
> **route** command changes are temporary and will only survive until the next time the system is booted. Permanent changes are made within your system's configuration files, which vary from one distribution to another.

/etc/sysconfig/network-scripts/

The directory **/etc/sysconfig/network-scripts/** is found on Red Hat–based distributions, such as Red Hat Enterprise Linux, CentOS, and Fedora. It contains a collection of files that are used to configure network devices, which you can see from the output in the following command.

```
[root@OCS ~]# ls /etc/sysconfig/network-scripts/
ifcfg-eth0      ifdown-Team        ifup-plusb
ifcfg-lo        ifdown-TeamPort    ifup-post
ifdown          ifdown-tunnel     ifup-ppp
```

```
ifdown-bnep     ifup              ifup-routes
ifdown-eth      ifup-aliases      ifup-sit
ifdown-ippp     ifup-bnep         ifup-Team
ifdown-ipv6     ifup-eth          ifup-TeamPort
ifdown-isdn     ifup-ippp         ifup-tunnel
ifdown-post      ifup-ipv6        ifup-wireless
ifdown-ppp       ifup-ipx         init.ipv6-global
ifdown-routes    ifup-isdn        network-functions
ifdown-sit       ifup-plip        network-functions-ipv6
```

In most cases, you can probably guess what a configuration file is used for based on its name. For example, the **ifup-wireless** file is used to configure wireless networks. Many of these files also have comments that are used to describe the purpose and use of the file.

The file most commonly edited is **ifcfg-***interface*, where *interface* is the name of the network interface. For example, **ifcfg-eth0** is used to configure the eth0 device, as shown here:

```
[root@OCS ~]# more /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
IPADDR=192.168.0.100
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
```

Table 5.6 lists some common **ifcfg-***interface* configuration settings.

TABLE 5.6 **ifcfg-*interface* Configuration Settings**

Setting	Description
DEVICE	The name of the interface.
BOOTPROTO	Set to **static** if you manually provide network information, such as IP address, netmask, and gateway. Set to **dhcp** to have these values dynamically assigned from a DHCP server.
ONBOOT	Normally set to **yes** to activate this device during the boot process.
IPADDR	The IP address that should be assigned to the device.
NETMASK	The device's netmask.
GATEWAY	The default router for this interface.

Name Resolution

Name resolution is the process of determining the IP address that corresponds to a given hostname. Reverse name resolution is the process of determining the hostname that corresponds to a given IP address. This section describes commands and features related to name resolution.

nsswitch

nsswitch refers to Name Service Switch (NSS), which is a tool for determining where the system will look for name resolution.

The NSS configuration file, **/etc/nsswitch.conf**, is used by applications to determine the sources from which to obtain name-service information and in what order. For example, for networking, this file contains the location of the name server resolver, the utility that provides hostname-to-IP-address translation.

```
[root@onesourcesource ~]#grep hosts /etc/nsswitch.conf
#hosts:     db files nisplus nis dns
hosts:      files dns
```

In this example, **files dns** means "look at the local **/etc/hosts** file first and then look at the DNS server if the required translation isn't in the local file."

Table 5.7 describes common hostname-to-IP-address translation utilities.

TABLE 5.7 **Hostname-to-IP-Address Translation Utilities**

Utility	Description
files	The local **/etc/hosts** file
dns	A DNS server
NIS	A Network Information Service server

/etc/resolv.conf

The **/etc/resolv.conf** file contains a list of the DNS servers for the system. A typical file looks as follows:

```
[root@OCS ~]# cat /etc/resolv.conf
search sample999.com
nameserver 192.168.1
```

If you are using a utility such as NetworkManager to configure your network settings or if you are using a DHCP client, then this file is normally populated by those utilities. For servers, this file is typically manually defined.

Table 5.8 describes common settings for the **/etc/resolv.conf** file.

TABLE 5.8 **Common Settings for the /etc/resolv.conf File**

Setting	Description
nameserver	The IP address of the DNS server. There can be up to three **nameserver** lines in the file.
domain	Used to specify the local domain, which allows for the use of short names for DNS queries.
search	A list of optional domains to perform DNS queries when using short names.

systemd

The **systemd** utility can be used to enable and disable network functionality. See the "System Services" section in Chapter 4, "Configure and Use the Appropriate Processes and Services," for more information.

hostnamectl

The **hostnamectl** command can be used to view and change host and system information. When it is used with no arguments, information about the system is displayed:

```
# hostnamectl
 Static hostname: student-VirtualBox
         Icon name: computer-vm
           Chassis: vm
    Machine ID: 7235c52cf8114b8188c985c05afe75c9
       Boot ID: e6ba643d8da44542a90a26c4466adca7
 Virtualization: oracle
       Operating System: Ubuntu 18.04.1 LTS
        Kernel: Linux 4.15.0-43-generic
           Architecture: x86-64
```

The **set-hostname** option allows you to specify one of two types of hostnames:

- **--static:** Changes are made in the **/etc/hostname** file and are persistent across reboots.

- **--transient:** Changes only apply to currently booted system. No changes are made to the **/etc/hostname** file.

There is also a feature (the **--pretty** option) that allows you to make a more flexible hostname that breaks standard network hostname rules. With **--pretty** and **--static**, changes are made to the **/etc/machine-info** file.

resolvectl

The **resolvectl** command can perform DNS lookups. For example, it can be used as follows to resolve the hostname of google.com:

```
[root@OCS ~]#  resolvectl query google.com
google.com: 142.250.189.238


-- Information acquired via protocol DNS in 611.6ms.
-- Data is authenticated: no
```

Bind-utils

Bind-utils is a software package that provides many of the commonly used commands for performing DNS queries. Examples of commands found in the Bind-utils package include **dig**, **host**, and **nslookup**. These commands are described in greater detail in this section.

The Bind-utils package is not normally installed by default on some Linux distributions, and you may need to install it.

dig

The **dig** command is useful for performing DNS queries on specific DNS servers. The use of the command is demonstrated here:

```
[root@OCS ~]# dig google.com


; <<>> DiG 9.9.4-RedHat-9.9.4-38.el7_3 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56840
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
```

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.         IN     A

;; ANSWER SECTION:
google.com. 268     IN     A 216.58.217.206

;; Query time: 36 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Sun Mar 05 17:01:08 PST 2017
;; MSG SIZE rcvd: 55
```

To query a specific DNS server rather than the default DNS servers for your host, use the following syntax:

```
dig @server host_to_look_up
```

Table 5.9 describes common options for the **dig** command.

TABLE 5.9 **dig Command Options**

Option	Description
-f *file*	Uses the contents of *file* to perform multiple lookups; the file should contain one hostname per line.
-4	Indicates to only perform IPv4 queries.
-6	Indicates to only perform IPv6 queries.
-x *address*	Performs a reverse lookup (and returns the hostname when provided an IP address).

nslookup

The **nslookup** command is designed to perform simple queries on DNS servers.

Syntax:

```
nslookup hostname
```

Example:

```
[root@OCS ~]# nslookup google.com
Server:        8.8.8.8
```

```
Address:      8.8.8.8#53

Non-authoritative answer:
Name:     google.com
Address: 216.58.219.238
```

While this command is often referred to as obsolete, it is still often used on modern distributions. The **nslookup** command is normally run without options.

host

The **host** command is normally used to perform simple hostname-to-IP-address translation operations (also called *DNS queries*). Here is an example:

```
[root@OCS ~]# host google.com
google.com has address 172.217.4.142
google.com has IPv6 address 2607:f8b0:4007:800::200e
google.com mail is handled by 30 alt2.aspmx.l.google.com.
google.com mail is handled by 50 alt4.aspmx.l.google.com.
google.com mail is handled by 20 alt1.aspmx.l.google.com.
google.com mail is handled by 10 aspmx.l.google.com.
google.com mail is handled by 40 alt3.aspmx.l.google.com.
```

Table 5.10 describes common options for the **host** command.

TABLE 5.10 **host Command Options**

Option	Description
-t	Specifies the type of query that you want to display; for example, **host -t ns google.com** displays Google's name servers.
-4	Indicates to only perform IPv4 queries.
-6	Indicates to only perform IPv6 queries.
-v	Enters verbose mode, with output like that of the **dig** command.

WHOIS

The **whois** command is useful for determining which company or person owns a domain. Often the output also contains information regarding how to contact this organization, although this information might be redacted for privacy reasons. Here is an example:

```
# whois onecoursesource.com | head
   Domain Name: ONECOURSESOURCE.COM
   Registry Domain ID: 116444640_DOMAIN_COM-VRSN
   Registrar WHOIS Server: whois.tucows.com
   Registrar URL: http://www.tucows.com
   Updated Date: 2016-01-15T01:49:45Z
   Creation Date: 2004-04-07T19:45:31Z
   Registry Expiry Date: 2021-04-07T19:45:31Z
   Registrar: Tucows Domains Inc.
   Registrar IANA ID: 69
   Registrar Abuse Contact Email:
```

Network Monitoring

Network monitoring is the process of watching traffic on the network to determine if there are network traffic issues. This section describes many of the commonly used network monitoring tools.

tcpdump

When troubleshooting network issues or performing network security audits, it can be helpful to view the network traffic, including traffic that isn't related to the local machine. The **tcpdump** command is a packet sniffer that allows you to view local network traffic.

By default, the **tcpdump** command displays all network traffic to standard output until you terminate the command. This could result in a dizzying amount of data flying by on your screen. You can limit the output to a specific number of network packets by using the **-c** options, as in this example:

```
# tcpdump -c 5
tcpdump: verbose output suppressed, use -v or -vv for full
 protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535
bytes
11:32:59.630873 IP localhost.43066 > 192.168.1.1.domain:
 16227+ A? onecoursesource.com. (37)
```

```
11:32:59.631272 IP localhost.59247 > 192.168.1.1.domain:
 2117+ PTR? 1.1.168.192.in-addr.arpa. (42)
11:32:59.631387 IP localhost.43066 > 192.168.1.1.domain:
 19647+ AAAA? onecoursesource.com. (37)
11:32:59.647932 IP 192.168.1.1.domain > localhost.59247:
 2117 NXDomain* 0/1/0 (97)
11:32:59.717499 IP 192.168.1.1.domain > localhost.43066:
 16227 1/0/0 A 38.89.136.109 (53)
5 packets captured
5 packets received by filter
0 packets dropped by kernel
```

Wireshark/tshark

Wireshark is an amazing network sniffer that provides both GUI-based and TUI-based tools. To start the GUI tool, execute the **wireshark** command. The output should be similar to that shown in Figure 5.1.

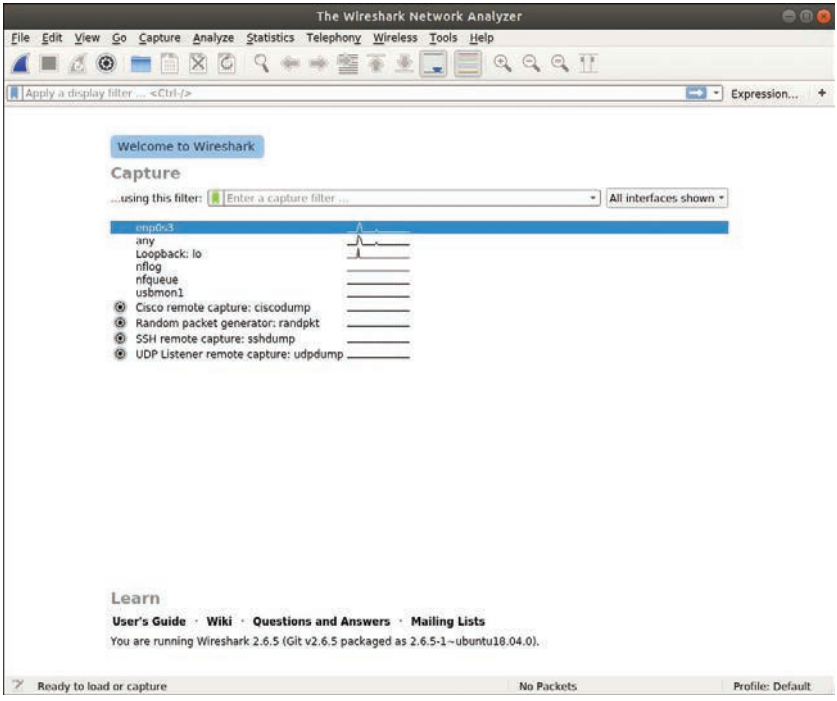

FIGURE 5.1 **The wireshark Command**

To view network traffic, you need to start a capture. Click **Capture**, **Start**. You can also limit what is captured by setting filters and options (click **Capture**, **Options**).

To use the TUI-based form of Wireshark, execute the **tstark** command as the root user. Here is an example:

```
# tshark
Capturing on 'enp0s3'
 1 0.000000000 10.0.2.15 → 68.105.28.11 DNS
 81 Standard query 0xeec4 A google.com OPT
 2 0.001031279 10.0.2.15 → 68.105.28.11 DNS
 81 Standard query 0x3469 AAAAgoogle.com OPT
 3 0.017196416 68.105.28.11 → 10.0.2.15 DNS
 109 Standard query response 0x3469 AAAA google.com AAAA
2607:f8b0:4007:800::200e OPT
 4 0.017265061 68.105.28.11 → 10.0.2.15 DNS
 97 Standard query response 0xeec4 A google.com A 172.217.14.110 OPT
 5 0.018482388 10.0.2.15 → 172.217.14.110 ICMP
 98 Echo (ping) request id=0x122c, seq=1/256, ttl=64
 6 0.036907577 172.217.14.110 → 10.0.2.15 ICMP
 98 Echo (ping) reply id=0x122c, seq=1/256, ttl=251 (request in 5)
 7 1.021052811 10.0.2.15 → 172.217.14.110 ICMP
 98 Echo (ping) request id=0x122c, seq=2/512, ttl=64
 8 1.039492225 172.217.14.110 → 10.0.2.15 ICMP
 98 Echo (ping) reply id=0x122c, seq=2/512, ttl=251 (request in 7)
```

netstat

The **netstat** command is useful for displaying a variety of network information. It is a key utility when troubleshooting network issues. Table 5.11 describes common options for the **netstat** command.

TABLE 5.11 **netstat Command Options**

Option	Description
-t or **--tcp**	Displays TCP information.
-u or **--udp**	Displays UDP information.
-r or **--route**	Displays the routing table.

Option	Description
-v or **--verbose**	Enters verbose mode and displays additional information.
-i or **--interfaces**	Displays information based on a specific interface.
-a or **--all**	Applies to all.
-s or **--statistics**	Displays statistics for the output.

For example, the following command displays all active TCP connections:

```
[root@OCS ~]# netstat -ta
Active Internet connections (servers and established)
Proto  Recv-Q  Send-Q Local Address          Foreign Address State
tcp    0       0       192.168.122.1:domain  0.0.0.0:* LISTEN
tcp    0       0       0.0.0.0:ssh           0.0.0.0:* LISTEN
tcp    0       0       localhost:ipp         0.0.0.0:* LISTEN
tcp    0       0       localhost:smtp        0.0.0.0:* LISTEN
tcp6   0       0       [::]:ssh              [::]:* LISTEN
tcp6   0       0       localhost:ipp          [::]:* LISTEN
tcp6   0      0      localhost:smtp         [::]:* LISTEN
```

traceroute

When you send a network packet to a remote system, especially across the Internet, it often needs to go through several gateways before it reaches its destination. You can see the gateways that the packet passes through by executing the **traceroute** command, as shown here:

```
# traceroute onecoursesource.com
traceroute to onecoursesource.com (38.89.136.109), 30 hops max, 60
byte packets
 1 10.0.2.2 (10.0.2.2) 0.606 ms 1.132 ms 1.087 ms
 2 b001649-3.jfk01.atlas.cogentco.com (38.104.71.201)
 0.738 ms 0.918 ms 0.838 ms
 3 154.24.42.205 (154.24.42.205) 0.952 ms 0.790 ms 0.906 ms
 4 be2629.ccr41.jfk02.atlas.cogentco.com (154.54.27.66)
 1.699 ms 1.643 ms 1.347 ms
 5 be2148.ccr41.dca01.atlas.cogentco.com (154.54.31.117)
 8.053 ms 7.719 ms 7.639 ms
 6 be2113.ccr42.atl01.atlas.cogentco.com (154.54.24.222)
```

```
 18.276 ms 18.418 ms 18.407 ms
 7 be2687.ccr21.iah01.atlas.cogentco.com (154.54.28.70)
 32.861 ms 32.917 ms 32.719 ms
 8 be2291.ccr21.sat01.atlas.cogentco.com (154.54.2.190)
 38.087 ms 38.025 ms 38.076 ms
 9 be2301.ccr21.elp01.atlas.cogentco.com (154.54.5.174)
 48.811 ms 48.952 ms 49.151 ms
10 be2254.ccr21.phx02.atlas.cogentco.com (154.54.7.33)
 57.332 ms 57.281 ms 56.896 ms
11 te2-1.mag02.phx02.atlas.cogentco.com (154.54.1.230)
 56.666 ms 65.279 ms 56.520 ms
12 154.24.18.26 (154.24.18.26) 57.924 ms 58.058 ms 58.032 ms
13 38.122.88.218 (38.122.88.218) 79.306 ms 57.740 ms 57.491 ms
14 onecoursesource.com (38.89.136.109) 58.112 57.884 ms 58.299 ms
```

ping

The **ping** command is used to verify that a remote host can respond to a network connection:

```
 [root@OCS ~]# ping -c 4 google.com
PING google.com (172.217.5.206) 56(84) bytes of data.
64 bytes from lax28s10-in-f14.1e100.net (172.217.5.206): icmp_seq=1
ttl=55 time=49.0 ms
64 bytes from lax28s10-in-f206.1e100.net (172.217.5.206): icmp_seq=2
ttl=55 time=30.2 ms
64 bytes from lax28s10-in-f14.1e100.net (172.217.5.206): icmp_seq=3
ttl=55 time=30.0 ms
64 bytes from lax28s10-in-f206.1e100.net (172.217.5.206): icmp_seq=4
ttl=55 time=29.5 ms


--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 29.595/34.726/49.027/8.261 ms
```

By default, the **ping** command continuously sends pings to the remote system until the user cancels the command (by pressing Ctrl+C). The **-c** option specifies how many ping requests to send.

mtr

If you want a really cool variation of the **traceroute** command (see the "**traceroute**" section, earlier in this chapter), install the **mtr** command. This command performs a **traceroute**-like operation every second, updating the display with statistics, as demonstrated in Figure 5.2.

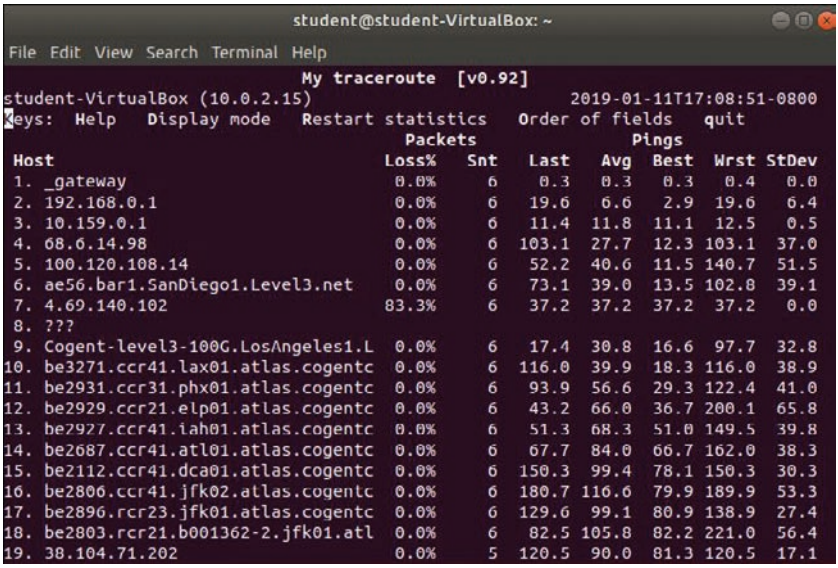

FIGURE 5.2 **The mtr Command**

Remote Networking Tools

Remote networking tools are designed to enable you to connect to remote systems for the following purposes:

- To execute commands on the remote systems
- To transfer files to the remote systems from your local system
- To transfer files from the remote systems to your local system

This section describes some of the most popular remote networking tools.

Secure Shell (SSH)

The **ssh** command is a utility that allows you to connect to a Secure Shell (SSH) server. The syntax of the command is as follows:

```
ssh user@hostname
```

where *user* is the username you want to use to log in as, and *hostname* is a system hostname or IP address.

The first time you use the **ssh** command to connect to a system, you see the following prompt:

```
[root@OCS ~]# ssh bob@server1
The authenticity of host 'server1' can't be established.
ECDSA key fingerprint is
 8a:d9:88:b0:e8:05:d6:2b:85:df:53:10:54:66:5f:0f.
Are you sure you want to continue connecting (yes/no)?
```

This ensures that you are logging in to the correct system. Typically users answer **yes** to this prompt, assuming that they are logging in to the correct machine, but this information can also be verified independently by contacting the system administrator of the remote system.

After the user answers **yes** to this prompt, the SSH server fingerprint is stored in the **known_hosts** file in the **~/.ssh** directory.

Table 5.12 describes common options for the **ssh** command.

TABLE 5.12 **ssh Command Options**

Option	Description
-F *configfile*	Specifies the configuration file to use for the **ssh** client utility. The default configuration file is **/etc/ssh/ssh_config**.
-4	Indicates to use only IPv4 addresses.
-6	Indicates to use only IPv6 addresses.
-E *logfile*	Places errors in the specified log file rather than displaying them to standard output.

SSH data for individual users is stored in each user's home directory under the **.ssh** subdirectory. This directory is used by SSH to store important data, and users can modify configurations in this directory. This section focuses on the files that may be stored in the **~/.ssh** directory.

After a connection is established with an SSH server, the SSH client stores the server's unique fingerprint key in the user's **.ssh/known_hosts** file, as shown here:

```
[root@OCS ~]# cat .ssh/known_hosts
|1|trm4BuvRf0HzJ6wusHssj6HcJKg=|EruYJY709DXorogeN5Hdcf6jTCo=
 ecdsa-sha2-nistp256AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzd
 HAyNTYAAABBBG3/rARemyZrhIuirJtfpfPjUVnph9S1w2NPfEWec/f59V7nA
 ztn5rbcGynNYOdnozdGNNizYAiZ2VEhJ3Y3JcE=
```

Typically the contents of this file should be left undisturbed; however, if the SSH server is reinstalled, it has a new fingerprint key. All users must then remove the entry for the SSH server in the **.ssh/known_hosts** file.

When a user wants to use password-based SSH authentication, the first step is to create authentication keys by using the **ssh-keygen** command.

After the authentication key files have been created, the public key (the contents of either the **~/.ssh/id_dsa.pub** or **~/.ssh/id_rsa.pub** file) needs to be copied to the system that the user is attempting to log in to. This requires placing the public key into the **~/.ssh/authorized_keys** file on the SSH server. This can be accomplished by manually copying over the content of the public key file from the client system and pasting it into the **~/.ssh/authorized_keys** file on the SSH server. Alternatively, you can use the **ssh-copy-id** command, which has the following syntax:

```
ssh-copy-id user@server
```

Users can customize how commands like **ssh**, **scp**, and **sftp** work by creating the **~/.ssh/config** file. The format and settings in this file are the same as those in the **/etc/sshd/ssh.conf** file.

cURL

The **curl** command allows for noninteractive data transfer from a large number of protocols, including the following:

- FTP
- FTPS
- HTTP
- SCP
- SFTP

- SMB
- SMBS
- Telnet
- TFTP

ExamAlert

You should consider memorizing the list of protocols that **curl** supports.

Although the **curl** command supports more protocols than the **wget** command, the **wget** command can perform recursive downloads and can recover from failed download attempts, so it is advantageous in certain situations. The **curl** command also supports wildcard characters. The goal of both of the commands is essentially the same.

Syntax:

```
curl location
```

Example:

```
# curl http://onecoursesource.com
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved
 <a href="http://www.onecoursesource.com/">here</a>.</p>
</body></html>
```

Note that the data is displayed, not stored in a file, as it is with the **wget** command. You can use redirection to put the contents into a file.

wget

The **wget** command is designed to be a noninteractive tool for downloading files from remote systems via HTTP, HTTPS, or FTP. It is often used within scripts.

Syntax:

```
wget location
```

Example:

```
# wget http://onecoursesource.com
--2019-01-09 15:18:26-- http://onecoursesource.com/
Resolving onecoursesource.com (onecoursesource.com)...38.89.136.109
Connecting to onecoursesource.com (onecoursesource.com)
 |38.89.136.109|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www.onecoursesource.com/ [following]
--2019-01-09 15:18:26-- http://www.onecoursesource.com/
Resolving www.onecoursesource.com
 (www.onecoursesource.com)... 38.89.136.109
Reusing existing connection to onecoursesource.com:80.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html'

index.html [ <=> ]
 12.25K --.-KB/s in 0s

2019-01-09 15:18:26 (259 MB/s) - 'index.html' saved [12539]
```

Table 5.13 lists some useful options to use with the **wget** command.

TABLE 5.13 **wget Command Options**

Option	Description
-h	Displays help.
-b	Performs downloads in the background. This is useful for large downloads.
-q	Downloads quietly.
-v	Enters verbose mode.
-nc	Indicates not to clobber existing files.
-c	Continues a partial download. This is useful if a download fails due to a disconnect.
-r	Enters recursive mode.

nc

See the "**nc**" section in Chapter 2, "Manage Files and Directories."

rsync

See the "**rsync**" section in Chapter 2.

Secure Copy Protocol (SCP)

See the "**scp**" section in Chapter 2.

SSH File Transfer Protocol (SFTP)

The **sftp** command uses the SSH (Secure Shell) protocol to securely transfer files across the network. To access a remote system, use the following syntax:

```
sftp user@machine
```

After logging in to the remote system, you are provided with an interface that begins with the following prompt:

```
sftp>
```

At this prompt, several commands can be used to transfer files and perform other operations. Table 5.14 describes the most important of these commands.

TABLE 5.14 **Commands That Can Be Used at the sftp Prompt**

Command	Description
pwd	Displays the current remote directory.
lpwd	Displays the current local directory.
cd	Changes to a different directory on a remote system.
lcd	Changes to a different directory on the local system.
get	Downloads a file from the current directory on the remote system to the current directory on the local system. Use **-r** to get an entire directory structure.
put	Uploads a file from the current directory on the local system to the current directory on the remote system. Use **-r** to put an entire directory structure.
ls	Displays files in the current directory of a remote system.
lls	Displays files in the current directory of the local system.
exit	Quits **sftp**.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which command enables you to configure NetworkManager?
   - ❍ **A. ifconfig**
   - ❍ **B. netconfig**
   - ❍ **C. nmcli**
   - ❍ **D. nmconfig**

2. Which of the following tools can display network route information? (Choose two.)
   - ❍ **A. route**
   - ❍ **B. ifconfig**
   - ❍ **C. arp**
   - ❍ **D. ip**

3. Which setting in the **/etc/sysconfig/network-scripts/ifcfg-eth0** file is used to set the default router?
   - ❍ **A. DEVICE**
   - ❍ **B. ROUTE**
   - ❍ **C. NETMASK**
   - ❍ **D. GATEWAY**

4. Which of the following commands allow you to display information about network packets? (Choose two.)
   - ❍ **A. tcpdump**
   - ❍ **B. wireshark**
   - ❍ **C. netstat**
   - ❍ **D. mtr**

Cram Quiz Answers

1. **C.** The **nmcli** command is used to configure NetworkManager, which is designed to detect and configure network connections.
2. **A and D.** The **route** command can be used to display the routing table. This information can also be displayed with the **ip** command, as follows: **ip route show**.
3. **D.** The **GATEWAY** setting is used to define the default router for an interface.
4. **A and B.** Both **tcpdump** and Wireshark capture network packet information, which you can then display. The **netstat** command provides overall network statistics, and **mtr** displays the hops to get from one system to another.

CHAPTER 6

Build and Install Software

This chapter covers the following Linux+ XK0-005 exam objective:

- **1.6:** Given a scenario, build and install software.

This chapter explores how to manage software on different Linux distributions. You will learn the essentials of different Linux sandbox applications as well as the differences between kernel updates and system package updates.

This chapter provides information on the following topics: package management, sandboxed applications, and system updates.

Package Management

A *package* in Linux is a file that contains software that you can install on the system. The process of managing a package includes performing any of the following operations:

- Listing installed packages
- Viewing the contents of a package
- Installing a package
- Removing a package

This section reviews the various tools that are used to manage software packages on different platforms.

ExamAlert

While you might only work on one distribution, you need to be prepared for Linux+ XK0-005 exam questions related to package management on different distributions.

DNF

The DNF tool is designed as an enhancement and replacement for **yum**. (See the next section, "YUM," for more information.) The majority of the changes were made to the back end of the software. Most **dnf** commands work just like **yum** commands.

On the back end, the DNF tool handles dependencies better and addresses some additional YUM deficiencies (such as using older versions of Python). YUM hasn't been completely replaced, so knowing that either command may be used is important.

It is important to note that the configuration file for DNF is different from the configuration file for YUM. Configure DNF by editing the **/etc/dnf/dnf.conf** file.

YUM

The **yum** command is used to install software from repositories. It can also be used to remove software and display information regarding software. Table 6.1 highlights the primary **yum** commands and options.

TABLE 6.1 **yum Commands and Options**

Command/Option	Description
install	Installs a package and any dependency packages from a repository. Example: **yum install zip**.
groupinstall	Installs an entire software group from a repository. Example: **yum groupinstall "Office Suite and Productivity"**.
update	Updates the specified software package.
remove	Removes the specified software package and any dependency packages from the system.
groupremove	Removes the specified software group from the system.
list	Lists information about packages, including which packages are installed and which packages are available. Example: **yum list available**.
grouplist	Lists information about software groups, including what packages are part of a group; use **yum grouplist** with no arguments to see a list of available software groups.
info	Provides information about a specific software package. Example: **yum info zip**.
groupinfo	Provides information about a specific software group.

Command/Option	Description
-y	Answers **yes** automatically to any prompts. Example: **yum -y install zip**.

Table 6.2 describes some important options to the **yum list** command.

TABLE 6.2 **yum list Command Options**

Option	Description
all	Lists all packages that are installed or available.
installed	Lists all packages that are currently installed.
available	Lists all packages that are currently not installed but available for installation.
updates	Lists all packages that are currently installed on the system and that also have an available newer version on a repository.

Note

Wildcards (or *globs*) may be used with **yum** commands. Here's an example:

```
# yum list installed "*zip*"
Loaded plugins: fastestmirror, langpacks
Repodata is over 2 weeks old. Install yum-cron?
 Or run: yum makecache fast
Loading mirror speeds from cached hostfile
 * base: mirror.supremebytes.com
 * epel: mirror.chpc.utah.edu
 * extras: mirrors.cat.pdx.edu
 * updates: centos.sonn.com
Installed Packages
bzip2.x86_64                  1.0.6-13.el7   @base
bzip2-libs.x86_64             1.0.6-13.el7   @base
gzip.x86_64                   1.5-8.el7      @base
perl-Compress-Raw-Bzip2.x86_64 2.061-3.el7   anaconda
unzip.x86_64                  6.0-15.el7     @base
zip.x86_64                    3.0-10.el7     @anaconda
```

The **yumdownloader** command is used to download software packages without installing the software. The resulting RPM file could be installed manually or copied to other systems.

Table 6.3 describes some important options to the **yumdownloader** command.

TABLE 6.3 **yumdownloader Command Options**

Option	Description
--destdir	Used to specify the directory for downloading RPM files. (The default is the current directory.)
--resolve	Used to download dependency packages for the specified package. (The default is to download only specified packages.)
--source	Used to download the source RPM, not the binary (installable) RPM.

The **/etc/yum.conf** file is the primary configuration file for **yum** commands.

Example:

```
[main]
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=0
debuglevel=2
logfile=/var/log/yum.log
exactarch=1
obsoletes=1
gpgcheck=1
plugins=1
installonly_limit=5
bugtracker_url=http://bugs.centos.org/set_project.php
 ?project_id=23&ref=http://bugs.centos.org/
 bug_report_page.php?category=yum
distroverpkg=centos-release
```

Table 6.4 describes some key settings of the **/etc/yum.conf** file.

TABLE 6.4 **/etc/yum.conf File Key Settings**

Setting	Description
cachedir	Directory where RPMs will be placed after download.
logfile	Location of the log file that contains **yum** actions.

Setting	Description
gpgcheck	A value of **1** means perform a GPG (GNU Privacy Guard) check to ensure that the package is valid; **0** means do not perform a GPG check. (This can be overridden by specific settings for each repository configuration file.)
assumeyes	A value of **1** means always assume "yes" to yes/no prompts; **0** means do not make any assumption (but provide a prompt instead).

ExamAlert

Expect a Linux+ XK0-005 exam question on the key settings described in Table 6.4.

The **/etc/yum.repos.d** directory contains files that end in **.repo** and that are used to specify the location of **yum** repositories. Each file defines one or more repositories, as illustrated in Figure 6.1.

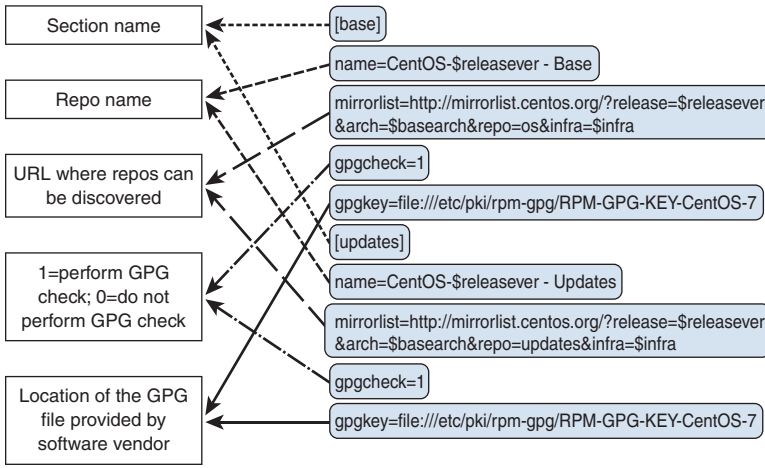

FIGURE 6.1 **The Format of Files in the /etc/yum.repos.d Directory**

APT

Use the **apt-get** command to manage Debian packages that are located on a repository. This command makes use of the **/etc/apt/sources.list** file to determine which repository to use. (See more about this file at the end of this section).

Here are some syntax examples of the **apt-get** command:

```
apt-get [options] command
apt-get [options] install|remove pkg1 [pkg2...]
apt-get [options] source pkg1 [pkg2...]
```

To specify what action to take, provide a keyword (command) to the **apt-get** command. Table 6.5 describes some useful commands for **apt-get**.

TABLE 6.5 **apt-get Commands**

Command	Description
install	Installs the specified package; if the package is currently installed, use the **--only-upgrade** option to upgrade rather than install fresh.
update	Updates the package cache of every available package.
upgrade	Updates all packages and their dependencies.
remove	Removes a package but leaves its configuration files on the system.
purge	Removes a package, including its configuration files.

Use the **apt-cache** command to display package information regarding the package cache.

Here are some syntax examples of the **apt-cache** command:

```
apt-cache [options] command
apt-cache [options] show pkg1 [pkg2...]
```

Example:

```
# apt-cache search xzip
xzip - Interpreter of Infocom-format story-files
```

To specify what action to take, provide a keyword (command) to the **apt-cache** command. Table 6.6 describes some useful commands for **apt-cache**.

TABLE 6.6 **apt-cache Commands**

Command	Description
search	Displays all packages with the search term listed in the package name or description; the search term can be a regular expression.
showpkg	Displays information about a package; the package name is provided as an argument.

Command	Description
stats	Displays statistics about the package cache (for example, **apt-cache stats**).
showsrc	Displays information about a source package; the package name is provided as an argument.
depends	Displays a package's dependencies.
rdepends	Displays a package's reverse dependencies (that is, packages that rely on this package).

The **aptitude** utility is a menu-driven tool designed to make it easy to display, add, and remove packages. Figure 6.2 shows this tool.

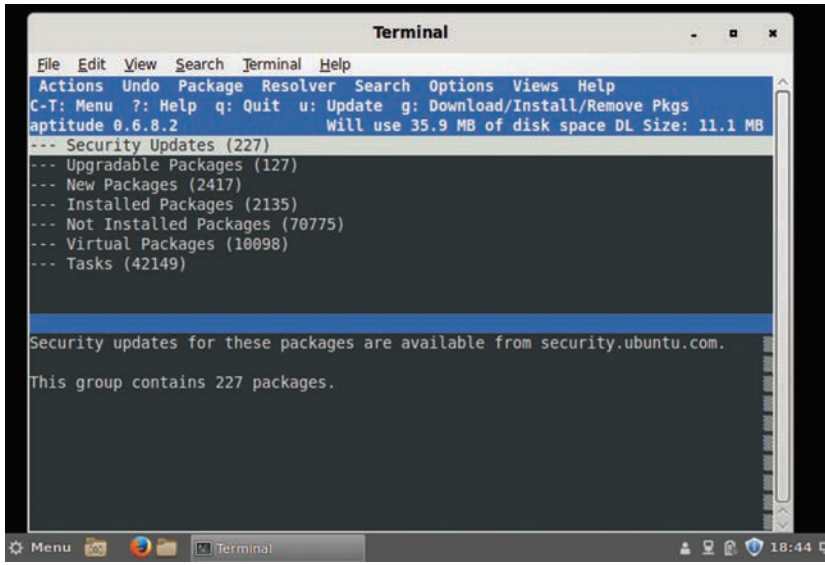

FIGURE 6.2 **The aptitude Utility Screen**

The **/etc/apt/sources.list** file contains a list of URLs of software repositories; there can also be files in the **/etc/apt/sources.list.d** directory that provide this information. Figure 6.3 describes this file.

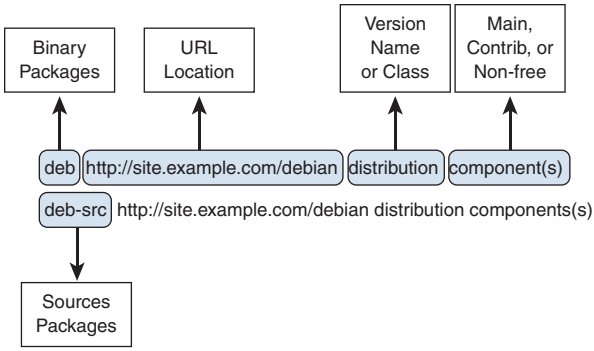

FIGURE 6.3 **The /etc/apt/sources.list File**

The distribution can be one of the following:

- Release name (for example, wheezy, jessie, stretch, or sid)
- Class name (for example, oldstable, stable, testing, or unstable)

The component can be one of the following:

- **main:** Packages must comply with DFSG (Debian Free Software Guidelines).
- **contrib:** Packages must comply with DFSG, but package dependencies do not have to.
- **non-free:** Packages do not comply with DFSG.

ExamAlert

Know the different components described in the previous list before you take the Linux+ XK0-005 exam.

The following is an example of the default file for the Jessie version of Debian:

```
deb http://httpredir.debian.org/debian jessie main
deb-src http://httpredir.debian.org/debian jessie main

deb http://httpredir.debian.org/debian jessie-updates main
deb-src http://httpredir.debian.org/debian jessie-updates main

deb http://security.debian.org/ jessie/updates main
deb-src http://security.debian.org/ jessie/updates main
```

The primary configuration file for APT is the **/etc/apt.conf** file. This file can be used to set debug options, configure proxy connections, and configure how APT caches packages.

RPM

The **rpm** command is useful for installing, upgrading, and removing packages that are already downloaded on your system. Table 6.7 describes some of the useful options.

TABLE 6.7 **rpm Command Options**

Option	Description
-i	Installs a package.
-U	Updates a package if an older version of the package exists; installs from scratch if the older version does not exist.
-F	Updates a package if an older version of the package exists; does nothing if an older version does not exist.
-e	Removes the package, including the configuration files.
-l	Lists packages that are currently installed.
-q	Performs a package query; additional options can be used to fine-tune the query.
-f	Determines which package a specific file belongs to.

Use the **-q** option to the **rpm** command to perform queries. Table 6.8 describes some additional options for fine-tuning a query.

TABLE 6.8 **rpm Query Options**

Option	Description
-a	Returns a list of all installed packages.
-c	Lists the configuration files installed with the specified package.
-d	Lists the documentation files installed with the specified package.
-i	Displays information about the specified package.
-K	Verifies the integrity of the specified package.
-l	Lists all files installed with the specified package.
-provides	Lists which capabilities the specified package provides.
-R	Lists which capabilities the specified package requires.
-s	Displays the state of each file that was installed by the specified package (normal, not installed, or replaced).

Example:

```
$ rpm -qc cups
/etc/cups/classes.conf
/etc/cups/client.conf
/etc/cups/cups-files.conf
/etc/cups/cupsd.conf
/etc/cups/lpoptions
/etc/cups/printers.conf
/etc/cups/snmp.conf
/etc/cups/subscriptions.conf
/etc/dbus-1/system.d/cups.conf
/etc/logrotate.d/cups
/etc/pam.d/cups
```

dpkg

Use the **dpkg** command to manage local Debian packages.

Syntax:

```
dpkg [option] command
```

Table 6.9 describes some useful **dpkg** options.

TABLE 6.9 **dpkg Command Options**

Option	Description
-i	Installs a package.
-r	Removes the package but keeps the configuration files.
-P	Removes the package, including the configuration files (purge).
-l	Lists the packages that are currently installed.
-L	Lists files that were installed with a package (for example, **dpkg -L zip**).
-V	Verifies the integrity of the specified package or packages.
-s	Displays package status.
-C	Checks for broken packages.
-S	Lists the name of the package that was responsible for a specific file being installed on the system (for example, **dpkg -S /usr/bin/zip**).

When a package is installed, it might run a configuration script as part of the installation process. To run this configuration script again at some point in the future, use the **dpkg-reconfigure** command. Although there are some options to this command, they are rarely used.

The syntax of the **dpkg-reconfigure** command is as follows:

```
dpkg-reconfigure [options] source packages
```

The following example reruns the **tzdata** configuration scripts:

```
dpkg-reconfigure tzdata
```

ZYpp

The ZYpp software package provides the zypper utility. The **zypper** utility is found on SUSE Linux. It is derived from the RPM software suite and works very similarly to **yum**. It has automatic dependency checking and uses repositories. Its command structure and options are almost identical to those of **yum**. Here's an example:

```
zypper install pkg_name
```

If you know how to use the **yum** command, then in most cases you can replace **yum** with **zypper**, and the command will work successfully.

Sandboxed Applications

One of the challenges in deploying applications on Linux is the many different distributions. It is very difficult for developers to properly test how well applications perform in all of the possible environments.

A sandbox can help solve this problem. A sandbox environment behaves the same, regardless of which Linux distribution it is placed on. A sandboxed application is any application that is installed within a sandbox environment.

ExamAlert

This topic covers some of the basics of popular Linux sandboxes. Keep in mind that the Linux+ XK0-005 exam won't ask you detailed questions about these environments.

snapd

The **snapd** daemon manages packages called *snaps*, which are downloaded from the Snap store maintained by Canonical (which also maintains Ubuntu Linux). In fact, Ubuntu Core is a Linux distribution that uses only snap applications.

Developers can use Snapcraft, a tool that allows developers to package their programs as snaps.

Flatpak

To use Flatpak, you first need to install the Flatpak package. This package is not available for all distributions, but most distributions have it available. Flatpak applications are installed from repositories that are called *remotes*.

The Flatpak applications are compiled for the Flatpak sandbox environment. These compiled applications typically include the required libraries, but libraries can also be shared between applications.

Flatpak applications are normally graphical applications designed for regular users. Note that Flatpak software can be larger than traditional RPM or DEP packages.

AppImage

AppImage uses a unique approach in that each AppImage package is fully self-contained. This means a single package file contains all of the software, including libraries. AppImage packages tend to be smaller than snapd or Flatpak packages.

System Updates

Packages can be grouped into two major categories: system packages or non-system packages. Think of system packages as any software than can affect the way the core operating system functions. Examples of system packages include:

- **iptables:** Software that provides firewall capabilities
- **libcrypt:** Software that provides encryption libraries
- **kernel:** Software that provides the kernel code

Non-system packages are packages that don't affect the core operating system functions. Think of applications that regular users will use but that aren't required for the operating system to function, including:

- Web browsers
- Email client programs
- Games

Updating non-system packages typically only requires the user to reset an application to start using the new version. Updating system packages might require more action by the system administrator. These system packages can also be broken into two categories: kernel and all others. Kernel updates are described in the "Kernel Updates" section that follows. All other system packages are described in the "Package Updates" section that concludes the chapter.

Kernel Updates

Kernel updates are different from other system updates in two ways:

- To use a new kernel, you need to reboot the operating system.
- When a kernel is installed, it doesn't replace the previous kernel. When a non-kernel software package is installed to upgrade a previous version, it replaces the original. Kernel packages are different because if the new kernel fails to boot the system, you want to be able to revert to the previous version.

Package Updates

One important aspect of system updates is that you might need to restart a service in order to have the updates take effect. This could be accomplished by a system reboot, but this technique is rarely needed as just restarting the service should be enough. In many cases, software packages have instructions built in to restart the software; however, as a system administrator, you are responsible for verifying that this has taken place.

Package updating is an important task that shouldn't be overlooked as many updates include security fixes. Failing to restart the software leaves your system vulnerable to attacks and other security issues.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. The _____ command is used to download software packages without installing the software.
   - ❍ **A. rpm**
   - ❍ **B. yumdownloader**
   - ❍ **C. dpkg**
   - ❍ **D.** None of these answers are correct.

2. Which of the following is the primary configuration file for the **yum** command?
   - ❍ **A. /etc/yum-config/yum.conf**
   - ❍ **B. /etc/yum**
   - ❍ **C. /etc/yum.conf**
   - ❍ **D. /etc/yum.config**

3. Which option to the **rpm** command updates a package if an older version of the package exists but does nothing if an older version does not exist?
   - ❍ **A. -i**
   - ❍ **B. -U**
   - ❍ **C. -e**
   - ❍ **D. -F**

4. You have a kernel package installed on your system, and you install a new one by using the **rpm -i** command. How many kernel packages should you end up with after running this command?
   - ❍ **A.** 1
   - ❍ **B.** 2
   - ❍ **C.** 3
   - ❍ **D.** None as this results in kernel corruption.

Cram Quiz Answers

1. **B.** The **yumdownloader** command is used to download software packages without installing the software. The **rpm** and **dpkg** commands don't download packages; you must do that manually.
2. **C.** The **/etc/yum.conf** file is the primary configuration file for **yum** commands. The other answers are not valid files.

3. **D.** The **-F** option updates a package if an older version of the package exists but does nothing if an older version does not exist. The **-i** option is used to install (not update) a package. The **-U** option updates a package if an older version of the package exists and installs from scratch if the older version does not exist. The **-e** option deletes a package.
4. **B.** When installing a kernel package with the **rpm -i** command, the old kernel is left on the system, and the new kernel is installed in a different, non-conflicting location. As a result, you should have two kernels: the old one and the new one.

This page intentionally left blank

CHAPTER 7

Manage Software Configurations

This chapter covers the following Linux+ XK0-005 exam objective:

- **1.7:** Given a scenario, manage software configurations.

In Chapter 6, "Build and Install Software," you learned about managing software. In this chapter the focus shifts toward software configuration. You will learn how to manage software repository configurations and how to configure the kernel. You will also learn how to configure common system services, such as SSH, NTP, and syslog.

This chapter provides information on the following topics: updating configuration files, configuring kernel options, configuring common system services, and localization.

Updating Configuration Files

This section explores the package management configuration files that you should know how to update.

Procedures

If you upgrade a package, you might need to restart or reload a service. Typically the documentation that comes with the software package should indicate if a restart or reload is required. You also might need to restart or reload a service if changes are made to the configuration file for the software. This section addresses these procedures.

Restart Service

The **restart** option is used with the **systemctl** command to restart a service that is currently running. A restart can make the service unavailable for a short period of time.

Syntax:

```
systemctl restart process_name
```

You can determine whether a service is currently running by using the **active** command:

```
# systemctl active cups
enabled
```

Reload Service

The **reload** option is used with the **systemctl** command to reload a service that is currently running. This command does not stop the service but rather has the service reread its configuration file and change its behavior based on changes in the configuration file.

Syntax:

```
systemctl reload process_name
```

You can determine whether a service is currently running by using the **active** command:

```
# systemctl active cups
enabled
```

.rpmnew

When RPMs are installed, they might overwrite the default configuration files of the software package as new configuration settings are added or older configuration settings are removed.

The overwriting of the configuration files can lead to frustration if the system administrator has spent time and effort creating a customized configuration file. As a result, software developers might choose to place the new configuration file contents in a file called **.rpmnew**. Then a system administrator can

review the **.rpmnew** file and determine which new settings should be incorporated into the primary configuration file.

Another technique is for the software developers to use a **.rpmsave** file, as described in the next section.

.rpmsave

When RPMs are installed, they might overwrite the default configuration files of the software package as new configuration settings are added or older configuration settings are removed.

The overwriting of the configuration files can lead to frustration if the system administrator has spent time and effort creating a customized configuration file. As a result, software developers might choose to place the previous configuration file contents in a file called **.rpmsave**. Then a system administrator can review the **.rpmsave** file and determine which of the previous settings should be incorporated into the primary configuration file.

Another technique is for the software developers to use a **.rpmnew** file, as described in the previous section.

ExamAlert

Be ready to be tested on the difference between the **.rpmnew** and **.rpmsave** files.

Repository Configuration Files

Repository configuration files, discussed in Chapter 6, are an integral component of package management. The Linux+ XK0-005 exam lists these topics in different exam categories, so the following section headings have been preserved to inform you where to go to find details about these topics.

/etc/apt.conf

See the "APT" section in Chapter 6.

/etc/yum.conf

See the "YUM" section in Chapter 6.

/etc/dnf/dnf.conf

See the "DNF" section in Chapter 6.

/etc/yum.repo.d

See the "YUM" section in Chapter 6.

/etc/apt/sources.list.d

See the "APT" section in Chapter 6.

Configure Kernel Options

The Linux kernel is highly configurable. You can make changes to the kernel by using parameters and kernel modules. This section covers these configuration features.

Parameters

A kernel parameter is a value that changes the behavior of the kernel.

sysctl

You can view and change parameters by using the **sysctl** command. For example, to view all the kernel and kernel module parameters, execute the **sysctl** command with the **-a** option:

```
[root@onecoursesource ~]# sysctl -a | head
kernel.sched_child_runs_first = 0
kernel.sched_min_granularity_ns = 1000000
kernel.sched_latency_ns = 5000000
kernel.sched_wakeup_granularity_ns = 1000000
kernel.sched_tunable_scaling = 1
kernel.sched_features = 3183
kernel.sched_migration_cost = 500000
kernel.sched_nr_migrate = 32
kernel.sched_time_avg = 1000
kernel.sched_shares_window = 10000000
```

ExamAlert

You are not expected to know details of any specific configuration for the Linux+ XK0-005 exam.

The name of the parameter (**kernel.sched_child_runs_first**, for example) is a relative pathname that starts from **/proc/sys** and has a dot (**.**) character between the directory and filename rather than a slash (**/**) character. For example, the **/proc/sys/dev/cdrom/lock** file is named using the **dev.cdrom.lock** parameter:

```
[root@onecoursesource ~]# sysctl -a | grep dev.cdrom.lock
dev.cdrom.lock = 1
```

You can change the value of this parameter by using the **sysctl** command:

```
[root@onecoursesource ~]# sysctl dev.cdrom.lock=0
dev.cdrom.lock = 0
[root@onecoursesource ~]# sysctl -a | grep dev.cdrom.lock
dev.cdrom.lock = 0
```

It is actually safer to use the **sysctl** command than to modify the file directly because the **sysctl** command knows which values for the parameter are valid and which ones are not:

```
[root@onecoursesource ~]# sysctl dev.cdrom.lock="abc"
error: "Invalid argument" setting key "dev.cdrom.lock"
```

The **sysctl** command knows which parameter values are valid because it can look at the **modinfo** output. For example, the value of the **lock** file must be a Boolean (0 or 1), according to the output of the **modinfo** command:

```
[root@onecoursesource ~]# modinfo cdrom | grep lock
parm:           lockdoor:bool
```

If you modify the file directly or use the **sysctl** command, the changes are temporary. When the system is rebooted, the values go back to the defaults, unless you make changes in the **/etc/sysctl.conf** file. The next section provides more details about **/etc/sysctl.conf**.

/etc/sysctl.conf

The **/etc/sysctl.conf** file is used to specify which kernel parameters to enable at boot.

Example:

```
[root@OCS ~]# cat /etc/sysctl.conf
# System default settings live in/usr/lib/sysctl.d/00-system.conf.
# To override those settings, enter new settings here, or
# in an /etc/sysctl.d/<name>.conf file.
#
# For more information, see sysctl.conf(5) and sysctl.d(5).
net.ipv4.ip_forward=1
```

In this example, the kernel parameter **ip_forward** is turned on, which means this machine will act as a router between two networks.

There are thousands of possible kernel settings, including dozens of settings that affect networking. The **ip_forward** setting is one of the most common network settings.

The parameters that optimize the IO (input/output) scheduler are examples of kernel parameters. Several parameters can be set to change the behavior of the scheduler. This section covers the parameters that are important for the Linux+ XK0-005exam.

To see the current scheduler, view the contents of the **/sys/block/**<*device*>**/queue/scheduler** file (where <*device*> is the actual device name). Here's an example:

```
[root@OCS ~]# cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
```

The value within the square brackets is the default. To change this, use the **echo** command, as shown here:

```
[root@OCS ~]# echo "cfq" > /sys/block/sda/queue/scheduler
[root@OCS ~]# cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
```

Additional scheduler types include the following:

- **cfq:** The Completely Fair Queuing schedule has a separate queue for each process, and the queues are served in a continuous loop.
- **noop:** This schedule follows the FIFO (first in, first out) principle.
- **deadline:** This is the standard scheduler. This scheduler creates two queues: a read queue and a write queue. It also puts a timestamp on each I/O request to ensure that requests are handled in a timely manner.

Modules

A *module* is a small software program that, when loaded, provides more features and capabilities to the kernel. This section describes the management of modules using the **lsmod**, **imsmod**, **rmmod**, **insmod**, **modprobe**, and **modinfo** commands.

lsmod

The **lsmod** command displays the kernel modules that are loaded into memory. This command has no options.

In the output of the **lsmod** command, each line describes one module. There are three columns of information for each line:

- The module name.
- The size of the module, in bytes.
- The "things" that are using the module. A "thing" could be a filesystem, a process, or another module. In the event that another module is using this module, the dependent module name is listed. Otherwise, a numeric value that indicates how many "things" use this module is provided.

Example:

```
[root@OCS ~]# lsmod | head
Module                  Size       Used by
tcp_lp                  12663      0
bnep                    19704      2
```

```
bluetooth                372944     5   bnep
rfkill                   26536      3   bluetooth
fuse                     87741      3
xt_CHECKSUM              12549      1
ipt_MASQUERADE           12678      3
nf_nat_masquerade_ipv4   13412       1   ipt_MASQUERADE
tun                       27141 1
```

imsmod

The **imsmod** command is used to add modules to the currently running kernel.

Syntax:

```
insmod [module_name]
```

The exact location of the module needs to be specified. For example:

```
[root@OCS ~]# lsmod | grep fat
[root@OCS ~]# insmod /usr/lib/modules/3.19.8-100.fc20.x86_64/kernel/
fs/ fat.ko
[root@OCS ~]# lsmod | grep fat
fat                    65107 0
```

There are no options to the **insmod** command; however, each module might have modules that can be passed into the module using the following syntax:

```
insmod module options
```

The **insmod** command has two disadvantages:

- You have to know the exact location of the module.
- If the module has any dependencies (that is, if the module needs another module), it will fail to load.

rmmod

The **rmmod** command is used to remove modules from the currently running kernel.

Syntax:

```
rmmod [options] [module_name]
```

Example:

```
[root@OCS ~]# lsmod | grep fat
fat   65107 0
[root@OCS ~]# rmmmod fat
[root@OCS ~]# lsmod | grep fat
```

Modules that are currently in use will not be removed by this command by default.

Key options for the **rm** command include the following:

- **-f** attempts to force removal of modules that are in use (which is very dangerous).
- **-w** waits for a module to be no longer used and then removes it.
- **-v** displays verbose messages.

insmod

The **insmod** command is used to add modules to the currently running kernel.

Syntax:

```
insmod [module_name]
```

The exact location of the module needs to be specified. For example:

```
[root@OCS ~]# lsmod | grep fat
[root@OCS ~]# insmod /usr/lib/modules/3.19.8-100.fc20.x86_64/kernel/
fs/ fat.ko
[root@OCS ~]# lsmod | grep fat
fat                    65107 0
```

There are no options to the **insmod** command; however, each module might have modules that can be passed into the module using the following syntax:

```
insmod module options
```

The **insmod** command has two disadvantages:

- You have to know the exact location of the module.
- If the module has any dependencies (that is, if the module needs another module), it will fail to load.

modprobe

The **modprobe** command is used to add and remove modules from the currently running kernel. It also attempts to load module dependencies.

Syntax:

```
modprobe [options] [module_name]
```

When used to remove modules (with the **-r** option), the **modprobe** command also removes dependency modules unless they are in use by another part of the subsystem (such as the kernel or a process).

Key options for the **modprobe** command include the following:

- **-c** displays the current **modprobe** configuration.
- **-q** causes **modprobe** to run in quiet mode.
- **-R** displays all modules that match an alias to assist you in debugging issues.
- **-r** removes the specified module from memory.
- **-v** displays verbose messages; this is useful for determining how **modprobe** is performing a task.

modinfo

The **modinfo** command is used to provide details about a module.

Syntax:

```
modinfo [module_name]
```

Example:

```
[root@OCS ~]# modinfo xen_wdt
filename: /lib/modules/3.19.8-100.fc20.x86_64/kernel/drivers/watchdog/
xen_wdt.ko
license: GPL
version: 0.01
description:   Xen WatchDog Timer Driver
author: Jan Beulich <jbeulich@novell.com>
srcversion:   D13298694740A00FF311BD0
depends:
intree:     Y
```

```
vermagic:    3.19.8-100.fc20.x86_64 SMP mod_unload

signer:       Fedora kernel signing key

sig_key:     06:AF:36:EB:7B:28:A5:AD:E9:0B:02:1E:17:E6:AA:B2:B6:52:
63:AA

sig_hashalgo:   sha256

parm:           timeout:Watchdog timeout in seconds (default=60)(uint)

parm:            nowayout:Watchdog cannot be stopped once started
(default=0) (bool)
```

One of the most important parts of the output of the **modinfo** command is the **parm** values, which describe parameters that can be passed to this module to affect its behavior.

Configure Common System Services

This section focuses on configuring common system services, including SSH, NTP, syslog, chrony, and localization.

SSH

The Secure Shell (SSH) protocol is designed to replace insecure remote communication operations, such as the **telnet**, **ftp**, **rlogin**, **rsh**, **rcp**, and **rexec** commands/protocols. The primary issue with earlier communication methods is that those methods send data across the network in plaintext rather than in an encrypted format. In some cases, such as with **telnet** and **ftp**, this can include sending user account data (such as name and password) across the network in plaintext.

SSH provides a better level of security by encrypting the data sent across the network. SSH has become such a standard in Linux that almost all distributions include both the client and server software by default. In the event that you do not have this software installed on your system, you should install the **openssh**, **openssh-server**, **openssh-clients**, and **openssh-askpass** software packages.

The **/etc/ssh** directory is the location where the Secure Shell configuration files are stored. The configuration file for the SSH server is the **/etc/ssh/sshd_config** file. Don't confuse this with the **/etc/ssh/ssh_config** file, which is used to configure client utilities, such as the **ssh**, **scp**, and **sftp** commands.

There are two different SSH protocols that are numbered 1 and 2. These are not versions but rather two separate protocols developed to provide secure data connections. There was a time when both protocols were commonly used, but

now almost all SSH clients use only protocol 2. To set the protocol that your SSH server accepts, use the **Protocol** keyword:

```
Protocol 2
```

If you have some older SSH clients that require protocol 1, you can configure your SSH server to accept both protocol connections by using the following keyword setting in the **/etc/ssh/sshd_config** file:

```
Protocol 1,2
```

If you have multiple network cards (or virtual interfaces), you may want to limit the SSH server to listen to only some of the network cards. To do this, use the **ListenAddress** keyword and specify the IP address assigned to the network cards that SSH should accept connections on:

```
ListenAddress 192.168.1.100:192.168.1.101
```

The standard port number that the SSH server listens to is port 22. You can modify the SSH server to listen to another port by using the **Port** keyword:

```
Port 2096
```

You might need to change what sort of log messages you want the SSH server to record. This can be set by using the **LogLevel** keyword. The levels available are as follows:

- **QUIET**
- **FATAL**
- **ERROR**
- **INFO**
- **VERBOSE**
- **DEBUG**
- **DEBUG1** (which is the same as **DEBUG**)
- **DEBUG2**
- **DEBUG3**

Network Time Protocol (NTP)

The Network Time Protocol daemon (**ntpd**) is a process that ensures the system clock is in sync with the time provided by remote NTP servers. Most of

the configuration for this process is handled via the **/etc/ntp.conf** file. Table 7.1 shows the important settings of the **/etc/ntp.conf** file.

TABLE 7.1 **/etc/ntp.conf File Settings**

Option	Description
driftfile	Contains a value that represents the typical delta (change) over time from the NTP-reported time and the system clock. This value is used to regularly update the system clock without having to access an NTP server.
restrict	Used to indicate restrictions for the daemon, including what machines can access this NTP server when it is used as a service.
server	Used to list an NTP server for this machine when it is used as an NTP client.

Here is an example of a typical **/etc/ntp.conf** file:

```
# For more information about this file, see the man pages
# ntp.conf(5), ntp_acc(5), ntp_auth(5), ntp_clock(5), ntp_misc(5),
ntp_mon(5).

driftfile /var/lib/ntp/drift

# Permit time synchronization with our time source, but do not
# permit the source to query or modify the service on this system.
restrict default kod nomodify notrap nopeer noquery

# Permit all access over the loopback interface. This could
# be tightened as well, but to do so would effect some of
# the administrative functions.
restrict 127.0.0.1
restrict ::1

# Hosts on local network are less restricted.
# restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap

# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.
  html).
server 0.fedora.pool.ntp.org iburst
server 1.fedora.pool.ntp.org iburst
```

```
server 2.fedora.pool.ntp.org iburst
server 3.fedora.pool.ntp.org iburst

# Enable public key cryptography.
#crypto

includefile /etc/ntp/crypto/pw

# Key file containing the keys and key identifiers used when operating
# with symmetric key cryptography.
keys /etc/ntp/keys
```

The **pool.ntp.org** address is a link to a cluster of NTP servers that are geographically spread throughout the world. These servers can be freely used within the **/etc/ntp.conf** file. For example, the following servers are provided by the Fedora project (but note that these are often mirrors, pointing to other systems, so the resulting hostnames for these servers will be different once you have connected to them):

0.fedora.pool.ntp.org

1.fedora.pool.ntp.org

2.fedora.pool.ntp.org

3.fedora.pool.ntp.org

The **ntpq** command allows you to perform queries on NTP servers. For example, the **ntpq** command in the following example displays a summary of the status of NTP servers:

```
[root@onecoursesource ~]# ntpq -p
     remote           refid      st t when poll reach   delay   offset
jitter
==============================================================
*propjet.latt.ne 68.110.9.223    2 u  120 1024  377   98.580    7.067
4.413
-services.quadra 208.75.88.4      3 u  272 1024  377   72.504  -10.689
1.612
+mirror          216.93.242.12    3 u  287 1024  377   20.406   -2.555
0.822
+108.61.194.85.v 200.23.51.102    2 u  741 1024  377   69.403   -3.670
1.610
```

Table 7.2 lists some important options to the **ntpq** command.

TABLE 7.2 **ntpq Command Options**

Option	Description
-d	Enables debugging mode.
-n	Lists host IP addresses rather than names.
-p	Prints a list of all peers.

Syslog

The syslog service has existed since 1980. Although it was advanced at the time it was created, its limitations have grown over time as more complex logging techniques have become required.

In the mid-2000s, the rsyslog service was created as an extension of the traditional syslog service. The rsyslog service extends the capabilities of syslog through the inclusion of modules.

The configuration of syslog and rsyslog services is consistent, with the exception of slightly different naming conventions (for example, **rsyslog.conf** versus **syslog.conf**) and additional features available in the log files.

The **syslogd** or **rsyslogd** daemon is responsible for logging of application and system events. It determines which events to log and where to place log entries, based on configuration settings in the **/etc/syslog.conf** file.

Table 7.3 describes some important options to the **syslogd** and **rsyslogd** commands.

TABLE 7.3 **syslogd and rsyslogd Command Options**

Option	Description
-d	Enables debugging mode.
-f	Specifies the configuration file (with **/etc/syslog.conf** as the default).
-m *x*	Creates a timestamp in the log files every *x* minutes. (You can set *x* to **0** to omit timestamps.)
-r	Enables the **syslogd** daemon to accept logs from remote systems.
-S	Enables verbose mode.
-x	Disables DNS lookups for IP addresses.

The **/etc/rsyslog.conf** file is one of the configuration files for the **rsyslogd** daemon. The following is a typical **rsyslog.conf** file with the comments and blank lines removed (along with the modules):

```
[root@OCS ~]# grep -v "^$" /etc/rsyslog.conf | grep -v "^#"
*.info;mail.none;authpriv.none;cron.none /var/log/messages
authpriv.*                               /var/log/secure
mail.*                                   -/var/log/maillog
cron.*                                    /var/log/cron
*.emerg                                    *
uucp,news.crit                            /var/log/spooler
local7.*                                 /var/log/boot.log
```

ExamAlert

On the Linux+ XK0-005 exam, you might be given an entry line from the **/etc/rsyslog.conf** file and be tested on your understanding of what is logged and where.

Every line represents one logging rule that is broken into two primary parts: the selector (for example, **uucp,news.crit**) and the action (**/var/log/spooler**). The selector is also broken into two parts: the facility (**uucp,news**) and the priority (**crit**). Note that when a priority is provided, it means "this priority and all priorities of a higher level."

The following list shows the available facilities in order from lower level to higher level:

- **auth** (or security)
- **authpriv**
- **cron**
- **daemon**
- **kern**
- **lpr**
- **mail**
- **mark**
- **news**
- **syslog**

- **user**
- **uucp**
- **local0** through **local7**

The following list shows the available priority levels:

- **debug**
- **info**
- **notice**
- **warning** (or **warn**)
- **err** (or **error**)
- **crit**
- **alert**
- **emerg** (or **panic**)

The following list shows the available "actions", which are really just where the log entry should be sent:

- Regular file (where using **-** before the filename prevents syncing with every log entry, thus reducing hard drive writes)
- Named pipes
- Console or terminal devices
- Remote hosts
- Users, which write to the specified user's terminal windows (where ***** specifies all users)

chrony

Traditionally, the NTP server on Linux has been the **ntpd** server (discussed earlier in this chapter, in the "Network Time Protocol [NTP]" section). A newer alternative, chrony, provides some benefits over the **ntpd** server, including:

- Faster synchronization
- Typically more accurate time

- Improved response to clock frequency changes
- No need for periodic polling of other NTP servers
- Smaller (less memory and CPU utilization)

Localization

Several commands can be used to display or modify the date and time on a system, as well as the locale of a system. This section explores these commands.

timedatectl

Use the **timedatectl** command to display the system clock.

Syntax:

```
timedatectl [option] [value]
```

Example:

```
[root@OCS ~]# timedatectl
       Local time   :       Wed 2018-10-10 14:41:41 PDT
       Universal time:      Wed 2018-10-10 21:41:41 UTC
       RTC time:            Wed 2018-10-10 09:51:09
      Timezone:             America/Los_Angeles (PDT, -0700)
      NTP enabled:          yes
       NTP synchronized:     yes
       RTC in local TZ:      no
       DST active:           yes
        Last DST change:    DST began at
                            Sun 2018-03-11 01:59:59 PST
                            Sun 2018-03-11 03:00:00 PDT
      Next DST change:  DST ends (the clock jumps one hour backwards)
at
                            Sun 2018-11-04 01:59:59 PDT
                             Sun 2018-11-04 01:00:00 PST
```

As the root user, you can use this command to set the system clock. Table 7.4 demonstrates the most commonly used methods of changing the system clock.

TABLE 7.4 **Methods to Change the System Clock**

Method	Description
set-time [*time*]	Sets the system clock to the specified *time*.
set-timezone [*zone*]	Sets the system time zone to the specified *zone*.
set-ntp [0|1]	Enables (**1**) or disables (**0**) Network Time Protocol.

localectl

The BASH shell and other processes need customized operations to fit the location of the user. For example, if currency is to be displayed and the user is located in the United States, the $ character should be used. If the user is located in Great Britain, the £ character should be used.

This section focuses on the variables used to inform programs what settings to use based on a user's locale.

LC_* refers to a collection of locale settings used to change the way the shell and other programs handle differences based on the geographic region of the user (or a region the user is familiar with). These values can be viewed by executing the **locale** command:

```
[root@OCS ~]# locale
LANG=en_US.UTF-8
LANGUAGE=en_US
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

The most important locale settings are described in Table 7.5.

TABLE 7.5 **Locale Settings**

Setting	Description
LANG	Language
LC_CTYPE	Case conversion
LC_NUMERIC	Numeric format
LC_TIME	Time and date format
LC_COLLATE	Collation order
LC_MONETARY	Currency format
LC_MESSAGES	Format of message
LC_PAPER	Paper size format
LC_NAME	Name format
LC_ADDRESS	Address format
LC_TELEPHONE	Telephone format
LC_ALL	When set, LC_ALL will override all other locale settings. This provides an easy means to change all locale settings by modifying one environment variable.

The **localectl** command can display and change both locale values and keyboard layouts.

Syntax:

```
localectl [options] command
```

To display values, use **status**:

```
[root@OCS ~]# localectl status
 System Locale:  LANG=en_US.utf8
    VC Keymap:   us
    X11 Layout:   us
    X11 Model:    pc105+inet
     X11 Options: terminate:ctrl_alt_bksp
```

Set the locale and keyboard as follows:

```
[root@OCS ~]# localectl set-locale "LANG=de_DE.utf8"set-keymap "de"
```

There are a handful of options to the **localectl** command, but none of them are commonly used.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which file can be revised to modify kernel parameters during the boot process?
   - ❍ **A. /etc/sysctl.conf**
   - ❍ **B. /etc/kern.parm**
   - ❍ **C. /etc/kern.conf**
   - ❍ **D. /etc/sys.conf**

2. Which command correctly sets the CD-ROM lock kernel parameter to 0?
   - ❍ **A. sysctl dev.cdrom.lock 0**
   - ❍ **B. sysctl dev.cdrom.lock=0**
   - ❍ **C. sysctl dev/cdrom/lock=0**
   - ❍ **D. sysctl dev/cdrom/lock 0**

3. Which of the following commands can be used to remove a module from memory? (Choose two.)
   - ❍ **A. modprobe**
   - ❍ **B. insmod**
   - ❍ **C. modinfo**
   - ❍ **D. rmmod**

4. Based on the following line from the **/etc/rsyslog.conf** file, which levels of messages for the **cron** service are logged?

```
cron.warn                                    /var/log/cron
```

   - ❍ **A.** All levels
   - ❍ **B.** Just warn level
   - ❍ **C.** Warn level and levels with higher priorities
   - ❍ **D.** Warn level and levels with lower priorities

Cram Quiz Answers

1. **A.** The **/etc/sysctl.conf** file is used to specify which kernel parameters to enable at boot. The other answers are not valid kernel configuration files.
2. **B.** You can change the value of this parameter by using the **sysctl** command: **sysctl dev.cdrom.lock=0**. The character between **dev**, **cdrom**, and **lock** must be a **.**, not a **/** character. An equal sign must be between the parameter and the value.
3. **A and D.** The **rmmod** command is designed specifically to remove a module from memory. The **-r** option to the **modprobe** command also can be used to remove a module from memory. The **insmod** command inserts a module into memory, and the **modinfo** command provides details about a module.
4. **C.** When a priority is provided, it means “this priority and all priorities of a higher level.”

CHAPTER 8

Security Best Practices in a Linux Environment

This chapter covers the following Linux+ XK0-005 exam objective:

- **2.1:** Summarize the purpose and use of security best practices in a Linux environment.

System administrators are typically responsible for the security of the systems they manage. Operating system security is a huge topic, with entire books devoted to the topic. You will learn different security best practices throughout this book, but this chapter is designed to cover topics that are addressed specifically on the Linux+ XK0-005 exam.

You will learn about public key infrastructure (PKI) and how certificates are used to enhance security. You will also learn about different factors related to authentication, including MFS and PAM, as well as authentication systems like Lightweight Directory Access Protocol (LDAP) and System Security Services Daemon (SSSD). The chapter concludes with coverage of a variety of topics related to Linux hardening, which is the process of making a Linux operating system more secure.

This chapter provides information on the following topics: managing PKI certificates, certificate use cases, authentication, and Linux hardening.

Managing Public Key Infrastructure (PKI) Certificates

When a user attempts to connect to a server, such as a web server, public key infrastructure (PKI) is used to ensure that the server is really where the user intended to go rather than a rogue server. Often this structure also provides the means to encrypt data between the server and the user.

The technique used by TLS (see the section "Secure Sockets Layer (SSL)/ Transport Layer Security (TLS)," later in this chapter) is called *asymmetric cryptography* (and is also referred to as public key cryptography [PKC]). With asymmetric cryptography, two keys are used: a *public key* and a *private key*. These keys are used to encrypt and decrypt data.

The public key is used to encrypt the data sent to the Apache web server. This key is provided to all systems upon request. For example, when a web browser first connects to an Apache web server that is using TLS, the web browser asks for the public key of the server. This public key is freely given to the web browser.

Any additional data sent from the web browser to the server, such as account names and passwords, is encrypted using this public key. This data can only be decrypted by the server using the private key in a process called *hashing*, which involves converting data into a different value using a key. This is *asymmetric cryptography* because different keys are used for encryption and decryption. This means that only the web server can decrypt the data sent by the web browser. With *symmetric cryptography*, in contrast, the same key is used to encrypt and decrypt data.

ExamAlert

Before you take the Linux+ XK0-005 exam, be sure you understand how public and private keys are different from one another.

You might be wondering how the web browser really knows that it reached the correct web server and not some "rogue" web server. When the web server sends its public key, it includes a *digital signature* (also called a *message digest*). This digital signature can be sent to a CA (certificate authority) server, a trusted third-party system used to verify the digital signature. In some cases, the server itself provides the signature; this is called a *self-signed* certificate.

From the user's perspective, all this takes place behind the scenes and is completely transparent—at least until something goes wrong and warning messages are displayed by the web browser. Digital certificates typically have expiration dates. In some cases, a CA does not have an updated digital certificate or the user changes the date on the computer. Such issues can cause problems.

Figure 8.1 provides a visual example of the steps in the SSL process.

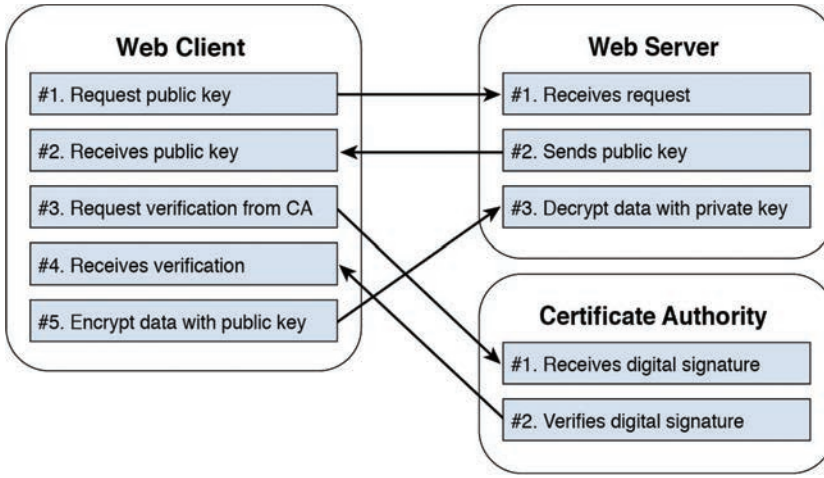

FIGURE 8.1 **SSL Process**

Public Key

See the "Managing Public Key Infrastructure (PKI) Certificates" section, earlier in this chapter.

Private Key

See the "Managing Public Key Infrastructure (PKI) Certificates" section, earlier in this chapter.

Self-Signed Certificate

See the "Managing Public Key Infrastructure (PKI) Certificates" section, earlier in this chapter.

Digital Signature

See the "Managing Public Key Infrastructure (PKI) Certificates" section, earlier in this chapter.

Wildcard Certificate

Typically, a certificate is associated with a single system, based on the system hostname. In an organization that has many systems that require certificates, having a certificate for each system can be costly to implement.

A wildcard certificate allows you to have a certificate that applies to an entire domain, including the subdomains for that domain. In other words, you could have one certificate that can be associated with all of the following systems:

- onecoursesource.com
- www.onecoursesource.com
- test.onecoursesource.com
- sales.onecoursesource.com
- dev.sales.onecoursesource.com
- private.sales.onecoursesource.com

Hashing

See the "Managing Public Key Infrastructure (PKI) Certificates" section, earlier in this chapter.

Certificate Authorities

IT security revolves around the concept of trust. For example, suppose you want to connect to your bank online and you type the bank's URL in a browser. Your browser gets a response from a remote server, but how can you ensure that remote server is actually your bank and not a server that hijacked your connection attempt?

The solution to this is digital certifications. When the browser connects to the server, the server issues a signature that is tied to a digital certificate. The web browser consults a third-party organization called a certificate authority (CA) to verify the signature. The trust is built between the client system and the server by the CA's verification.

Certificate Use Cases

For the Linux+ exam you should be aware of the situations in which certificates are used. This section describes some common use cases.

Secure Sockets Layer (SSL)/Transport Layer Security (TLS)

Transport Layer Security (TLS) is a cryptographic protocol that is used by VPNs to provide secure transport of data. Secure Sockets Layer (SSL) is the predecessor to TLS and is considered deprecated.

TLS isn't just used for VPNs; it is also commonly used for web server communications, email transport, and voice over IP (VoIP).

Certificate Authentication

Certificate authentication is the process of using a public key certificate to allow one system to verify the identity of another system. It is commonly used by web clients to ensure that a web server is not a rogue system.

Encryption

See the "Managing Public Key Infrastructure (PKI) Certificates" section, earlier in this chapter, for information about the role of encryption.

Authentication

The standard authentication method is to use local user accounts and passwords. This method lacks some of the security features that more complex methods provide. This section explores alternative authentication methods that should be considered when developing a security best practice policy.

Tokens

A token is a unique value (typically either a number or an alphanumeric value) that is generated by either a hardware device or a software program. Tokens are typically automatically generated on an ongoing basis and normally are valid for only short periods of time, such as 30 seconds.

A hardware token is a token that is generated by a hardware device. Typically it is a very small device, such as a key fob. The hardware device may have additional authentication methods, such as a fingerprint scanner, but this tends to be somewhat rare.

A software token is a token that is generated by a program. A common example is an app on a mobile device.

Multifactor Authentication (MFA)

With multifactor authentication (MFA), the user is requested to provide multiple bits of evidence that prove the user's identity. One common multifactor authentication method is called two-factor authentication, or 2FA. For this method, the user is required to provide two forms of identification, which could include the following:

- Something the user knows
- Something the user has
- Something the user "is"

To utilize MFA, you can implement a number of different features, including tokens, one-time passwords (OTPs), and biometrics (fingerprint scanner, iris scanner, and so on).

Pluggable Authentication Modules (PAM)

Pluggable authentication modules (PAM) are already used by almost all the Linux utilities that attempt to authenticate users. This is important because you don't need to make adjustments to the utilities to change how users are authenticated; rather, you make changes to the PAM configuration files.

To understand how beneficial PAM is, you should realize some of the things you can have PAM do. Here are a few of the many examples of what you can do with PAM:

- You can use PAM to enforce more robust password requirements.
- You can limit what days and times users can log in to the system with PAM.
- You can use PAM to limit where users log in from.

- You can use PAM to set or unset environment variables. Because you can have different PAM rules for different utilities, this means you would have one set of environment variables for local login, one set for SSH logins, and one for FTP logins.
- User accounts can have restrictions placed on them with PAM. For example, the number of processes a user can execute could be limited.
- You can limit where the root user can log in from. This can make your system more secure by only allowing the root user to log in locally or use the **su** command to gain access to the root account.

Keep in mind that these are just a few of the things you can do with PAM. Dozens of different PAM modules provide many different authentication features.

ExamAlert

PAM can be a complicated system. Remember that the goal of this book is to provide a summary, not a complete description, of each topic covered on the Linux+ XK0-005 exam. This section describes what you need to know about PAM for the exam.

To understand what the contents of a PAM configuration file mean, you first have to understand the first column of data. There are four possible types in the first column: **auth**, **account**, **session**, and **password**. Figure 8.2 visually describes how they are used.

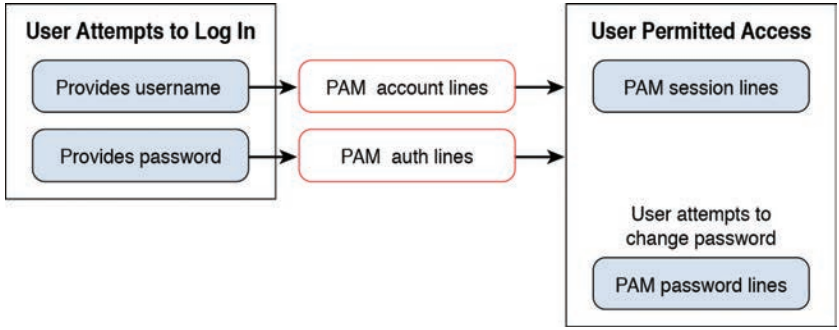

FIGURE 8.2 **PAM Types**

The first value of each line in a PAM configuration file, the type, basically indicates when to use the module. The last value of each line is the PAM module to run (or another configuration file to consult). The middle value is called the control value. When the PAM module returns successful or unsuccessful, how this return value is used depends on the control value.

Table 8.1 describes the different traditional control values. When you're reading the descriptions in this table, it is important to note that when a set of rules is parsed, the end result must be successful for the action to take place. For example, consider the following rules:

```
account required pam_nologin.so
account include system-auth
```

For the user account to be validated, the end result of the chain of rules must be successful. This chain of rules is also referred to as the *rule stack*.

TABLE 8.1 **Traditional Control Values**

Value	Description
required	**Returns successful:** If the current PAM returns successful, the next rule in the list is checked. If this is the final rule, the stack returns with a value of successful (except when a previous requisite rule is unsuccessful). **Returns unsuccessful:** If the current PAM returns unsuccessful, no additional modules are executed. The stack returns with a value of unsuccessful. In other words, the modules run with the required value must return success for the stack to return success.
requisite	In some ways, the **requisite** value is like **required**. The difference is highlighted below in italics. **Returns successful:** If the current PAM returns successful, the next rule in the list is checked. If this is the final rule, the stack returns with a value of successful. **Returns unsuccessful:** If the current PAM returns unsuccessful, *additional modules are executed; however, regardless of what happens with these other modules, the stack returns with a value of unsuccessful*. This is useful sometimes because the additional modules may provide more information in the log files.

Value	Description
optional	**Returns successful:** If the current PAM returns successful, the next rule in the list is checked. If this is the final rule, the stack returns with a value of successful (except when a previous requisite rule is unsuccessful). **Returns unsuccessful:** If the current PAM returns unsuccessful, the next rule in the list is checked. If this is the final rule, the stack returns with a value of successful (except when a previous requisite rule is unsuccessful). In other words, the return value of the module has no impact on the success or failure of the stack. Exception: If this is the only rule in the stack, **optional** acts like **required**.
sufficient	**Returns successful:** If the current PAM returns successful, then no additional rules are checked. The stack immediately returns with a value of successful (except when a previous requisite rule is unsuccessful). **Returns unsuccessful:** If the current PAM returns unsuccessful, the next rule in the list is checked. If this is the final rule, the stack returns with a value of successful (except when a previous requisite rule is unsuccessful). In other words, if the return value of the module is successful, no other modules are checked, and the stack is automatically successful. A return of unsuccessful has no impact on the overall return value of the stack.
include	This value tells PAM to use all the rules from the file specified. Only the rules for the current type (**auth**, **account**, and so on) are used.

Figure 8.3 shows a flowchart diagram for those who like more visual examples. Note that this figure makes use of the following information:

```
[root@OCS ~]# grep ^account /etc/pam.d/sshd
account required pam_nologin.so
account include system-auth
[root@OCS ~]# grep ^account /etc/pam.d/system-auth
account required pam_unix.so
account sufficient pam_succeed_if.so uid < 500 quiet
account required pam_permit.so
```

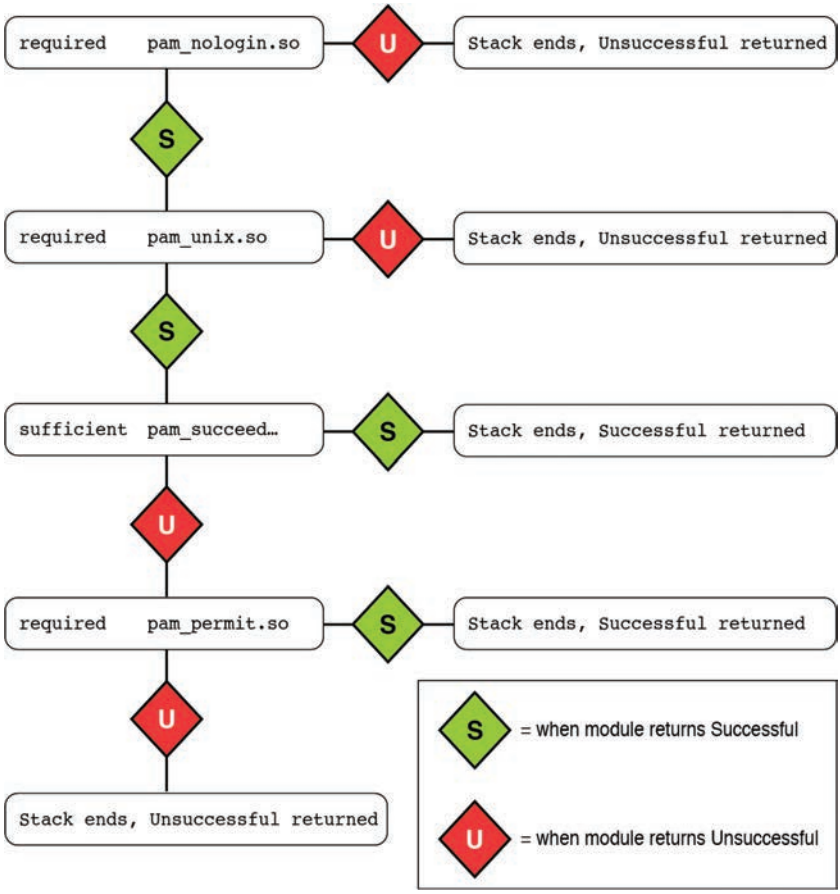

FIGURE 8.3 **PAM Control Values Example**

System Security Services Daemon (SSSD)

The System Security Services Daemon (SSSD) is designed to provide an authenticated method to access remote directories. It uses an authentication service, like LDAP, to create a local cache of user accounts on the client system. These accounts are used to access the remote directories.

ExamAlert

Configuration of SSSD is beyond the scope of the Linux+ XK0-005 exam. For the exam you should be aware of the function of this daemon.

Lightweight Directory Access Protocol (LDAP)

Lightweight Directory Access Protocol (LDAP) is often compared to RADIUS and TACACS+, so it is good to understand those technologies. Remote Authentication Dial-In Service (RADIUS) is a protocol that allows a client system to make use of a server to authenticate users. This service provides AAA (authentication, authorization, and accounting) management in a centralized location.

RADIUS is often compared to TACACS+, but knowing the differences between these protocols is beyond the scope of the Linux+ exam. Most importantly, realize that RADIUS can provide very powerful functions within a security policy because it offers centralized AAA management. Also, you should realize that RADIUS is more vendor agnostic than TACACS+.

Terminal Access Controller Access-Control System Plus (TACACS+) is a protocol that allows client systems to make use of a server to authenticate users. This service provides AAA management in a centralized location.

TACACS+ is often compared to RADIUS, but knowing the differences between these protocols is beyond the scope of the Linux+ XK0-005 exam. Most importantly, realize that TACACS+ can provide very powerful functions within a security policy because it offers centralized AAA management. Also, you should realize that TACACS+ was developed by Cisco and is not as vendor agnostic as RADIUS.

Lightweight Directory Access Protocol (LDAP) is a protocol that provides directory services information. This means the protocol can be used to provide user account information, but it does not provide the full AAA functionality that RADIUS and TACACS+ provide. LDAP is also often used to store other information not directly related to user accounts, such as hostnames.

LDAP can be used to authenticate users, but when considering a security policy, you should realize that RADIUS and TACACS+ provide a more robust solution. For example, LDAP does not provide the accounting that AAA protocols like RADIUS and TACACS+ provide.

ExamAlert

Configuration of LDAP is beyond the scope of the Linux+ XK0-005 exam. For the exam you should be aware of the function of this service.

Single Sign-on (SSO)

Single sign-on (SSO) is a technique in which once a user authenticates on one system, this authentication can be used to automatically grant access to other systems. This is normally implemented using a RADIUS or TACACS+ server. (See the "Lightweight Directory Access Protocol (LDAP)" section, earlier in this chapter, for more information about RADIUS and TACACS+.)

Linux Hardening

Linux hardening is a process that involves making changes to the Linux operating system to make it more secure. This section focuses on several different techniques that can be used to perform Linus hardening operations.

Security Scanning

Scanning is the process of using a tool to search for security weaknesses. For example, the **john** utility is a command-line password hacking tool. With this tool, you need to have a file that contains both **/etc/passwd** and **/etc/shadow** entries. You can make this file by running the following command:

```
cat /etc/passwd /etc/shadow > file_to_store
```

Then you run the **john** utility on the combined file, and it will discover which user accounts have weak passwords.

A wide variety of scanning tools can scan for weaknesses such as the following in different areas of the Linux operating system:

- Software configuration errors
- Root kits
- File and directory permissions issues
- Kernel modules
- Viruses
- Malware
- Network port issues
- Intrusions

ExamAlert

Given the large number of scanning tools available, you shouldn't expect the Linux+ XK0-005 exam to ask any questions about specific tools.

Secure Boot (UEFI)

When a Linux system is first turned on, a software program is started to boot the system. This software program is not part of Linux but rather a program that came with your system hardware. There are two different types of programs: Basic Input/Output System (BIOS) and Unified Extensible Firmware Interface (UEFI). While BIOS and UEFI perform similar tasks (starting the boot process), they have different features and are configured using different interfaces.

One advantage of UEFI is that it provides a feature called *secure boot* that addresses the concern of booting from a boot image that is not authorized or that may contain malware. Digital signatures are used to confirm the validity of the boot image.

System Logging Configurations

For detailed information about system logging configurations, see the section "Syslog" in Chapter 7, " Manage Software Configurations."

Setting Default umask

The **umask** command sets default permissions for files and directories. These default permissions are applied only when the file or directory is initially created.

The **umask** command accepts one argument: the mask value. The mask value is an octal value that is applied against the maximum possible permissions for new files or new directories, as shown in Table 8.2.

TABLE 8.2 **umask Permissions**

Type	Maximum Possible Permission for a New Item
File	**rw-rw-rw-**
Directory	**rwxrwxrwx**

Figure 8.4 illustrates how a **umask** value of **027** would apply to new files compared to how it would apply to new directories.

Description	File			Directories		
Maximum	rw-	rw-	rw-	rwx	rwx	rwx
Umask Applied	---	-M-	MM-	---	-M-	MM-
Result	rw-	r--	---	rwx	r-x	--x

FIGURE 8.4 **How umask Is Applied**

> **Note**
>
> Each shell has its own **umask** value. If you change the **umask** in one shell, it will not affect the **umask** value in any other shell. To make a persistent change to your **umask** across logins, add the **umask** command to the **~/.bash_profile** file.

Disabling/Removing Insecure Services

Some services should be used either never or sparingly. Table 8.3 focuses on insecure services and the limited cases in which they should be used.

TABLE 8.3 **Insecure Services Concerns**

Service	Description
FTP	File Transfer Protocol (FTP) is a commonly used protocol designed to transfer files between systems. Because data is sent between the FTP client and server unencrypted, the protocol is not secure. This data includes any user account information, which means anyone who can "snoop" the network can see a user's name and password. FTP servers provide an important and useful function: anonymous FTP service. An anonymous FTP server doesn't require any user authentication and should only allow for the downloading of content (with no uploading permitted). This allows someone to create a repository that holds content that is designed for anyone to download (like the ISO files for a Linux distribution).
Telnet	The Telnet protocol permits remote login, but unlike SSH, it provides no encryption. While the **telnet** command may be used to test connections to remote ports, a Telnet server should not be used on modern systems where security is important.

Service	Description
Finger	The Name/Finger protocol is designed to provide information about computers and users. The **finger** command has long been used as a way to provide reports on users. However, because this information is sent through the network unencrypted, this command and protocol should never be used on modern systems where security is important.
Sendmail/Postfix	Both sendmail and postfix are email servers. Often there is no need for email services on the local system, or at least no email should be needed externally. A real email server should be configured to send and receive messages to and from remote email servers. Desktop systems and most non-email servers should not be configured to handle external email. This may require some configuration changes to these services.

You should consider disabling or limiting all services that do not need to have remote access. For example, on most Linux distributions, the CUPS print server is installed and configured to start by default. If the system in question has no need for printing, consider disabling and perhaps removing the software for CUPS.

Enforcing Password Strength

The **pam_unix** module provides a lot of features that modify how passwords are set. For example, consider the following commands:

```
[root@OCS ~]# grep ^password /etc/pam.d/passwd
password include  system-auth
[root@OCS ~]# grep ^password /etc/pam.d/system-auth
password requisite pam_cracklib.so try_first_pass retry=3
password sufficient pam_unix.so md5 shadow nullok try_first_pass
use_authtok
password required pam_deny.so
```

The **shadow** option to the **pam_unix** module causes passwords to be stored using the shadow system. Passwords are stored in the **/etc/shadow** file, not the **/etc/passwd** file.

You can make some changes to the **pam_unix** module to make your system more secure. For example, if you modify the **pam_unix** line in the **/etc/pam.d/**

system-auth file to match the following, the system does not allow users to reuse passwords:

```
password sufficient pam_unix.so md5 shadow nullok try_first_pass use_
authtok remember=5
```

With the line modified in this way, the last five passwords for each user are remembered, and the user cannot use one of these five passwords when changing to a new password. This prevents users from using the same passwords over and over.

Table 8.4 describes some additional important **pam_unix** options.

TABLE 8.4 **pam_unix Password Options**

Option	Description
md5	Encrypts a password using the MD5 encryption algorithm.
sha256	Encrypts a password using the SHA256 encryption algorithm.
nullok	Allows the root user to provide users with null (empty) passwords.
remember=x	Remembers old user passwords and doesn't allow users to use those passwords when changing passwords.

Removing Unused Packages

Removing unused packages may be a challenge at first, but it is very important that you only have necessary software installed on a system. The challenge of determining which packages are necessary is that a typical default Linux installation includes thousands of software packages. For example, a default installation of Kali Linux has almost 3,000 packages, as demonstrated here:

```
root@kali:~# dpkg -l | wc -l
2801
```

Do you really need to decide which packages should be installed on each system by reviewing each package one at a time? You should explore what each package in a distribution provides and then create rules for different kinds of systems. For example, you should determine what belongs on a typical user workstation and what belongs on a web server. In fact, you should create three categories:

- The software that should be installed by default
- Software considered optional (that is okay to install but is not installed by default)

- Potentially risky or problematic software that should never be installed on any system

You can determine whether a package should be installed by exploring the package. For example, you might first look at the package to determine what it does, as in the following example:

```
[root@onecoursesource ~]#rpm -qi zsh | grep "Description"
Description :
The zsh shell is a command interpreter usable as an interactive login
shell and as a shell script command processor. Zsh resembles the ksh
shell (the Korn shell), but includes many enhancements. Zsh supports
command line editing, built-in spelling correction, programmable
command completion, shell functions (with autoloading), a history
mechanism, and more.
```

Based on the information provided by this command, you may be able to determine that this package should not be installed on the system. Perhaps you do not want to support another shell and only want to support BASH on this system. Or you might consider this package unnecessary because a good shell already exists. If you are considering an installation that requires an additional shell, be sure to investigate the implications and ramifications of running a second shell on the system prior to installation.

If you decide that a package might be a good package to have, then before approving it, you should also look at the files it provides—particularly the executable programs (which are typically located in a **/bin** directory, such as **/usr/bin** or **/bin**, but could also be located under **/sbin** or **/usr/sbin**):

```
[root@onecoursesource ~]# rpm -ql zsh | grep /bin/
/usr/bin/zsh
[root@onecoursesource ~]# ls -l /usr/bin/zsh
-rwxr-xr-x 1 root root 726032 May 22  2015 /usr/bin/zsh
```

Look at the documentation (man pages) for these executable programs and look closely at the permissions and ownership of the files. Consider whether each program provides elevated privileges because of a SUID or SGID permission.

Finally, note that the commands demonstrated in this section were performed on a system that already had the software installed. You should vet software packages on a test system so that you can safely install packages without

worrying about impacting a critical system. This system should not even be attached to the network. You want to be able to run the commands and test each of them before deciding if the software is safe to install on "real" systems in your organization.

Tuning Kernel Parameters

Note that kernel parameters are covered in the section "Configure Kernel Options" in Chapter 7. This section focuses on kernel parameters that can be used to make the kernel more secure. Four commonly used kernel parameters are discussed in this section:

- Ignoring ping requests
- Ignoring broadcast requests
- Enabling TCP SYN protection
- Disabling IP source routing

The **ping** command is often used to determine if a remote host is accessible through the network. It does this by sending an Internet Control Message Protocol (ICMP) echo request packet to the remote host, and it expects a response packet if the remote system is reachable. This poses two security challenges:

- A hacker can use the **ping** command to probe for active systems, trying to find systems that she can break into. While this is not the only way to probe for active systems, it is a very common one. As a result, it is best to ignore ping requests for mission-critical systems.
- Responding to ping requests can leave a system vulnerable to DoS (denial of service) attacks. In a DoS attack, a large number of network packets are sent to a system, making it difficult for the system to handle all of the data and rendering the system nonresponsive. In a similar attack, called a DDoS (distributed denial of service) attack, packets are sent from multiple systems to overwhelm a host. To prevent DoS and DDoS attacks, it is best to ignore ping requests for mission-critical systems.

To ignore ping requests, use the following setting in the **/etc/sysctl.conf** file:

```
net.ipv4.icmp_echo_ignore_all = 1
```

To ignore broadcast requests, which occur with DoS or DDoS attacks, use the following setting in the **/etc/sysctl.conf** file:

```
net.ipv4.icmp_echo_ignore_broadcasts = 1
```

A SYN flood attack is a type of DoS attack in which SYN requests are used to make a system unresponsive. To ignore SYN requests, use the following setting in the **/etc/sysctl.conf** file:

```
net.ipv4.tcp_syncookies = 1
```

IP source routing is a feature that enables the sender of a packet to specify the network route that should be taken. This feature bypasses routing tables and makes the system vulnerable as hackers can employ on-path attack attacks, also referred to as man-in-the-middle attacks (in which a system interjects itself between a client and a server), or use this feature to map your network or bypass your firewalls.

Each network interface has a separate IP source routing kernel parameter. To disable this feature from a specific network device, use a setting like the following:

```
net.ipv4.conf.eth0.accept_source_route = 0
```

Securing Service Accounts

A typical Linux system has many service user accounts. These service user accounts typically have UID values under 1,000, making it easy for an administrator to recognize these as special accounts. Some of these service accounts are often referred to as "daemon accounts" because they are used by daemon-based software. *Daemons* are programs that run in the background, performing specific system tasks.

Other service accounts may exist to provide features for the operating system. For example, the "nobody" account is used to apply permissions for files that are shared via NFS (Network File System).

In addition, if you add new software to a system, more users might be added because software vendors make use of both user and group accounts to provide controlled access to files that are part of the software.

These service accounts should not have passwords set on them as they aren't used by regular users. In other words, if you look at the password field in the **/etc/shadow** file, all system accounts should have values like the following:

```
nobody:*:15921:0:99999:7:::
dbus:!!:16056::::::
```

A system account also should not have a valid shell associated with it. So, if you look in the **/etc/password** file, the login shell for a system account should look like the following:

```
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
```

You should perform routine audits on system accounts to ensure that they don't have passwords or valid login shells.

Configuring the Host Firewall

A *firewall* (also known as an access control list) is a network appliance (either hardware or software based) that is designed to either allow or block network traffic. Firewalls can be implemented on a variety of devices, including routers, network servers, and users' systems (in which case they are known as host firewalls). This section explores the basic concepts of firewalls.

Table 8.5 describes key firewall terms.

TABLE 8.5 **Firewall Terminology**

Term	Description
Source	This term refers to where a network packet originated. Firewall rules can be created to filter network traffic from the source by using the source's IP address, port, or MAC address.
Destination	This term refers to where a network packet is being sent. Firewall rules can be created to filter network traffic for the destination by using either the destination's IP address, port, or MAC address.
Port	A port has a unique number that is used to address a service on a system. Each network packet includes a source port and a destination port. Firewall rules can be created to filter network traffic by using the source or destination port number.
Protocol	It is common to filter packets by protocol—either a protocol like ICMP, TCP, or UDP or a protocol associated with a specific port (such as Telnet, which uses port 23).
Log	Firewall rules are normally used to allow or block a network packet; however, there are also rules that are designed to log information regarding a network packet. This is useful when you want to see information about a packet that you will block with a later rule.

Term	Description
Stateful and stateless	A stateful rule is a rule that applies to any previously established connection. Consider a firewall that specifies the following: Allow most outbound connections, including to websites, SSH access, FTP sites, and so on. Block most inbound connections so outside users can't connect to services you may have running. The problem with this scenario is that when you connect to a remote system, both outbound and inbound rules apply. The outbound rules apply when you establish a connection, and the inbound rules apply when the remote server responds. If you have a rule that blocks most inbound connections, you may never receive the response from that website you are trying to visit. You can overcome this by creating stateful rules. Stateful rules essentially say, "if a network packet is responding to a request that the local machine initiated, let it through the firewall." A stateless rule applies regardless of any previously established connection.
iptables targets	Once a network packet matches the criteria of a firewall rule, a target is used to determine what action to take. There are four standard targets for **iptables** (a commonly used Linux firewall tool): The Accept target tells **iptables** to allow the packet to advance to the next filtering point. The Reject target tells **iptables** to return the packet to the source with an error message. The message is not advanced to the next filtering point. The Drop target tells **iptables** to discard the packet. No response is sent to the source, and the message is not advanced to the next filtering point. The Log target tells **iptables** to create a log entry that contains detailed information about the packet. With this target, the packet is neither explicitly allowed nor blocked, but further rules in the rule set (the chain) may determine this result.

One commonly used firewall tool is called **iptables**. With **iptables**, rule sets (called *chains*) are applied at different places (called *filter points*), allowing for more flexibility in this ACL software. Figure 8.5 demonstrates these filter points.

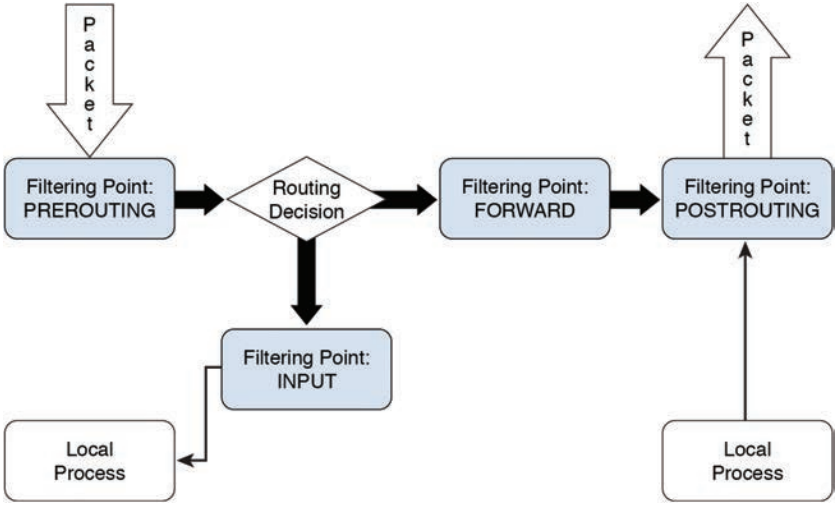

FIGURE 8.5 **Packet Filtering: Outbound Packets**

A *table* of rules can be placed on a filtering point. A filtering point can have one or more sets of rules because **iptables** performs multiple functions: either filter (block or allow) the data, perform a NAT operation on the packet, or mangle the packet. The combination of the filtering point plus the table (**filter**, **nat**, or **mangle**) forms a single set of rules called a *chain*.

Consider a chain to be a set of rules that determines what actions to take on a specific packet. For example, a rule on the "filter INPUT" chain could block an incoming packet based on the source IP address. Another rule could be used to allow packets destined for a specific network port.

The order of the rules is also important. Once a matching rule is found, an action (called a *target*) takes place, and additional rules are ignored (with one exception, as noted next). As mentioned in Table 8.5, the typical targets are Accept, Drop, Reject, and Log.

Table 8.6 describes several common **iptables** rules.

TABLE 8.6 **iptables Rules**

Rule	Description
iptables -t filter -L INPUT	Displays all the rules for the INPUT-filter chain.
iptables -D INPUT 1	Deletes rule 1 from the INPUT-filter chain.
iptables -F INPUT	Flushes all the rules of the INPUT-filter chain.

Rule	Description
iptables -A INPUT -s 192.168.10.100 -j DROP	Appends a rule at the end of the INPUT-filter chain that drops any packet from the 192.168.10.100 system.
iptables -A INPUT -s 192.168.20.0/24 -j DROP	Appends a rule at the end of the INPUT-filter chain that drops any packet from the 192.168.10.0/24 network.
iptables -I INPUT 2 -s 192.168.20.125 -j ACCEPT	Inserts this rule as rule 2 and moves all remaining rules down by one.
iptables -A INPUT -p icmp -j DROP	Creates a rule that drops packets that match the ICMP protocol.
iptables -A INPUT -m tcp -p tcp --dport 23 -j DROP	Creates a rule that drops packages that are destined for port 23.
iptables -A INPUT -p icmp -s 192.168.125.125 -j DROP	Creates a rule that must match all criteria (ICMP protocol and source IP address 192.168.125.125).
iptables -A INPUT -i eth0 -s 192.168.100.0/24 -j DROP	Creates a rule that filters based on network interface.
iptables -L INPUT -v	Displays detailed information about the rules.
iptables -P INPUT DROP	Sets the default chain policy. If no rules match, this default target is used.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. When a web server sends its public key, it includes a digital signature. This digital signature can be sent to a CA server, which is a trusted third-party system used to verify the digital signature. In some cases, the server itself provides the signature as a(n) _____certificate.

   - ❍ **A.** self-signed
   - ❍ **B.** independent
   - ❍ **C.** solo
   - ❍ **D.** invalid

2. Which of the following allows for the signing of multiple systems within a domain?

   - ❍ **A.** Domain certificate
   - ❍ **B.** Wildcard certificate
   - ❍ **C.** Multi certificate
   - ❍ **D.** Range certificate

3. Which of the following is not a component of MFA?
   - ❍ **A.** Something the user knows
   - ❍ **B.** Somewhere the person has been
   - ❍ **C.** Something the user has
   - ❍ **D.** Something the user "is"

4. Which command is used to set the default mask for files to **rw-r-----**?
   - ❍ **A. umask 026**
   - ❍ **B. umask 022**
   - ❍ **C. umask 002**
   - ❍ **D. umask 066**

Cram Quiz Answers

1. **A.** The correct term is *self-signed*. The other answers are not valid certificate-signing terms.
2. **B.** A wildcard certificate applies to an entire domain, including the subdomains for that domain. The other answers are not valid certificate terms.
3. **B.** One common multifactor authentication method is called two-factor authentication, or 2FA. For this method, the user is required to provide two forms of identification, which could include the following:
   - ❍ Something the user knows
   - ❍ Something the user has
   - ❍ Something the user "is"
4. **A.** A **umask** of 022 would result in the default permission **rw-r--r--** for files. A **umask** of 002 would result in the default permission **rw-rw-r--** for files. A **umask** of 066 would result in the default permission **rw-------** for files.

CHAPTER 9

Implement Identity Management

This chapter covers the following Linux+ XK0-005 exam objective:

- **2.2:** Given a scenario, implement identity management.

Identity management is the process of managing user and group accounts. On a Linux system, account information is stored in the plaintext files **/etc/passwd**, **/etc/shadow**, and **/etc/group**, which you will learn about in this chapter.

In this chapter, you will also learn how to create, modify, and delete user accounts. In addition, you will discover how to view account information by using the commands **id**, **who**, and **w**.

This chapter provides information on account creation and deletion as well as account management.

Account Creation and Deletion

One of the most common operations that a Linux system administrator is required to perform is creating and deleting user and group accounts. This section describes the commands used to create accounts, display information about accounts, and delete user accounts.

useradd

The **useradd** command is used by the root user to create a user account.

Example:

```
[root@OCS ~]$ useradd julia
```

Table 9.1 lists some important options for the **useradd** command.

TABLE 9.1 **useradd Command Options**

Option	Description
-c	Sets the comment or GECOS field for the user.
-d	Specifies the home directory for the user. This option is often used together with the **-m** option, which creates the home directory.
-e	Sets the account's Expiration Date value. (See the "**/etc/shadow**" section, later in this chapter, for more details.)
-f	Sets the account's Inactive value. (See the "**/etc/shadow**" section, later in this chapter, for more details.)
-g	Specifies the user's primary group.
-G	Specifies the user's secondary groups.
-k	Specifies the **skel** directory, which is the directory from where files are copied automatically into the user's home directory.
-s	Specifies the user's login shell.
-u	Specifies the user's UID.

groupadd

The **groupadd** command is used by the root user to create a group account.

Example:

```
[root@OCS ~]$ groupadd -g 2050 test
```

The **-g** option is used to specify the GID for the new group.

userdel

The **userdel** command is used by the root user to delete a user account.

Example:

```
[root@OCS ~]$ userdel susan
```

An important option for the **userdel** command is the **-r** option, which deletes the user account as well as the user's home directory and mail spool.

groupdel

The **groupdel** command is used by the root user to delete a group account.

Example:

```
[root@OCS ~]$ groupdel test
```

> **Note**
>
> Be sure to remove all files owned by a group (or reassign the files to another group) before running the **groupdel** command.

usermod

The **usermod** command is used by the root user to modify a user account.

Example:

```
[root@OCS ~]$ usermod -s /bin/tcsh julia
```

The **usermod** command uses many of the same options as the **useradd** command. See the "**useradd**" section, earlier in this chapter, for a list of these options.

groupmod

The **groupmod** command is used by the root user to modify a group account.

Example:

```
[root@OCS ~]$ groupmod -n proj test
```

Table 9.2 describes some important options for the **groupmod** command.

TABLE 9.2 **groupmod Command Options**

Option	Description
-g	Changes the GID.
-n	Changes the group name.

id

The **id** command displays basic account information. When run with no arguments, it displays the current user's UID, username, and primary GID and name, as well as all secondary group memberships, as in this example:

```
[root@OCS ~]$ id
uid=0(root) gid=0(root) groups=0(root)
```

When passed a username as an argument, it provides information about the user specified, as in this example:

```
[root@OCS ~]$ id bo
uid=1001(bo) gid=1001(bo)
groups=1001(bo),10(wheel),20001(temprandy)
```

While there are a few options for the **id** command, they are not typically used.

who

The **who** command shows who is currently logged in.

Example:

```
[root@OCS ~]# who
student     :0     2017-02-18 01:52 (:0)
student      pts/0 2017-02-18 01:52 (:0)
student      pts/1 2017-03-05 19:55 (:0)
student      pts/2 2017-03-06 18:24 (:0)
root         pts/3 2017-03-06 18:24 (localhost)
```

The output of the **who** command includes the username, the terminal device the user is using, the login date and time, and where the user logged in from (where **:0** means a local login). Table 9.3 describes common options for the **who** command.

TABLE 9.3 **who Command Options**

Option	Description
-b or **--boot**	Specifies the time of the last system boot.
-H or **--heading**	Displays headings on columns.
-q or **--count**	Displays the number of users currently logged in.

w

The **w** command displays who is logged in as well as other useful information:

```
[root@OCS ~]# w
 18:25:08 up 3 days, 1:24, 5 users, load average: 0.27, 0.08, 0.07
USER     TTY      FROM       LOGIN@ IDLE JCPU PCPU WHAT
student :0       :0         18Feb17   41:48 1.01s    gdm-session-wor
student pts/0    :0         18Feb17   4.00s 0.46s    20.33s /usr/
libexec/gn
student pts/1    :0         Sun19 1:32 0.04s 0.00s less -s
student pts/2    :0         18:24 12.00s 0.05s    0.01s /usr/bin/
sss_ss
root    pts/3     localhost  18:24 12.00s 0.03s     0.03s -bash
```

The JCPU column head stands for "Job CPU," and this column indicates how much CPU time has been used by all processes that were launched from the terminal. The PCPU column head stands for "Process CPU," and this column indicates how much CPU time has been used by the current process (which is listed as the last item in the line of output).

Table 9.4 describes common options for the **w** command.

TABLE 9.4 **w Command Options**

Option	Description
-h or **--no-header**	Specifies not to display headings on columns.
-s or **--short**	Specifies not to display the JCPU and PCPU columns.

Default Shell

When using the **useradd** command to create a new user account, a default shell is used unless the **-s** option is used to specify a shell. The default shell value can be viewed and modified by using the **-D** option to the **useradd** command. For example, to view the default shell, use the following command and look for the **SHELL** setting:

```
[root@onecoursesource.com ~]# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
```

```
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

To change the default shell, use the **-s** option in combination with the **-D** option, as shown below:

```
[root@mail ~]# useradd -D -s /bin/csh
[root@mail ~]# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/csh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

/etc/passwd

The **/etc/passwd** file is used to store user account information. It contains most of the user account data, except for the password and password-aging policies. (See the "**/etc/shadow**" section, later in this chapter, for details about those settings.)

Each line in the **/etc/passwd** file describes one user account.

Example:

```
root:x:0:0:root:/root:/bin/bash
```

Every line is broken into fields of data. Table 9.5 describes these fields.

TABLE 9.5 **/etc/passwd File Data Fields**

Field	Description
Username	**root**:x:0:0:root:/root:/bin/bash This is the name the user provides when logging in to the system.
Password placeholder	root:**x**:0:0:root:/root:/bin/bash This is the place where the password used to be stored in older versions of Unix.
UID	root:x:**0**:0:root:/root:/bin/bash This is the user ID.

Field	Description
GID	root:x:0:**0**:root:/root:/bin/bash This is the user's primary group ID.
GECOS	root:x:0:0:**root**:/root:/bin/bash GECOS, which stands for General Electric Comprehensive Operating System, is typically the user's real name and other identifying data.
Home directory	root:x:0:0:root:**/root**:/bin/bash This is the user's home directory.
Login shell	root:x:0:0:root:/root:**/bin/bash** This is the user's login shell.

ExamAlert

When you take the Linux+ XK0-005 exam, you will likely see an example of an **/etc/passwd** file and be asked the purposes of different fields in the file.

/etc/group

The **/etc/group** file is used to store group account information. Each line in the **/etc/group** file describes one group account.

Example:

```
admin:x:110:bob,sue
```

Every line is broken into fields of data. Table 9.6 describes these fields.

TABLE 9.6 **/etc/group File Data Fields**

Field	Description
Group name	**admin**:x:110:bob,sue This is the name of the group.
Password placeholder	admin:**x**:110:bob,sue This is the place where the password used to be stored in older versions of Unix.
GID	admin:x:**110**:bob,sue This is the group ID.
Member list	admin:x:110:**bob,sue** These are the members of the group.

ExamAlert

When you take the Linux+ XK0-005 exam, you will likely see an example of an **/etc/group** file and be asked the purposes of different fields in the file.

/etc/shadow

The **/etc/shadow** file is used to store user password information. (See the "**/etc/passwd**" section, earlier in this chapter, for details about other user account data.)

Each line in the **/etc/shadow** file describes one user account. Here's an example of an account with a valid password:

```
student:$6$ciPSOxID$E5p6cgsPs2lng7dQjrVMIUBGhd/dqs49d
jnCB1h1oGhryfitzaVGvsODflyNU67uX3uBraVY0GIOO2zaVGeZ/.:
17116:0:99999:7:30:17246:
```

Note

In the **/etc/shadow** file, the account information appears in a single line.

Here's an example of an account with a locked password:

```
bob:*LK*:17116:0:99999:7:30:17246:
```

Every line is broken into fields of data. Table 9.7 describes these fields.

TABLE 9.7 **/etc/shadow File Data Fields**

Field	Description
Username	**bob**:*LK*:17116:0:99999:7:30:17246: This is the user's name, which is matched up to the entry in the **/etc/passwd** file.
Password	bob:***LK*:**17116:0:99999:7:30:17246: This is the user's password or some locked value. Note that passwords are encrypted.
Last change	bob:*LK*:**17116**:0:99999:7:30:17246: This is the number of days since January 1, 1970, and the last time the user's password was changed. It's used by the system for the next three fields.

Field	Description
Min	bob:*LK*:17116:**0**:99999:7:30:17246: This is how many days after a user's password is changed before the user can change the password again.
Max	bob:*LK*:17116:0:**99999**:7:30:17246: This is how many days after a user's password is changed before the user must change the password again. If the user doesn't change the password before this time limit, the account is locked.
Warn	bob:*LK*:17116:0:99999:**7**:30:17246: This is how many days before the account is to be locked to start issuing warnings to the user as the user logs in.
Inactive	bob:*LK*:17116:0:99999:7:**30**:17246: After the account is locked, this is how many "grace days" the user has to log in—but only if a new password is provided at login.
Expiration date	bob:*LK*:17116:0:99999:7:30:**17246**: This is the number of days since January 1, 1970, until the user's account will expire.
Unused	bob:*LK*:17116:0:99999:7:30:17246: This is an unused field that may be used in the future.

ExamAlert

When you take the Linux+ XK0-005 exam, you will likely see an example of an **/etc/shadow** file and be asked the purposes of different fields in the file.

/etc/profile

When a user logs in to the system, a login shell is started. When a user starts a new shell after login, it is referred to as a *non-login shell*. In each case, initialization files are used to set up the shell environment. Which initialization files are executed depends on whether the shell is a login shell or a non-login shell.

Figure 9.1 demonstrates which initialization files are executed when the user logs in to the system.

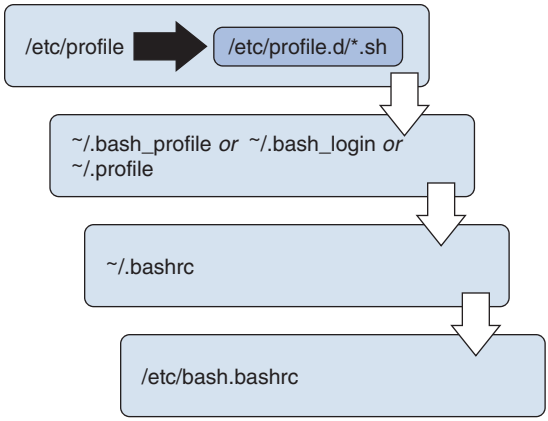

FIGURE 9.1 **Initialization Files Executed When the User Logs In to the System**

Figure 9.1 illustrates the following process:

1. When a user logs in, Linux executes the **/etc/profile** initialization file. On most Linux platforms, this script includes code that executes all the initialization files in the **/etc/profile.d** directory that end in **.sh**. *The* **/etc/profile** *file is a place for system administrators to put code (typically login messages and environment variable definitions) that will execute every time every BASH shell user logs in.*
2. The login shell looks in the user's home directory for a file named **~/.bash_profile**. If it's found, the login shell executes the code in this file. Otherwise, the login shell looks for a file named **~/.bash_login**. If it's found, the login shell executes the code in this file. Otherwise, the login shell looks for a file named **~/.profile** and executes the code in this file. *These files provide a place where each user can put code that will execute every time that specific user logs in (typically environment variable definitions).*
3. Linux executes the **~/.bashrc** initialization file. *This file is a place where each user can put code (typically alias and function definitions) that will execute every time the user opens a new shell.*
4. Linux executes the **/etc/bash.bashrc** initialization file. *This file is a place where system administrators can put code (typically alias and function definitions) that will execute every time the user opens a new shell.*

Figure 9.2 illustrates which initialization files are executed when the user opens a new shell.

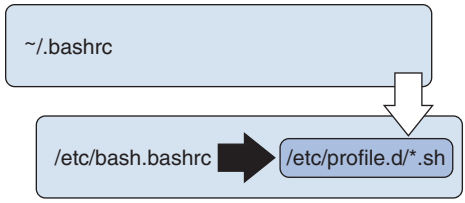

FIGURE 9.2 **Initialization Files Executed When a User Starts a Non-Login Shell**

Figure 9.2 illustrates the following process:

1. When a user logs in, Linux executes the **~/.bashrc** script. *This file is a place where each user can put code that will execute every time that user opens a new shell (typically alias and function definitions).*
2. Linux executes the **/etc/bash.bashrc** initialization file. On most Linux platforms, this script includes code that executes all the initialization files in the **/etc/profile.d** directory that end in **.sh**. *These files provide a place where system administrators can put code that will execute every time the user opens a new shell (typically alias and function definitions).*

ExamAlert

Expect questions on the Linux+ XK0-005 exam to test your understanding of the login process, including which file is executed at which point during the login process.

/etc/skel

The **/etc/skel** directory is used when a new user account is created to provide the new account with default files, such as BASH configuration files (**.bashrc**, **.profile**, and so on). See the "**useradd**" section, earlier in this chapter, for additional details.

.bash_profile

When a user logs in to the system and the user's login shell is a BASH shell, the commands in the **~/.bash_profile** file are executed if this file exists. This enables the user to set up the account by placing commands in this file.

See the "**/etc/profile**" section, earlier in this chapter, for further details regarding how initialization files are used.

.bashrc

When a user opens a new BASH shell, the commands in the **~/.bashrc** file are executed. The user can set up the account by placing commands in this file.

See the "**/etc/profile**" section, earlier in this chapter, for further details regarding how initialization files are used.

Account Management

Account management involves using commands to change user accounts. This section focuses on the vital commands you need to know to manage user accounts.

passwd

The **passwd** command allows a user to change her password. The root user can also use this command to change any user password or to change other password-based features for a user account. Table 9.8 describes some important options for the **passwd** command.

TABLE 9.8 **passwd Command Options**

Option	Description
-d	Deletes the user's password.
-e	Expires the user account immediately.
-l	Locks the account.
-u	Unlocks the account.
-m	Changes the **Min** field of the **/etc/shadow** file for the user.
-M	Changes the **Max** field of the **/etc/shadow** file for the user.
-w	Changes the **Warn** field of the **/etc/shadow** file for the user.

ExamAlert

Be ready for Linux+ XK0-005 exam questions that ask you which **passwd** options change which **/etc/shadow** fields.

chage

The **chage** command is executed by the root user to modify password-aging features for a user account. Table 9.9 describes some important options for the **chage** command.

TABLE 9.9 **chage Command Options**

Option	Description
-d	Changes the **Last Change** field of the **/etc/shadow** file for the user.
-E	Sets the **Expiration Date** field of the **/etc/shadow** file for the user (for example, **chage -E 2025-01-01 bob**).
-m	Changes the **Min** field of the **/etc/shadow** file for the user.
-M	Changes the **Max** field of the **/etc/shadow** file for the user.
-W	Changes the **Warn** field of the **/etc/shadow** file for the user.

See the "**/etc/shadow**" section, earlier in this chapter, for additional information regarding password-aging settings.

ExamAlert

Be ready for Linux+ XK0-005 exam questions that ask you which **chage** options change which **/etc/shadow** fields.

pam_tally2

You can lock out a user due to unsuccessful login attempts by using the following highlighted setting in the **/etc/pam.d/password-auth** file:

```
auth required     pam_env.so
auth required pam_tally2.so file=
   /var/log/tallylog deny=3 even_deny_root unlock_time=1200
auth    sufficient pam_unix.so try_first_pass nullok
auth    required  pam_deny.so
account required  pam_unix.so
account required  pam_tally2.so
```

Here are the key values in these lines:

- **file:** Specifies the name of the file to store the failed login attempts.

- **deny:** Specifies when to lock out the user (after a specified number of unsuccessful attempts).
- **even_deny_root:** Applies to the root account as well as regular users.
- **unlock_time:** Specifies the time, in seconds, to keep the account locked.

See the "Pluggable Authentication Modules (PAM)" section in Chapter 8, "Security Best Practices in a Linux Environment," for more details regarding using PAM.

faillock

You can lock out a user due to unsuccessful login attempts by using the following highlighted settings in the **/etc/pam.d/system-auth** file:

```
auth   required   pam_env.so
auth required pam_faillock.so preauth
 silent audit deny=4 unlock_time=1200
auth   sufficient  pam_unix.so try_first_pass nullok
auth [default=die] pam_faillock.so authfail
 audit deny=4 unlock_time=1200
auth   required   pam_deny.so
```

Here are the key values in these lines:

- **deny:** Specifies when to lock out the user (after a specified number of unsuccessful attempts).
- **unlock_time:** Specifies the time, in seconds, to keep the account locked.

See the "Pluggable Authentication Modules (PAM)" section in Chapter 8 for more details regarding using PAM.

/etc/login.defs

The **/etc/login.defs** file provides several default values for when a new account is created or when an account password is changed. The following example shows the values of this file (not including the comment lines):

```
[bo@onecoursesource.com ~]$ grep -v "^#" /etc/login.defs
MAIL_DIR        /var/spool/mail
```

```
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_MIN_LEN    5
PASS_WARN_AGE   7
UID_MIN                  1000
UID_MAX                 60000
SYS_UID_MIN               201
SYS_UID_MAX               999
GID_MIN                  1000
GID_MAX                 60000
SYS_GID_MIN               201
SYS_GID_MAX               999
CREATE_HOME     yes
UMASK           077
USERGROUPS_ENAB yes
ENCRYPT_METHOD SHA512
```

Table 9.10 describes some important settings for the **/etc/login.defs** file.

TABLE 9.10 **/etc/login.defs File Settings**

Setting	Description
PASS_MAX_DAYS	The default value for a new account's **Max** value. (See the "**/etc/shadow**" section, earlier in this chapter, for more details about this value.)
PASS_MIN_DAYS	The default value for a new account's **Min** value. (See the "**/etc/shadow**" section, earlier in this chapter, for more details about this value.)
PASS_MIN_LEN	The minimum number of characters a user's password must be when a regular user changes his own password.
PASS_WARN_AGE	The default value for a new account's **Warn** value. (See the "**/etc/shadow**" section, earlier in this chapter, for more details about this value.)

ExamAlert

The other settings in the **/etc/login.defs** file should not be tested on the Linux+ XK0-005 exam.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. You are asked to create a user and assign them to a secondary group called payroll. Which option is missing in the command **useradd __ payroll bill** to accomplish this task?

   - ❍ **A. -a**
   - ❍ **B. -g**
   - ❍ **C. -G**
   - ❍ **D. -c**

2. Which option to the **useradd** command displays the default shell?

   - ❍ **A. -D**
   - ❍ **B. -d**
   - ❍ **C. -s**
   - ❍ **D. -t**

3. What does the value **7** represent in the following line from the **/etc/shadow** file?

   ```
   bob:*LK*:17116:0:99999:7:30:17246:
   ```

   - ❍ **A.** After the account is locked, the number of how many "grace days" the user has to log in, but only if a new password is provided at login
   - ❍ **B.** How many days after a user's password is changed before the user must change his password again
   - ❍ **C.** How many days after a user's password is changed before the user can change his password again
   - ❍ **D.** How many days before the account is to be locked to start issuing warnings to the user as the user logs in

4. Which login files are executed only during a login process and not when a new shell is opened after login? (Choose two.)

   - ❍ **A. /etc/profile**
   - ❍ **B. .bashrc**
   - ❍ **C. ~/.profile**
   - ❍ **D. /etc/bash.bashrc**

Cram Quiz Answers

1. **C.** The **-G** option adds a user to a group as a secondary member. The **-g** option sets the user's primary group. The **-c** option is used to specify a comment for the user. There is no **-a** option to the **useradd** command.
2. **A.** The **-D** option displays default values used for creating accounts, including the default shell. The **-d** and **-t** options are not valid for the **useradd** command. The **-s** option sets a shell; it doesn't display the shell.
3. **D.** The sixth field of the **/etc/shadow** file specifies how many days before the account is to be locked to start issuing warnings to the user as the user logs in. The fourth field indicates how many days after a user's password is changed before the user can change his password again. The fifth field shows how many days after a user's password is changed before the user must change the password again. The seventh field indicates how many "grace days" the user has to log in after the account is locked.
4. **A and C.** Both the **profile** and **bashrc** files are run during the login process. When a new shell is opened post-login, only the **bashrc** file is run.

This page intentionally left blank

CHAPTER 10

Implement and Configure Firewalls

This chapter covers the following Linux+ XK0-005 exam objective:

- **2.3:** Given a scenario, implement and configure firewalls.

One of the major components of securing a system or a network is to keep the "bad guys" out while letting the "good guys" have the necessary access. This means you need to make sure that network access to your systems is secure and stable.

Firewalls enable you to allow or block network connections. A firewall inspects each network packet and determines if the packet should be allowed in. In this chapter you will learn the essentials of creating firewalls.

This chapter provides information on the following topics: firewall use cases, common firewall technologies, and key firewall features.

Firewall Use Cases

A *firewall* (also known as an access control list) is a network appliance (either hardware or software based) that is designed to either allow or block network traffic. Firewalls can be implemented on a variety of devices, including routers, network servers, and users' systems. This section explores the basic concepts of firewalls.

The following are some firewall-related terms you should be aware of:

- **Source:** This term refers to where a network packet originated. Firewall rules can be created to filter network traffic from the source by using the source's IP address, port, or MAC address.

- **Destination:** This term refers to where a network packet is being sent. Firewall rules can be created to filter network traffic for the destination by using either the destination's IP address, port, or MAC address.
- **Port:** A port has a unique number that is used to address a service on a system. Each network packet includes a source port and a destination port. Firewall rules can be created to filter network traffic by using the source or destination port number.
- **Protocol:** It is common to filter packets by protocol—either a protocol like ICMP, TCP, or UDP or a protocol associated with a specific port (such as Telnet, which uses port 23).
- **Log:** Firewall rules are normally used to allow or block a network packet; however, there are also rules that are designed to log information regarding a network packet. This is useful when you want to see information about a packet that you will block with a later rule.

Open and Close Ports

An *open port* allows a network packet through the firewall. A *close port* blocks a network packet (or returns it to the sender). The firewall determines which action to take based on the rules created for the firewall, including what criteria to match (based on port number, source, destination, and so on) and what to do with the packet, which is referred to as the *target*.

When a network packet matches the criteria of a firewall rule, a target is used to determine what action to take. There are four standard targets for **iptables** (the core firewall on Linux):

- **Accept:** The Accept target tells **iptables** to allow the packet to advance to the next filtering point.
- **Reject:** The Reject target tells **iptables** to return the packet to the source with an error message. The message is not advanced to the next filtering point.
- **Drop:** The Drop target tells **iptables** to discard the packet. No response is sent to the source, and the message is not advanced to the next filtering point.
- **Log:** The Log target tells **iptables** to create a log entry that contains detailed information about the packet. With this target, the packet is neither explicitly allowed nor blocked, but further rules in the rule set (the chain) may determine this result.

Check Current Configuration

The **-L** option to the **iptables** command can be used to check the current configuration of the firewall. See the "Configuring the Host Firewall" section in Chapter 8, "Security Best Practices in a Linux Environment," for details.

Enable/Disable Internet Protocol (IP) Forwarding

IP forwarding is a kernel feature that allows network packets to be passed from one network to another. This feature is commonly used to create a router on a server. This is a simple matter, as you just need to set the value of each of the following files to **1**:

- **/proc/sys/net/ipv4/ip_forward:** This enables IP forwarding for IPv4 network packets.
- **/proc/sys/net/ipv6/conf/all/forwarding:** This enables IP forwarding for IPv6 network packets.

Common Firewall Technologies

A large number of firewall technologies can be implemented on Linux systems. This section focuses on the technologies included on the Linux+ XK0-005 exam.

firewalld

On Red Hat–based distributions, it is common to use a utility called **firewalld** to configure **iptables** rules. Rules are configured into categories called "zones," and the rules are managed using the **firewall-cmd** command. For example, you can execute the command **firewall-cmd --get-zones** to display all available zones, as in this example:

```
root@OCS:~# firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

You can set different rules on each zone and assign zone rule sets to different interfaces. For example, if your **eth0** interface connects to your internal network, you can apply the "trusted" or "internal" zone, which should have

less-restrictive rules. Conversely, your **eth1** interface might connect to the Internet, which means the "dmz" or "external" zone might be a better solution.

Rules created at the command line using the **firewall-cmd** command affect the active firewall on the system. This is referred to as the *runtime firewall*. If the system is rebooted or the firewall service is restarted, these rules are lost.

To make runtime rules persist after a system or firewall reset, use the following command:

```
root@OCS:~# firewall-cmd --runtime-to-permanent
```

iptables

For more information about **iptables**, see the "Configuring the Host Firewall" section in Chapter 8.

nftables

The Netfilter project, also known as **nftables**, is a newer firewall technology that is designed to address some of the shortcomings of **iptables**. The technology behind **iptables** has been around for decades, and while **iptables** is still very much in use, **nftables** is starting to become more popular.

To learn how **nftables** is different from **iptables**, see https://wiki.nftables.org/wiki-nftables/index.php/What_is_nftables%3F#Main_differences_with_iptables.

ExamAlert

This chapter provides little information about **nftables** due to the fact that it is newer and not in widespread use. For the Linux+ XK0-005 exam, you should be aware of what it is, but you shouldn't expect to see complicated questions on this technology.

Uncomplicated Firewall (UFW)

On Debian-based distributions, you will likely have the ability to use **ufw** (which stands for "uncomplicated firewall"). Much like **firewalld**, this utility acts as a front-end interface for creating **iptables** rules.

For example, the following command creates a rule to allow for both inbound and outbound SSH connections:

```
root@OCS:~# ufw allow ssh
```

The **ufw** firewall rules are stored in the following files:

- **/etc/default/ufw**
- **/etc/ufw**

Note that the **/etc/default/ufw** rules should not be modified manually. System administrators can use the **/etc/ufw** file to modify and override the default rules from **/etc/default/ufw**.

Key Firewall Features

This section describes the features of firewalls that are mentioned in the exam objectives; other sections of this chapter cover additional firewall features.

Zones

For information about zones, see the "**firewalld**" section, earlier in this chapter.

Services

It is common to filter packets by protocol—either a protocol like ICMP, TCP, or UDP or a protocol associated with a specific port (such as Telnet, which uses port 23, or SSH, which is associated with port 22).

You can use the **/etc/services** file to look up which ports are usually associated with a specific service. For example, the following command displays the ports that are normally assigned to the mail service:

```
[root@mail ~]# grep ^smtp /etc/services
smtp            25/tcp          mail
smtp            25/udp          mail
```

ExamAlert

While the Linux+ XK0-005 exam doesn't specifically talk about knowing specific port numbers, keep in mind that the "Recommended experience" section of the exam blueprint states "12 months of hands-on experience working with Linux servers, as well as A+, Network+, and Server+ or similar certifications and/or knowledge." Those exams do require knowledge of specific port numbers and the names of the services they are associated with. As a result, you may want to examine the exam blueprints of those certifications and make sure you are familiar with the ports listed in those exam objectives.

Stateful/Stateless

A stateful rule is a rule that applies to any previously established connection. Consider a firewall that specifies the following:

- Allow most outbound connections, including to websites, SSH access, FTP sites, and so on.
- Block most inbound connections so outside users can't connect to services you may have running.

The problem with this scenario is that when you connect to a remote system, both outbound and inbound rules apply. The outbound rules apply when you establish a connection, and the inbound rules apply when the remote server responds. If you have a rule that blocks most inbound connections, you may never receive the response from that website you are trying to visit.

You can overcome this by creating stateful rules. Stateful rules essentially say, "if a network packet is responding to a request that the local machine initiated, let it through the firewall."

A stateless rule is a rule that applies regardless of any previously established connection.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which firewall target returns the network packet to the original sender?
   - ❍ **A.** Resend
   - ❍ **B.** Drop
   - ❍ **C.** Reject
   - ❍ **D.** Refuse

2. Which option to the **iptables** command displays the firewall rules?
   - ❍ **A. -L**
   - ❍ **B. -D**
   - ❍ **C. -A**
   - ❍ **D. -I**

3. Which file is used to forward IPv6 network packets?
   - ❍ **A. /proc/sys/net/ipv6/conf/all/forwarding**
   - ❍ **B. /proc/sys/net/ipv6/conf/all/ip_forward**
   - ❍ **C. /proc/sys/net/ipv6/forwarding**
   - ❍ **D. /proc/sys/net/ipv6/ip_forward**

4. Which directories are used to store **ufw** firewall rules? (Choose two.)
   - ❍ **A. /etc/default/ufw**
   - ❍ **B. /etc/ufw/default**
   - ❍ **C. /etc/ufw/rules.d**
   - ❍ **D. /etc/ufw**

Cram Quiz Answers

1. **C.** The Reject target tells **iptables** to return the packet to the source with an error message. The Drop target tells **iptables** to discard the packet. No response is sent to the source, and the message is not advanced to the next filtering point. The other answers are not valid firewall targets.
2. **A.** The **-L** option lists firewall rules. The **-D** option deletes firewall rules. The **-A** option adds a firewall rule, and the **-I** option inserts a new firewall rule.
3. **A.** The **/proc/sys/net/ipv6/conf/all/forwarding** file enables IP forwarding for IPv6 network packets. The other answers are not valid files for IP forwarding.
4. **A and D.** The **ufw** firewall rules are stored in the **/etc/default/ufw** and **/etc/ufw** directories. The other answers are not valid locations for **ufw** rules.

This page intentionally left blank

CHAPTER 11

Configure and Execute Remote Connectivity for System Management

This chapter covers the following Linux+ XK0-005 exam objective:

- **2.4:** Given a scenario, configure and execute remote connectivity for system management.

This chapter covers configuration of SSH—both the client and the server utilities—and establishment of SSH tunnels.

This chapter also discusses privileged access to commands and system services. It explores three technologies in particular: **sudo**, PolicyKit, and the **su** utility.

This chapter provides information on SSH as well as on executing commands as another user.

SSH

The **ssh** command is a utility that allows you to connect to a Secure Shell (SSH) server. The syntax of this command is as follows:

```
ssh user@hostname
```

where *user* is the username you want to use for logging in and *hostname* is a system hostname or IP address.

The first time you use the **ssh** command to connect to a system, you get the following prompt:

```
[root@OCS ~]# ssh bob@server1
The authenticity of host 'server1' can't be established.
ECDSA key fingerprint is
```

```
 8a:d9:88:b0:e8:05:d6:2b:85:df:53:10:54:66:5f:0f.
Are you sure you want to continue connecting (yes/no)?
```

This prompt ensures that you are logging in to the correct system. Typically users answer **yes** to this prompt, assuming that they are logging in to the correct machine, but this information can also be verified independently by contacting the system administrator of the remote system.

After the user answers **yes** to this prompt, the SSH server fingerprint is stored in the **known_hosts** file in the **~/.ssh** directory.

Table 11.1 describes common options for the **ssh** command.

TABLE 11.1 **ssh Command Options**

Option	Description
-F *configfile*	Specifies the configuration file to use for the **ssh** client utility. The default configuration file is **/etc/ssh/ssh_config**. See the "**/etc/ssh/ssh_config**" section, later in this chapter, for details regarding this file.
-4	Indicates to use only IPv4 addresses.
-6	Indicates to use only IPv6 addresses.
-E *logfile*	Places errors in the specified log file rather than displaying them to standard output.

~/.ssh/known_hosts

After a connection is established with an SSH server, the SSH client stores the server's unique fingerprint key in the user's **~/.ssh/known_hosts** file, as shown here:

```
[root@OCS ~]# cat !/.ssh/known_hosts
|1|trm4BuvRf0HzJ6wusHssj6HcJKg=|EruYJY709DXorogeN5Hdcf6jTCo=
 ecdsa-sha2-nistp256AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzd
 HAyNTYAAABBBG3/rARemyZrhIuirJtfpfPjUVnph9S1w2NPfEWec/f59V7nA
 ztn5rbcGynNYOdnozdGNNizYAiZ2VEhJ3Y3JcE=
```

Typically the contents of this file should be left undisturbed; however, if the SSH server is reinstalled, it then has a new fingerprint key, and all users must remove the entry for the SSH server from the **.ssh/known_hosts** file.

~/.ssh/authorized_keys

To create authentication keys, you use the **ssh-keygen** command. See the "**ssh-keygen**" section, later in this chapter, for details regarding this process.

After the authentication key files have been created, the public key (in either the **~/.ssh/id_dsa.pub** file or the **~/.ssh/id_rsa.pub** file) needs to be copied to the system that the user is attempting to log in to. This requires placing the public key into the **~/.ssh/authorized_keys** file on the SSH server. This can be accomplished by manually copying the content of the public key file from the client system and pasting it into the **~/.ssh/authorized_keys** file on the SSH server. Alternatively, you can use the **ssh-copy-id** command, which has the following syntax:

```
ssh-copy-id user@server
```

/etc/ssh/sshd_config

The **sshd_conf** file is used to configure the SSH server. Important settings in this file include the following:

- **Protocol:** Defines the protocol type for the SSH server (set to either **1**, **2**, or **1,2**,). Protocol 1 is no longer supported, so this value normally should be set to **2**.
- **ListenAddress:** Specifies the IP address assigned to the network cards that SSH should accept connections on. Example: **ListenAddress 192.168.1.100:192.168.1.101**.
- **Port:** The port on which the SSH server listens for inbound connections. Example: **Port 2096**.
- **LogLevel:** The logs you want the SSH server to record. The following values are permitted:
  - **QUIET**
  - **FATAL**
  - **ERROR**
  - **INFO**
  - **VERBOSE**
  - **DEBUG**
  - **DEBUG1** (same as **DEBUG**)
  - **DEBUG2**

- **DEBUG3**

- **PermitRootLogin:** When set to **no**, prevents the root user from logging in directly via SSH.
- **AllowUsers or DenyUsers:** Defines what users can log in via SSH (**AllowUsers**) or which users cannot login via SSH (**DenyUsers**). Example: **AllowUsers bob sue ted**.
- **PasswordAuthentication:** When set to **yes** (the default), allows users to log in by providing their username and password. When set to **no**, allows users to log in using an authentication key only.
- **PubkeyAuthentication:** When set to **yes**, allows a user to store a public key on the server. The public key is generated by the **ssh-keygen** command on the user's client system. See the "**ssh-keygen**" section, later in this chapter, for details.
- **Banner:** Specifies a file whose contents are displayed prior to the user authentication process.
- **PrintMotd:** When set to **yes** (the default), prevents the contents of the **/etc/motd** file from being displayed when a user logs in to the system via SSH.

/etc/ssh/ssh_config

The **/etc/ssh/ssh.conf** file is used to modify the behavior of the SSH client utilities, such as **ssh**, **scp**, and **sftp**. This file affects all users, but users can override these settings by creating their own configuration files in their home directory (the **~/.ssh/config** file).

A few components of the **ssh_config** file are different from the components of the **sshd_config** file. (Consider reviewing the preceding section, "**/etc/ssh/sshd_config**," before continuing.) To begin with, there is the systemwide **/etc/ssh/ssh_config** file, which applies to all users. In addition, each user can create a file in their home directory (**~/.ssh/config**) that can be used to override the settings in the **/etc/ssh/ssh_config** file.

Command-line options can override the values specified in the configuration files. Here is the order in which all this information is parsed:

1. Command-line options
2. The user's **~/.ssh/config** file
3. The **/etc/ssh/ssh_config** file

The first parameter found is the one used. For example, if **ConnectTimeout** is set in the user's **~/.ssh/config** file and a different value is set in the **/etc/ssh/ssh_config** file, the user's configuration file is used to set this value.

An important difference between the **ssh_config** file and the **sshd_config** file is that most of the settings in the **ssh_config** file are subsettings of the **Host** setting. The **Host** setting allows you to specify different rules for different SSH servers you are connecting to. For example, the following would apply the **ConnectTimeout** value **0** when connecting to server1.localhost.com and the value **600** when connecting to test.example.com:

```
Host server1.localhost.com
ConnectTimeout 0
Host test.example.com
ConnectTimeout 600
```

Many of the settings in the **ssh_config** file are related to settings in the SSH server configuration file, such as the following **/etc/ssh/sshd_config** file setting:

```
X11Forwarding yes
```

On the client side, this feature is typically enabled by using the **-X** or **-Y** option when executing **ssh**-based commands. However, if you want this feature to be the default, the **ForwardX11** and **ForwardX11Trusted** settings could be used in either the **/etc/ssh/ssd_config** file or the **~/.ssh/config** file.

~/.ssh/config

Users can customize how commands like **ssh**, **scp**, and **sftp** work by creating the **~/.ssh/config** file. The format and settings in this file are the same as those found in the **/etc/sshd/ssh.conf** file. See the "**/etc/ssh/ssh_config**" section, earlier in this chapter, for details.

ssh-keygen

The **ssh-keygen** command can be used to generate authentication keys. A common use for this command is to create the authentication files used for passwordless authentication. Here is an example:

```
[julia@OCS ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/julia/.ssh/id_rsa):
```

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/julia/.ssh/id_rsa.
Your public key has been saved in /home/julia/.ssh/id_rsa.pub.
The key fingerprint is:
6d:40:24:e2:3e:62:7f:a4:f0:5d:d5:05:1b:06:fd:d7 bo@ubuntu
The key's randomart image is:
+--[ RSA 2048]----     +
|     . ..o .o+.. |
| . . o.+          |
| .    . . o. .|
| . +    . E|
| + o . S o     . |
| . = = . .     |
| + o   |
| .     |
|       |
+-----------------   +
```

This command creates two new files: **~/.ssh/id_rsa** and **~/.ssh.id_rsa.pub**. The contents of these files are used for password authentication, in conjunction with the **~/.ssh/authorized_keys** file.

RSA is an encryption algorithm. Another popular encryption algorithm is DSA. To use DSA instead of RSA, use the **-t DSA** option to the **ssh-keygen** command. The result of using this option is two new files: **~/.ssh/id_dsa** and **~/.ssh.id_dsa.pub**.

ExamAlert

If a passphrase is requested during the execution of the **ssh-keygen** command, this passphrase needs to be entered in place of a password whenever an SSH connection is established. Repeatedly entering this passphrase can be avoided by using the **ssh-agent** and **ssh-add** utilities. See the "**ssh-add**" section, later in this chapter, for details regarding these utilities.

ssh-copy-id

The **ssh-copy-id** command is used to copy the login keys that are created by the **ssh-keygen** command to a remote system. (See also the "ssh-keygen" section, earlier in this chapter.) The format for this command is as follows:

```
ssh-copy-id user@hostname
```

ssh-add

The **ssh-agent** utility enables you to avoid needing to enter a passphrase whenever passwordless SSH authentication is used. See the "**ssh-keygen**" section, earlier in this chapter, for details regarding setting up passwordless SSH authentication. One way to use this feature is to execute the following command:

```
ssh-agent bash
```

A shell is started in which the SSH agent caches RSA and DSA encryption keys. After the **ssh-agent** utility has been started, the **ssh-add** command can be used to add RSA and DSA encryption keys to the SSH agent's cache.

To use the **ssh-add** utility, execute the command with no arguments, as in this example:

```
[julia@OCS ~]$ ssh-add
Identity added: /home/julia/.ssh/id_rsa (/home/julia/.ssh/id_rsa)
```

After the keys have been added, they are automatically used in future SSH connections.

Tunneling

SSH port forwarding is often called *SSH tunneling*. This technique provides the means to create a secure connection between two systems using software provided by SSH. This section explores several commonly used SSH port-forwarding techniques.

X11 Forwarding

When you connect to a remote server via SSH, you can execute commands and see the output of the commands in your local terminal. However, if you attempt

to execute any program that results in GUI output, the X server on your local system refuses to display the output.

An X11 forward tunnel gives all the SSH client programs the ability to receive data from the GUI-based program to display on the local X server. This just requires including the **-X** option when connection occurs via the **ssh** command.

Port Forwarding

When the SSH forwarding process originates from the client machine, this is referred to as *local port forwarding*. In this scenario, an outgoing port on the local system is configured to connect via SSH to a specific port on a remote system. This is typically configured on the local system with a command like the following:

```
ssh -L 9000:onecoursesource.com:80 localhost
```

Now any connections from the SSH client are forwarded *via the SSH server* to the destination server.

While the command to create a remote SSH tunnel looks very much like the command to establish a local SSH tunnel, the results are a bit different:

```
sh -R 9001:localhost:9001 name@remote
```

Now any connections from the SSH client are forwarded *via the SSH client* to the destination server.

ExamAlert

Port forwarding normally requires a change to the **/etc/ssh/sshd_config** file as the **GatewayPorts** value must be set to **yes**.

Dynamic Forwarding

Dynamic forwarding uses the SOCKS protocol (where SOCKS is an abbreviation of the word *sockets*) to establish a tunnel between a browser and a server. On the server, a command like the following is executed to establish the dynamic tunnel:

```
ssh -D <port_number> <domain>
```

You can configure the connection by using settings in a web browser, as shown with Firefox in Figure 11.1.

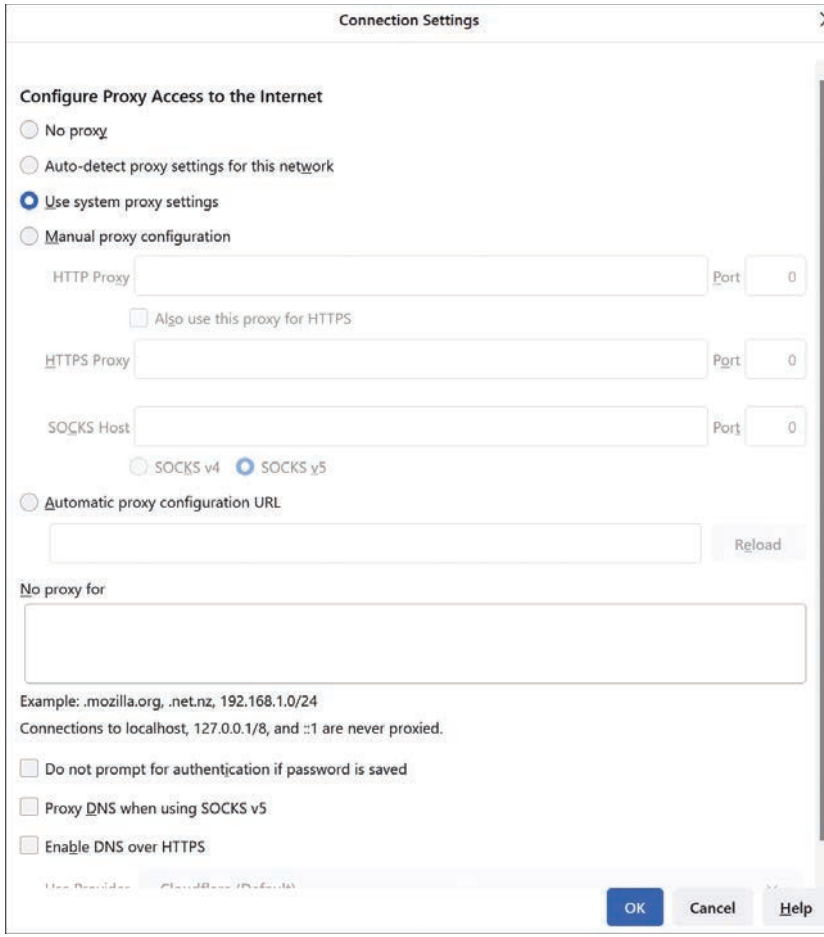

FIGURE 11.1 **Client-Side Dynamic Port Configuration in the Firefox Web Browser**

Executing Commands as Another User

Privilege escalation is a way of executing commands as another user that has elevated privileges. It is used, for example, when a user needs to be able to execute commands using an account that has more privileges than the user's account normally has. For example, a regular user may need to execute a command that requires root user access. Several techniques can provide privilege access; this section covers the techniques that may be included on the Linux+ XK0-005 exam.

/etc/sudoers

For information about **/etc/sudoers**, see the "**sudo**" and "**visudo**" sections, later in this chapter.

PolicyKit Rules

PolicyKit is an authorization framework which allows the operations that are considered privileged to be performed by unprivileged programs. The rules for PolicyKit are stored in files in the following directories:

- **/etc/polkit-1/rules.d** (Red Hat–based systems)
- **/etc/polkit-1/localauthority** (Debian-based systems)
- **/usr/share/polkit-1/rules.d** (third-party software rules)

Rules can be complex to write. For example, the following rule allows a user to mount a filesystem via the GUI interface if that user is a member of the filesystem group:

```
polkit.addRule(function(action, subject) {
    if (action.id == "org.freedesktop.udisks2.filesystem-mount-system"
&&
        subject.isInGroup("filesystem")) {
        return polkit.Result.YES;
    }
});
```

Note

In this example, **org.freedesktop.udisks2.filesystem-mount-system** is an application programming interface (API) call, and this file is written in JavaScript.

ExamAlert

While it is possible that the Linux+ XK0-005 exam will present a PolicyKit rule and ask you a question on it, this is unlikely as the rules are considered somewhat complex and require some JavaScript knowledge. It is more likely that you will be asked a general question about how rules are used to provide privileged access to services via APIs.

sudo

When the sudo command is properly configured by the administrator, users can use the **sudo** command to run commands as other users (typically as the root user). To execute a command as root, enter the following:

```
sudo command
```

You are then prompted for your own password and, if the settings in the **/etc/sudoers** file are correct, the command executes correctly. If the settings are not correct, an error message appears.

Table 11.2 describes common options for the **sudo** command.

TABLE 11.2 **sudo Command Options**

Option	Description
-b	Runs the command in the background.
-e	Allows you to edit the **/etc/sudoers** file. This is an alternative to the visudo utility.
-l	Lists which commands are allowed for the user.
-u *user*	Runs the command as *user* rather than as the root user.

See the following section, "visudo," for details regarding the **/etc/sudoers** file.

visudo

The **/etc/sudoers** file is used to determine which users can use the **sudo** command to execute commands as other users (typically as the root user). To edit this file, you must be logged in as the root user, and you should use the **visudo** command rather than edit the file directly.

Table 11.3 describes important options for the **/etc/sudoers** file.

TABLE 11.3 **/etc/sudoers File Definitions**

Option	Description
User_Alias	Represents a group of users (for example, **User_Alias ADMINS = julia, sarah**).
Cmnd_Alias	Represents a group of commands (for example, **Cmnd_Alias SOFTWARE = /bin/rpm, /usr/bin/yum**).

An entry in the **/etc/sudoers** file uses the following syntax:

```
user machine=commands
```

To allow a student user the ability to execute the **/usr/bin/yum** command as the root user, add an entry like the following to the **/etc/sudoers** file:

```
student ALL=/usr/bin/yum
```

To allow all members of **ADMINS** to execute all of the **SOFTWARE** command as the root user, add an entry like the following to the **/etc/sudoers** file:

```
ADMINS ALL=SOFTWARE
```

su -

The **su** command allows a user to shift user accounts. Here is an example:

```
[student@OCS ~]# id
uid=1000(student) gid=1000(student) groups=1000(student)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[student@localhost ~]# su root
Password:
[root@OCS ~]# id
uid=0(root) gid=0(root) groups=0(root)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

One option is permitted when executing the **su** command: the **-** option. When you execute the **su** command with the **-** option, a new login shell is provided. When you're not using the **-** character, a non-login shell is provided.

ExamAlert

Be clear about the differences between using and not using **-** with the **su** command. The Linux+ XK0-005 exam objectives specifically state that you should know **su -**, which means you are likely going to see a question that will require you to know the difference.

pkexec

Use the **pkexec** command to execute privileged commands that have been configured via PolicyKit. This command is executed using the following syntax:

```
pkexec command
```

As with the **sudo** command, with the **pkexec** command, you should be asked to provide your account password to confirm your identity.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. After a connection is established with an SSH server, the SSH client stores the server's unique fingerprint key in the user's _____ file.
   - ❍ **A. .ssh/hosts**
   - ❍ **B. .ssh/known_host**
   - ❍ **C. .ssh/known**
   - ❍ **D. .ssh/known_hosts**

2. Which file is used to configure the SSH server?
   - ❍ **A. /etc/ssh/ssh_config**
   - ❍ **B. /etc/ssh/sshd_config**
   - ❍ **C. ~/.ssh/config**
   - ❍ **D. /etc/ssh/sshd_configd**

3. Which files are created by the **ssh-keygen** utility when using RSA as the encryption algorithm? (Choose two.)
   - ❍ **A. ~/.ssh/id_rsa.key**
   - ❍ **B. ~/.ssh/id_rsa.pri**
   - ❍ **C. ~/.ssh/id_rsa**
   - ❍ **D. ~/.ssh.id_rsa.pub**

4. Which utility should be used to modify the **/etc/sudoers** file?
   - ❍ **A. sudo**
   - ❍ **B. visudo**
   - ❍ **C. editsudo**
   - ❍ **D. vi**

Cram Quiz Answers

1. **D.** After a connection is established with an SSH server, the SSH client stores the server's unique fingerprint key in the user's **.ssh/known_hosts** file. The other answers are not valid files for SSH.
2. **B.** The **/etc/ssh/sshd_config** file is used to configure the SSH server. The **/etc/ssh/ssh_config** and **~/.ssh/config** files are used to configure SSH client utilities. The last answer is not a valid SSH configuration file.
3. **C and D.** The result of running the **ssh-keygen** command using RSA encryption is two new files: **~/.ssh/id_rsa** and **~/.ssh.id_rsa.pub**. The other answers are not valid files for the **ssh-keygen** command.
4. **B.** To edit the **/etc/sudoers** file, you must be logged in as the root user and should use the **visudo** command rather than edit the file directly.

CHAPTER 12

Apply the Appropriate Access Controls

This chapter covers the following Linux+ XK0-005 exam objective:

- **2.5:** Given a scenario, apply the appropriate access controls.

Security is an important focus in IT today, and knowing how to secure the file-system is critical to a solid IT security plan. In this chapter you will learn how to secure files and directories using permissions, including access control lists, **suid**, **sgid**, and the sticky bit. You will also learn how to implement context-based security and be introduced to SELinux and AppArmor.

This chapter provides information on the following topics: file permissions, Security-Enhanced Linux (SELinux), AppArmor, and command-line utilities.

File Permissions

A user who owns a file or directory has the ability to allow or deny access to the file or directory by using permissions. In addition, the root user has the ability to change the permissions of any file or directory on the system.

Every file and directory has standard permissions (also called "read, write, and execute" permissions) that either allow or disallow a user access. Using these standard permissions is something that every Linux user should understand how to do, as this is the primary way a user can protect files from other users.

To view the permissions of a file or directory, use the **ls -l** command:

```
[student@OCS ~]$ ls -l /etc/chrony.keys
-rw-r-----. 1 root chrony 62 May 9 2018 /etc/chrony.keys
```

The first 10 characters of the output denote the file type and the permissions for the file. (Recall that a **-** [hyphen] as the character in the first position denotes a plain file, whereas a **d** denotes a directory.) Permissions are broken into three sets: the user owner of the file (**root** in the previous example), the group owner (**chrony**), and all other users (referred to as "others").

Each set has three possible permissions: read (**r**), write (**w**), and execute (**x**). If the permission is set, the character that symbolizes the permission is displayed. Otherwise, a hyphen (**-**) character is displayed to indicate that a permission is not set. Thus, **r-x** means "read and execute are set, but write is not set."

What read, write, and execute permissions really mean depends on whether the object is a file or directory. For files, these permissions mean the following:

- **Read:** Can view or copy file contents.
- **Write:** Can modify file contents.
- **Execute:** Can run the file like a program. After you create a program, you must make it executable before you can run it.

For directories, these permissions mean the following:

- **Read:** Can list files in the directory.
- **Write:** Can add and delete files in the directory (requires execute permission).
- **Execute:** Can change into the directory (using the **cd** command) or use it in a pathname.

Access Control List (ACL)

See the "**setfacl/getfacl**" section, later in this chapter.

Set User ID (SUID), Set Group ID (SGID), and Sticky Bit

Table 12.1 describes the special permission sets **suid**, **sgid**, and the sticky bit.

TABLE 12.1 **suid/sgid/sticky bit Special Permission Sets**

Permission	Description	Set	Remove
suid	When set on executable files, **suid** allows a program to access files using the permissions of the user owner of the executable file.	**chmod u+s** *file* or **chmod 4***xxx file* (where *xxx* refers to regular read, write, and execute permissions)	**chmod u-s** *file* or **chmod 0***xxx file*
sgid	When set on executable files, **sgid** allows a program to access files using the permissions of the group owner of the executable file. When it's set on directories, all new files in a directory inherit the group ownership of the directory.	**chmod g+s** *file* or **chmod 2***xxx file*	**chmod g-s** *file* or **chmod 0***xxx file*
Sticky bit	When the sticky bit is set on directories, files in the directory can only be removed by the user owner of a file, the owner of a directory, or the root user.	**chmod o+t** *file* or **chmod 1***xxx file* Note: Sticky bit permissions are almost always set to the octal value 1777.	**chmod o-t** *file* or **chmod 0***xxx file*

Security-Enhanced Linux (SELinux)

Files and directories may be compromised by users who either do not understand permissions or who accidently provide more access than intended. System administration used to often say, "If we didn't have users, nothing would break, and the system would be more secure." Of course, the response to this saying is, "Without users, we wouldn't have a job!" Users' mistakes often do provide unintended access to the data that is stored in files.

Note that traditional Linux permissions make use of Discretionary Access Control (DAC), while context-based permissions use Mandatory Access Control (MAC). However, when a context-based solution is enabled, DAC still applies (that is, both MAC and DAC are enforced).

Context-based permissions can be configured to compensate for this flaw by providing an additional level of security when processes (programs) are used to

access files. This section and the next cover two commonly used context-based methods: SELinux and AppArmor.

Context Permissions

With SELinux, you can apply a security policy that requires processes to be part of an SELinux security context (which is basically a security group) in order to allow access to files and directories. Regular permissions are still used to further define access, but for accessing a file/directory, the SELinux policy would be applied first.

A bigger concern—and one that most SELinux policies are designed to address—is how daemon (or system) processes present security risks. Consider a situation in which you have many active processes that provide a variety of services. For example, one of these processes might be a web server, as shown in the following example:

```
root@OCS:~# ps -fe | grep httpd
root   1109  1    0 2018 ?  00:51:56 /usr/sbin/httpd
apache 1412  1109 0 Dec24 ? 00:00:09 /usr/sbin/httpd
apache 4085  1109 0 05:40 ? 00:00:12 /usr/sbin/httpd
apache 8868  1109 0 08:41 ? 00:00:06 /usr/sbin/httpd
apache 9263  1109 0 08:57 ? 00:00:04 /usr/sbin/httpd
apache 12388 1109 0 Dec26 ? 00:00:47 /usr/sbin/httpd
apache 18707 1109 0 14:41 ? 00:00:00 /usr/sbin/httpd
apache 18708 1109 0 14:41 ? 00:00:00 /usr/sbin/httpd
apache 19769 1109 0 Dec27 ? 00:00:15 /usr/sbin/httpd
apache 29802 1109 0 01:43 ? 00:00:17 /usr/sbin/httpd
apache 29811 1109 0 01:43 ? 00:00:11 /usr/sbin/httpd
apache 29898 1109 0 01:44 ? 00:00:10 /usr/sbin/httpd
```

In this output, note that each line describes one Apache web server process (**/usr/sbin/httpd**) that is running on the system. The first part of the line is the user who initiated the process. The process that runs as root is only used to spawn additional **/usr/sbin/httpd** processes. The others, however, respond to incoming web page requests from client utilities (web browsers).

Imagine for a moment that a security flaw is discovered in the software for the Apache web server that allows a client utility to gain control of one of the **/usr/sbin/httpd** processes and issue custom commands or operations to that

process. One of those operations could be to view the content of the **/etc/passwd** file, which would be successful because of the permissions placed on this file:

```
root@OCS:~# ls -l /etc/passwd
-rw-r--r-- 1 root root 2690 Dec 11 2018 /etc/passwd
```

As you can see from the output of this command, all users have the ability to view the contents of the **/etc/passwd** file. But do you really want some random person (perhaps a hacker) to have the ability to view the contents of the file that stores user account data?

With an SELinux policy, the **/usr/sbin/httpd** processes can be locked down so each process can access only a certain set of files. This is what most administrators use SELinux for: to secure processes to prevent them from being compromised by hackers making use of known (or, perhaps, unknown) exploits.

Labels

Labeling a file in SELinux involves applying a security context to the file. See the "**chcon**" section, later in this chapter, for information on security contexts.

Autorelabel

If you want to change the SELinux context for all of the files on a system, you can tell the system to perform an autorelabel process on every file in the filesystem during the next boot process. To do this, create an empty file in the root directory as shown here:

```
root@OCS:~# touch /.autorelabel
```

System Booleans

See the "**getsebool**" section, later in this chapter, for details about SELinux Booleans.

States

SELinux can be in one of three states (or modes):

- **Enforcing:** When in enforcing mode, SELinux performs checks and blocks access to files or directories if necessary.

- **Permissive:** When in permissive mode, SELinux performs checks but never blocks access to a file or directory. This mode is designed for troubleshooting problems as log messages are created when in this mode.
- **Disabled:** When in disabled mode, SELinux is not functional at all. No checks are performed when users attempt to access files or directories.

> **Note**
>
> See the "**setenforce**" and "**getenforce**" sections, later in this chapter, for more details on viewing and changing the SELinux mode.

Policy Types

An SELinux policy is a collection of rules that determine what restrictions are imposed by the policy. A policy itself is often very complex, and details of SELinux policies are beyond the scope of the Linux+ XK0-005 exam. It is, however, important to know that a policy sets restrictions based on rules.

You should also know that one of the most commonly used policies is the targeted policy. This policy normally exists by default on systems that have SELinux installed, and it is typically the default policy that is enabled when SELinux is first started.

Targeted

A targeted policy contains rules designed to protect a system from services, rather than from regular users. Each service is assigned one or more security contexts, Boolean values, and additional rules that limit the service's ability to access files and directories.

Minimum

SELinux includes a target that contains only the minimum security settings. This is called the minimum target.

AppArmor

AppArmor is a MAC system that plays a similar role to SELinux in that it provides a context-based permission model. This section describes the key components of AppArmor that the Linux+ XK0-005 exam covers.

An AppArmor profile is a rule set that describes how AppArmor should restrict a process. A profile can be disabled for a specific service by using the **aa-disable** command. Here's an example:

```
root@OCS:~# aa-disable /usr/sbin/mysqld
```

To view the status of a profile, use the **aa-status** command.

```
root@OCS:~# aa-status
apparmor module is loaded.
19 profiles are loaded.
17 profiles are in enforce mode.
   /usr/lib/NetworkManager/nm-dhcp-client.action
   /usr/lib/NetworkManager/nm-dhcp-helper
   /usr/lib/connman/scripts/dhclient-script
   /usr/lib/cups/backend/cups-pdf
   /usr/lib/lightdm/lightdm-guest-session
   /usr/lib/lightdm/lightdm-guest-session//chromium
   /usr/sbin/cups-browsed
   /usr/sbin/cupsd
   /usr/sbin/cupsd//third_party
   /usr/sbin/tcpdump
   /{,usr/}sbin/dhclient
   libreoffice-senddoc
   libreoffice-soffice//gpg
   libreoffice-xpdfimport
   lsb_release
   nvidia_modprobe
   nvidia_modprobe//kmod
2 profiles are in complain mode.
   libreoffice-oopslash
   libreoffice-soffice
```

```
5 processes have profiles defined.
5 processes are in enforce mode.
   /usr/sbin/cups-browsed (591)
   /usr/sbin/cupsd (495)
   /usr/lib/cups/notifier/dbus (592) /usr/sbin/cupsd
   /usr/lib/cups/notifier/dbus (593) /usr/sbin/cupsd
   /usr/lib/cups/notifier/dbus (594) /usr/sbin/cupsd
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

If you need to troubleshoot an AppArmor profile, it is best to put it into complain mode. In this mode, no restrictions are enforced, but any problems are reported.

Use the **aa-complain** command to put a profile into complain mode:

```
root@OCS:~# aa-complain /usr/sbin/mysqld
Setting /usr/sbin/mysqld to complain mode.
```

To put the profile back into the enforcing mode, you can use the following commands:

```
chief@chief-VirtualBox:~$ sudo aa-enforce /usr/bin/man
Setting /usr/bin/man to enforce mode.
chief@chief-VirtualBox:~$ sudo aa-disable /usr/bin/man
Disabling /usr/bin/man.
chief@chief-VirtualBox:~$ sudo aa-status
apparmor module is loaded.
19 profiles are loaded.
17 profiles are in enforce mode.
   /usr/lib/NetworkManager/nm-dhcp-client.action
   /usr/lib/NetworkManager/nm-dhcp-helper
   /usr/lib/connman/scripts/dhclient-script
   /usr/lib/cups/backend/cups-pdf
   /usr/lib/lightdm/lightdm-guest-session
   /usr/lib/lightdm/lightdm-guest-session//chromium
   /usr/sbin/cups-browsed
   /usr/sbin/cupsd
   /usr/sbin/cupsd//third_party
```

```
   /usr/sbin/tcpdump
   /{,usr/}sbin/dhclient
   libreoffice-senddoc
   libreoffice-soffice//gpg
   libreoffice-xpdfimport
   lsb_release
   nvidia_modprobe
   nvidia_modprobe//kmod
2 profiles are in complain mode.
   libreoffice-oopslash
   libreoffice-soffice
5 processes have profiles defined.
5 processes are in enforce mode.
   /usr/sbin/cups-browsed (591)
   /usr/sbin/cupsd (495)
   /usr/lib/cups/notifier/dbus (592) /usr/sbin/cupsd
   /usr/lib/cups/notifier/dbus (593) /usr/sbin/cupsd
   /usr/lib/cups/notifier/dbus (594) /usr/sbin/cupsd
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
chief@chief-VirtualBox:~$ sudo aa-complain /usr/bin/man
Setting /usr/bin/man to complain mode.
chief@chief-VirtualBox:~$ sudo aa-enforce /usr/bin/man
Setting /usr/bin/man to enforce mode.
```

You can use the **aa-unconfined** command to list processes that are not restricted by the AppArmor profiles.

The **/etc/apparmor.d** directory is the location of the definitions of the AppArmor profiles. Knowing how to create or read these files is beyond the scope of the Linux+ XK0-005 exam, but it is important to know the location of these profiles in order to determine which profiles are available and to use the AppArmor commands, such as the **aa-disable** command.

The **/etc/apparmor.d/tunables** directory is the location of files that can be used to fine-tune the behavior of AppArmor. Knowing how to create or read these files is beyond the scope of the Linux+ XK0-005 exam.

Command-Line Utilities

This section covers several utilities related to permissions, SELinux, and AppArmor.

chmod

The **chmod** command is used to change permissions on files. It can be used in two ways: using the symbolic method and using the octal method. With the octal method, permissions are assigned numeric values:

- Read = 4
- Write = 2
- Execute = 1

With these numeric values, one number can be used to describe an entire permission set:

- 7 = **rwx**
- 6 = **rw-**
- 5 = **r-x**
- 4 = **r--**
- 3 = **-wx**
- 2 = **-w-**
- 1 = **--x**
- 0 = **---**

So, to change the permissions of a file to **rwxr-xr--**, you would execute the following command:

```
chmod 754 filename
```

Table 12.2 provides some examples using the octal method.

TABLE 12.2 **Examples Using the Octal Mode**

Example	Description
chmod 755 file	Sets the permissions **rwxr-xr-x**.
chmod 511 file	Sets the permissions **r-x--x--x**.
chmod 600 file	Sets the permissions **rw-------**.

With octal permissions, you should always provide three numbers, which will change all the permissions. But what if you only want to change a single permission of the set? For that you use the symbolic method by passing three values to the **chmod** command, as shown in Table 12.3.

TABLE 12.3 **Symbolic Mode Options**

Who	What	Permission
u = user owner	**+**	**r**
g = group owner	**-**	**w**
o = other	**=**	**x**
a = all sets		

The following example demonstrates the addition of execute permission to all three sets—user owner, group owner, and others—using the symbolic method:

```
[student@OCS ~]$ ls -l display.sh
-rw-rw-r--. 1 student student 291 Apr 30 20:09 display.sh
[student@OCS ~]$ chmod a+x display.sh
[student@OCS ~]$ ls -l display.sh
-rwxrwxr-x. 1 student student 291 Apr 30 20:09 display.sh
```

Table 12.4 describes some important options for the **chmod** command.

TABLE 12.4 **chmod Command Options**

Option	Description
-R	Recursively applies changes to an entire directory structure.
-v	Enters verbose mode and produces output demonstrating the changes made.

umask

The **umask** command sets default permissions for files and directories. These default permissions are applied only when a file or directory is initially created.

The **umask** command accepts one argument: the mask value. The mask value is an octal value that is applied against the maximum possible permissions for new files or new directories, as shown in Table 12.5.

TABLE 12.5 **Maximum Permissions for New Files/Directories**

Type	Maximum Possible Permission for a New Item
File	**rw-rw-rw-**
Directory	**rwxrwxrwx**

Figure 12.1 illustrates how the umask value of **027** would apply to new files compared to how it would apply to new directories.

Description	File			Directories		
Maximum	rw-	rw-	rw-	rwx	rwx	rwx
Umask Applied	---	-M-	MM-	---	-M-	MM-
Result	rw-	r--	---	rwx	r-x	--x

FIGURE 12.1 **How umask Is Applied**

ExamAlert

Each shell has its own **umask** value. If you change the **umask** in one shell, it will not affect the **umask** value in any other shell. To make a persistent change to your **umask** across logins, add the **umask** command to the **~/.bash_profile** file.

chown

The **chown** command is used to change the user owner or group owner of a file or directory. Table 12.6 demonstrates different ways to use this command.

TABLE 12.6 **chown Command Uses**

Example	Description
chown tim abc.txt	Changes the user owner of the **abc.txt** file to the **tim** user.
chown tim:staff abc.txt	Changes the user owner of the **abc.txt** file to the **tim** user and the group owner to the **staff** group.
chown :staff abc.txt	Changes the group owner of the **abc.txt** file to the **staff** group.

> **Note**
>
> Only the root user can change the user owner of a file. To change the group owner of a file, the user who executes the **chown** command must own the file and must be a member of the group that the ownership is being changed to.

Table 12.7 describes some important options for the **chown** command.

TABLE 12.7 **chown Command Options**

Option	Description
-R	Recursively applies changes to an entire directory structure.
--reference=*file*	Changes the user and group owner to the ownership of *file*.
-v	Enters verbose mode and produces output demonstrating the changes made.

setfacl/getfacl

An ACL (access control list) allows the owner of a file to give permissions for specific users and groups. The **setfacl** command is used to create an ACL on a file or directory:

```
sarah@OCS:~$ setfacl -m user:dane:r-- sales_report
```

The **-m** option is used to make a new ACL for the file. The format of the argument to the **-m** option is *what*:*who*:*permission*. The value for *what* can be one of the following:

- **user** or **u** when applying an ACL for a specific user
- **group** or **g** when applying an ACL for a specific group
- **others** or **o** when applying an ACL for others

- **mask** or **m** when setting the mask for the ACL (The mask is explained later in this section.)

The value for *who* is the user or group to which the permission will be applied. The permission can be provided as either a symbolic value (**r--**) or an octal value (**4**).

Once an ACL has been applied on a file or directory, a plus sign (**+**) character appears after the permissions when the **ls -l** command is executed, as shown here:

```
sarah@OCS:~$ ls -l sales_report
-rw-rw-r--+ 1 sarah sales 98970 Dec 27 16:45 sales_report
```

To view the ACL, use the **getfacl** command:

```
sarah@OCS:~$ getfacl sales_report
# file: sales_report
# owner: sarah
# group: sarah
user::rw-
user:william:r--
group::rw-
mask::rw-
other::r--
```

The following example demonstrates the process of setting an ACL for a group:

```
sarah@OCS:~$ setfacl -m g:games:6 sales_report
sarah@OCS:~$ getfacl sales_report
# file: sales_report
# owner: sarah
# group: sarah
user::rw-
user:william:r--
group::rw-
group:games:rw-
```

```
mask::rw-
other::r--
```

For regular permissions, the **umask** value is used to determine the default permissions applied for new files and directories. For ACLs, you can define a default ACL set for all new files and directories that are created within a shell by using the **-m** option with the **setfacl** command. In this case, the following syntax is used for the argument: **default:***what***:***who***:***permission*.

The following example creates a default ACL for the **reports** directory:

```
sarah@OCS:~$ mkdir reports
sarah@OCS:~$ setfacl -m default:g:games:r-x reports
sarah@OCS:~$ setfacl -m default:u:bin:rwx reports
sarah@OCS:~$ getfacl reports
# file: reports
# owner: sarah
# group: sarah
user::rwx
group::rwx
other::r-x
default:user::rwx
default:user:bin:rwx
default:group::rwx
default:group:games:r-x
default:mask::rwx
default:other::r-x
```

The following example demonstrates how new files and directories inherit the ACLs that were created by the commands executed in the previous example:

```
sarah@OCS:~$ mkdir reports/test
sarah@OCS:~$ getfacl reports/test
# file: reports/test
# owner: sarah
# group: sarah
user::rwx
user:bin:rwx
group::rwx
```

```
group:games:r-x
mask::rwx
other::r-x
default:user::rwx
default:user:bin:rwx
default:group::rwx
default:group:games:r-x
default:mask::rwx
default:other::r-x
sarah@OCS:~$ touch reports/sample1
sarah@OCS:~$ getfacl reports/sample1
# file: reports/sample1
# owner: sarah
# group: sarah
user::rw-
user:bin:rwx #effective:rw-
group::rwx #effective:rw-
group:games:r-x #effective:r--
mask::rw-
other::r--
```

Table 12.8 describes some important options for the **setfacl** command.

TABLE 12.8 **setfacl Command Options**

Option	Description
-b	Removes all ACLs.
-d	Sets a default ACL on a directory that will be inherited by any new file or directory created within this directory.
-R	Applies recursively.

ls

See the "File Permissions" section, earlier in this chapter, for details on how the **-l** option to the **ls** command is used to display file and directory permissions.

setenforce

You can disable the security policy (which is useful when testing a new policy or troubleshooting SELinux problems) with the **setenforce** command:

```
root@OCS:~# setenforce 0
root@OCS:~# getenforce
Permissive
```

While in permissive mode, SELinux does not block any access to files and directories, but warnings are issued and viewable in the system log files.

getenforce

Use the **getenforce** command to determine the current SELinux mode:

```
root@OCS:~# getenforce
Enforcing
```

The result **Enforcing** means SELinux is installed and the security policy is currently active. See the "States" section, earlier in this chapter, for more details regarding SELinux modes.

chattr/lsattr

While not technically permissions, file attributes affect how users access files and directories, so they logically belong in any discussion regarding permissions. With file attributes, the system administrator can modify key features of file access.

For example, a useful file attribute is one that makes a file immutable. An *immutable file* is completely unchangeable; it cannot be deleted or modified by anyone, including the root user. To make a file immutable, use the **chattr** command:

```
root@onecoursesource:~# chattr +i /etc/passwd
```

Now that this command has been executed, no user can change the **/etc/passwd** file, which means new users cannot be added to the system and existing users cannot be removed. This may seem like an odd thing to do, but imagine a situation in which a system is publicly available (like a kiosk in a mall). There is no need for new users, and you do not want anyone to remove users either.

To see the attributes for a file, use the **lsattr** command:

```
root@onecoursesource:~# lsattr /etc/passwd
----i--------e-- /etc/passwd
```

The **-** characters indicate file attributes that are not set. A complete list of attributes can be found in the man page for the **chattr** command. Table 12.9 describes some of the most useful file attributes.

TABLE 12.9 **File Attributes**

Attribute	Description
a	Enters append-only mode, in which only new data can be placed in the file.
A	Prevents modification of the access timestamp. This timestamp can be important for security reasons to determine when key system files have been accessed. However, for non-critical files, disabling the access time can make the system faster as it results in fewer hard drive writes.
e	Uses extent format, which allows for key features such as SELinux (discussed earlier in this chapter).
i	Makes a file immutable so that the file cannot be deleted or modified.
u	Makes a file undeletable so that the file cannot be deleted, but its contents can be modified.

To remove the immutable file attribute, use the following command:

```
root@onecoursesource:~# chattr -i /etc/passwd
root@onecoursesource:~# lsattr /etc/passwd
-------------e-- /etc/passwd
```

Table 12.10 describes some important options for the **chattr** command.

TABLE 12.10 **chattr Command Options**

Option	Description
-R	Recursively applies changes to an entire directory structure.
-V	Enters verbose mode and produces output demonstrating the changes made.

chgrp

The **chgrp** command is designed to change the group ownership of a file. The syntax of this command is as follows:

chgrp [*options*] *group_name file*

In the following example, the group ownership of the **abc.txt** file is changed to the **staff** group:

```
[student@OCS ~]$ chgrp staff abc.txt
```

> **Note**
>
> To change the group owner of a file, the user who executes the **chgrp** command must own the file and must be a member of the group that the ownership is being changed to.

Table 12.11 describes some important options for the **chgrp** command.

TABLE 12.11 **chgrp Command Options**

Option	Description
-R	Recursively applies changes to an entire directory structure.
--reference=*file*	Changes the user and group owner to the ownership of *file*.
-v	Enters verbose mode and produces output demonstrating the changes made.

setsebool

SELinux security policies include Booleans. A Boolean is a setting that can be assigned either a true or false value. This value can affect the behavior of the SELinux policy. To set an SELinux Boolean, use the **setsebool** command:

```
root@OCS:~# getsebool -a | grep abrt_anon_write
abrt_anon_write --> off
root@OCS:~# setsebool abrt_anon_write 1
root@OCS:~# getsebool -a | grep abrt_anon_write
abrt_anon_write --> on
```

See the next section for information about Boolean values.

getsebool

To view all Booleans, use the **getsebool** command:

```
root@OCS:~# getsebool -a | head
abrt_anon_write --> off
abrt_handle_event --> off
```

```
abrt_upload_watch_anon_write --> on
antivirus_can_scan_system --> off
antivirus_use_jit --> off
auditadm_exec_content --> on
authlogin_nsswitch_use_ldap --> off
authlogin_radius --> off
authlogin_yubikey --> off
awstats_purge_apache_log_files --> off
```

To determine what a Boolean is used for, use the **semanage** command:

```
root@OCS:~# semanage boolean -l | head
SELinux boolean State Default Description

privoxy_connect_any (on , on)
 Allow privoxy to connect any
smartmon_3ware       (off , off) Allow smartmon
 to 3ware
mpd_enable_homedirs  (off , off) Allow mpd to
 enable homedirs
xdm_sysadm_login      (off , off) Allow xdm to
 sysadm login
xen_use_nfs            (off , off) Allow xen to use nfs
mozilla_read_content   (off , off) Allow mozilla
 to read content
ssh_chroot_rw_homedirs (off , off) Allow ssh to
 chroot rw homedirs
mount_anyfile          (on , on) Allow mount to anyfile
```

See the previous section, "**setsebool**," for information on how to set a Boolean value.

chcon

Each process runs with a security context. To see this, use the **-Z** option to the **ps** command:

```
root@OCS:~# ps -fe | grep httpd | head -2
system_u:system_r:httpd_t:s0 root 1109 1 0 2018 ?
```

```
  00:51:56 /usr/sbin/httpd
system_u:system_r:httpd_t:s0 apache 1412 1109 0 Dec24 ?
  00:00:09 /usr/sbin/httpd
```

> **Note**
>
> The **head** command is used in this example simply to limit the output of the command.

The security context (**system_u:system_r:httpd_t:s0**) is complicated, but for understanding the basics of SELinux, the important part is **httpd_t**, which is like a security group or domain. As part of this security domain, the **/usr/sbin/httpd** process can only access files that are allowed by the security policy for **httpd_t**. This policy is typically written by someone who is an SELinux expert, and that expert needs to have proven experience regarding which processes should be able to access specific files and directories on the system.

Files and directories also have an SELinux security context that is defined by the policy. To see a security context for a specific file, use the **-Z** option to the **ls** command (and note that the SELinux context contains so much data that the filename cannot fit on a single line):

```
root@OCS:~# ls -Z /var/www/html/index.html
unconfined_u:object_r:httpd_sys_content_t:s0/var/www/html/index.html
```

Use the **chcon** command to change the context of a file or directory:

```
root@OCS:~# chcon -t user_home_t /var/www/html/index.html
```

restorecon

SELinux rules define the default security contexts for a majority of the system files. The **restorecon** command is used to reset the default security context on a file or directory.

Example:

```
root@OCS:~# restorecon /var/www/html/index.html
```

A commonly used option to the **restorecon** command is the **-R** option, which recursively performs the changes on a directory structure.

semanage

The **semanage** command works just like the **chcon** command with one exception: The changes are persistent. In other works, the **restorecon** command converts the SELinux contexts back to a previous value if you set the contexts with the **chcon** command. However, if you use the **semanage** command, then using **restorecon** (or using the **/.autorelabel** method) will not result in the SELinux contexts reverting to a previous value.

audit2allow

When a process is denied access by SELinux, an entry like the following is made in the **/var/log/audit/audit.log** file:

```
type=AVC msg=audit(1435467578.546:231): avc:  denied  { write } for
pid=12345 comm="test" name="cache" dev=dm-0 ino=12345
```

If you discover that this access should have been permitted, you can use the **audit2allow** command to generate a rule that permits access:

```
$ grep test /var/log/audit/audit.log | audit2allow -R -M test
```

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which user can view the contents of the file specified in the following command?

```
[student@OCS ~]$ ls -l /etc/chrony.keys
-rw-r-----. 1 root chrony 62 May 9 2018 /etc/chrony.keys
```

   - ❍ **A.** Only the root user
   - ❍ **B.** The root user and members of the **chrony** group
   - ❍ **C.** All users
   - ❍ **D.** No users

2. Which permission changes the meaning of the write permission on directories?
   - ❍ **A. suid**
   - ❍ **B. sgid**
   - ❍ **C.** sticky bit
   - ❍ **D.** None of these answers are correct.

3. Which file automatically relabels the SELinux contexts for the entire filesystem at the next boot?
   - ❍ **A. /.autolabel**
   - ❍ **B. /autorelabel**
   - ❍ **C. /.auto-relabel**
   - ❍ **D. /.autorelabel**

4. Which filesystem attribute prevents modification of the access timestamp?
   - ❍ **A. a**
   - ❍ **B. A**
   - ❍ **C. e**
   - ❍ **D. u**

Cram Quiz Answers

1. **B.** Read permissions are required to view the contents of a file. The root user and the **chrony** group have read permissions, but the **others** permission set lacks this permission.
2. **C.** When the sticky bit is set on directories, files in the directory can only be removed by the user owner of the file, the owner of the directory, or the root user. Removal of files is related to the write permission on directories.
3. **D.** If you want to change the SELinux contexts for all of the files, you can tell the system to perform an autorelabel process on every file in the filesystem during the next boot process. To do this, create an empty file in the root directory like this: **touch /.autorelabel**. Note that the other answers are files that have no impact on SELinux.
4. **B. A** prevents modification of the access timestamp. **a** means “append-only mode.” **e** allows features like SELinux. **u** makes the file undeletable.

This page intentionally left blank

CHAPTER 13

Create Simple Shell Scripts to Automate Common Tasks

This chapter covers the following Linux+ XK0-005 exam objective:

- **3.1:** Given a scenario, create simple shell scripts to automate common tasks.

Get ready for a larger-than-average exam topic: scripting. *Scripting* is the process of creating programs from BASH commands. Although the programming features in BASH are rudimentary compared to those in other languages, the vast number of commands available makes BASH a very powerful language for Linux.

While reading this chapter, keep in mind that much of what you will learn here is not just for scripting. For example, you will learn about redirection, which includes the process of sending output of commands into files rather than the screen. This is very useful not just for scripting but for everyday situations.

In addition to learning about scripting features, like loops and conditional statements, you will also learn about some utilities that are often used in scripts. You will also learn how to make use of variables to store information as well as display useful shell information.

This chapter provides information on the following topics: shell script elements, command script utilities, environment variables, and relative and absolute paths.

Shell Script Elements

The first line of a script should include the path to the interpreter. For BASH shell scripts, this should be the path to the executable **bash** command. This path can be discovered by executing the following command:

```
[root@OCS ~]$ which bash
/bin/bash
```

You can add this path to the first line of the script as follows:

```
#!/bin/bash
```

The combination of **#** (the hash character) and **!** (the bang character) forms the *shebang* sequence. (Say "hash bang" quickly to discover where the term *shebang* comes from; often it is said with a silent *e*.)

Scripts should be owned by regular users, not by the root user. Running scripts owned by root can result in security issues. If a script is owned by root, it should not have the **suid** permission set on it because that would result in a serious security risk. (Most modern Linux distributions don't permit root-owned **suid** scripts.)

After creating a script, you need to add execute permission to run it like a program. Here's the syntax to do this:

```
chmod a+x script_name
```

A BASH shell script should end with a **.sh** extension to indicate that the contents of the file contains a BASH shell script. (The **.bash** extension is also used but not as widely.)

Other scripting languages should have extensions to indicate the language type:

- **.pl:** Perl scripts
- **.py:** Python scripts
- **.tcsh:** TC shell scripts
- **.csh:** C shell scripts

A comment starts with a hash (**#**) character and extends to the end of the line, as in this example:

```
#Set the count variable:
count = 100
```

The following is an example of a comment on a line that also contains code:

```
count = 100 #Set the count variable
```

The rest of this section describes additional shell script elements.

Loops

A looping construct is code within a BASH script that performs a set of operations—potentially multiple times. This section examines three commonly used loop constructs: **while**, **until**, and **for** loops.

while

A **while** loop is designed to perform a set of tasks (commands) as long as a conditional statement returns the value **true**. For example, the following code can be used to check user input to verify that the correct value is provided:

```
read -p "Enter a zip code: "
while echo $zip | egrep -v "^[0-9]{5}$"
do
   echo "$zip is not a valid ZIP code, try again"
   read -p "Enter a zip code" zip
done
```

Here is an example of this code being run:

```
Enter a zip code: 8888
8888 is not a valid ZIP code, try again
Enter a zip code: huh?
huh? is not a valid ZIP code, try again
Enter a zip code: 92020
```

> **Note**
>
> The **read** command reads input from stdin, which is typically the user's keyboard input. See the "read" section, later in this chapter, for details.

for

A **for** loop is used to iterate over a list of values. For example, the following executes the **wc -l** command on a list of files:

```
for file in "/etc/r*.conf"
do
```

```
    wc -l $file
done
```

Sample output:

```
 43 /etc/request-key.conf
 3 /etc/resolv.conf
 61 /etc/rsyslog.conf
```

ExamAlert

Be aware of when you should use a **for** loop instead of a **while** loop as this might come up on the Linux+ XK0-005 exam.

until

The **until** statement works like the **while** statement, but it executes its code set if the conditional statement is false. Compare the following pseudocode for the **while** statement to the pseudocode for the **until** statement:

```
while conditional_check
do
    #executes if conditional_check returns true
done

until conditional_check
do
    #executes if conditional_check returns false
done
```

ExamAlert

Be sure you know the difference between a **while** loop and an **until** loop.

Conditionals

The **test** statement is used to perform conditional tests that are used to compare values and perform file-testing operations.

A common method of running the **test** statement is to place the operation within square brackets. For example, these two statements perform the same action:

```
test $name1 = $name2
[ $name1 = $name2 ]
```

Table 13.1 describes the common **test** operations.

TABLE 13.1 **Common test Operations**

Operation	Description
str1 = *str2*	Returns **true** if two strings are equal to each other.
str1 != *str2*	Returns **true** if two strings are not equal to each other.
-z *str*	Returns **true** if the size of *str* is zero; for example, **[-z $name]**.
-n *str*	Returns **true** if the size of *str* is not zero; for example, **[-n $name]**.
int1 **-eq** *int2*	Returns **true** if two integers are equal to each other.
int1 **-ne** *int2*	Returns **true** if two integers are not equal to each other.
int1 **-gt** *int2*	Returns **true** if the first integer is greater than the second integer.
int1 **-ge** *int2*	Returns **true** if the first integer is greater than or equal to the second integer.
int1 **-lt** *int2*	Returns **true** if the first integer is less than the second integer.
int1 **-le** *int2*	Returns **true** if the first integer is less than or equal to the second integer.
-d *file*	Returns **true** if the file is a directory.
-f *file*	Returns **true** if the file is a plain file.
-e *file*	Returns **true** if the file exists (regardless of what type of file or directory it is).
-r *file*	Returns **true** if the file is readable.
-w *file*	Returns **true** if the file is writable.
-x *file*	Returns **true** if the file is executable.

if

An **if** statement is used to execute one or more commands, based on the outcome of a conditional statement. For example, the following script displays **"hi"** if the **$name** variable is set to **"Bob"**:

```
if [ $name = "Bob" ]
then
    echo "hi"
fi
```

Note that to end the commands executed within an **if** statement, you use the **fi** statement (which is *if* spelled backward).

An **else** statement is used with an **if** statement to provide an alternative in the event that the condition returns **false**. Here's an example:

```
if [ $name = "Bob" ]
then
    echo "hi"
else
    echo "I don't know you"
fi
```

Multiple conditional checks can be performed, like so:

```
if [ $name = "Bob" ]
then
    echo "hi, Bob"
elif [ $name = "Sue" ]
then
    echo "hi, Sue"
else
    echo "I don't know you"
fi
```

ExamAlert

You should know how the different statements end. **fi** ends an **if** statement, and **done** ends **for**, **while**, and **until** statements.

switch/case

The **case** statement (sometimes called a "switch" statement) compares a variable's value to globbing patterns.

Syntax:

```
case $var in
glob 1)
   #statements
   #statements;;
glob 2)
   #statements
   #statements;;
esac
```

Two semicolons (**;;**) are used to end a set of statements and must appear before the next glob. To end a **case** statement, use **esac** (*case* spelled backward).

Example:

```
case $1 in
 start)
 echo "Starting the process"
 ;;
 stop)
 echo "Stopping the process"
 ;;
 *)
 echo "I need to hear start or stop"
esac
```

Shell Parameter Expansion

A script can be run with arguments, like so:

```
./example.sh arg1 arg2 arg3
```

Each of the arguments is stored in $1, $2, $3, and so forth. $0 is the name of the script itself.

The special BASH variable **$#** contains the number of arguments passed to the script.

Globbing

A file glob (also called a *wildcard*) is any character provided on the command line that represents a portion of a filename. Table 13.2 describes the globs that are supported.

TABLE 13.2 **Supported Globs**

Glob	Description
*****	Matches zero or more characters in a filename.
?	Matches any single character in a filename.
[ ]	Matches a single character in a filename as long as that character is represented within the **[ ]** characters.

This example copies all files that end in **.txt**:

```
[student@OCS ~]$ cp *.txt ~
```

This example removes all files that are four characters long:

```
[student@OCS ~]$ rm ????
```

To view the file types for all files in the current directory that begin with **a**, **b**, or **c**, execute the following command:

```
[student@OCS ~]$ file [abc]*
```

This command could also be executed as follows:

```
[student@OCS ~]$ file [a-c]*
```

where **a-c** represents a range of characters. This must be a valid range within the ASCII text table. To view this table, execute the following command:

```
[student@OCS ~]$ man ascii
```

Multiple ranges or lists of values are permitted, as shown in this example:

```
[student@OCS ~]$ file [a-cg-jz]*
```

By placing a **!** character at the beginning of a range, you can specify what characters cannot match. For example, the following command copies all files in the current directory that don't begin with an **a**, a **b**, or a **c**:

```
[student@OCS ~]$ cp [!abc]* ~
```

This example also demonstrates that complex patterns can be created using multiple glob characters.

Brace Expansions

With the variable expansion BASH feature, you can return a different value. For example, the following commands return **"Bob"** if the **$name** variable isn't set:

```
$ echo "Hello, ${name:-Bob}"
Hello, Bob
$ echo "Hello, $name"
Hello,
```

Note that when you use the commands in this way, you don't set the variable but rather use the provided value for this one situation. To return a value and also set the variable, use the following commands:

```
$ echo "Hello, ${name:=Bob}"
Hello, Bob
$ echo "Hello, $name"
Hello, Bob
```

There are other variable expansion features. For example, the following commands will only return the first 24 characters of the variable, resulting in a more "uplifting" statement:

```
$ times="It was the best of times, it was the worst of times."
$ echo ${times: 0:24}
It was the best of times
```

Note that in this example, **0** represents the first character to return, and **24** represents the total number of characters to return (starting with character **0**).

Command substitution is the process of executing a subcommand within a larger command. It is typically used to gather data and store it in a variable. For example, the following command stores the output of the **date** command in the **$today** variable:

```
today=$(date)
```

Command substitution can be performed by using either of the following:

- **$(cmd)**
- **'cmd'**

Note that the second method uses backquote characters (also called *backtick* characters), not single-quote characters.

Comparisons

Several conditional statements are available for the BASH shell, including the **if** statement, the **while** loop, and the **until** loop. These conditional statements make use of a comparison feature in the BASH shell, which is highlighted in the following example:

```
if [ $name = "Bob" ]
then
    echo "hi"
fi
```

This example performs an implicit call of the BASH command **test** that can be used to perform several comparison tests, including integer (arithmetic) comparisons, string comparisons, and file testing operations. For example, the following example tests whether the string value that is stored in the **$name1** variable does not equal the string stored in the **$name2** variable:

```
[ "$name1" != "$name2" ]
```

> **Note**
>
> The spacing around the square brackets is very important. There should be a space after each opening square bracket and before each closing square bracket. Without these spaces, an error message will occur.

The next few sections cover different types of comparison operations.

Arithmetic

If you want to perform integer (arithmetic) comparison operations, use the following:

- **-eq:** Evaluates to true if values are equal to each other.

- **-ne:** Evaluates to true if values are not equal to each other.
- **-gt:** Evaluates to true if the first value is greater than the second value.
- **-lt:** Evaluates to true if the first value is less than the second value.
- **-ge:** Evaluates to true if the first value is greater than or equal to the second value.
- **-le:** Evaluates to true if the first value is less than or equal to the second value.

Example:

```
if [ $age -lt 25 ]
then
    echo "the person is less than 25 years old"
fi
```

String

The two most common string comparisons are equal to and not equal to. Here is an example of equal to:

```
if [ $name = "Bob" ]
then
    echo "hi"
fi
```

Here is an example of not equal to:

```
if [ $name != "Bob" ]
then
    echo "goodbye"
fi
```

In addition to determining whether two strings are equal or not equal, you may also find the **-n** option useful. This option determines whether a string is not empty, which is useful when testing user input. For example, the previous code will read data from user input (the keyboard), assign the input to the **$name** variable, and test to make sure the user typed something for the name.

ExamAlert

Be prepared for Linux+ XK0-005 exam questions that test your ability to use the right type of comparison (arithmetic or string).

Boolean

Comparisons return Boolean true or false values. You can reverse these values. For example, if a comparison returns **false**, you can reverse it to be **true**. One common comparison set that this reversal feature is useful for is file test operators. These operations include the following:

- **-d:** True if a file is a directory.
- **-f:** True if a file is a regular file.
- **-r:** True if a file exists and is readable by the user running the script.
- **-w:** True if a file exists and is writable by the user running the script.
- **-x:** True if a file exists and is executable by the user running the script.
- **-L:** True if the first value is less than or equal to the second value.

For example, you can check whether a file is actually readable by using the following:

```
if [ -r /etc/shadow ]
then
    echo "You can read the shadow file"
fi
```

What if you don't want to check whether a file is readable but instead want to check whether it is not readable? You can reverse the conditional value by placing a **!** character before it:

```
 if [ ! -r /etc/shadow ]
then
    echo "You can't read the shadow file"
fi
```

Variables

See the "Environment Variables" section, later in this chapter.

Search and Replace

See the "**grep**" and "**sed**" sections, later in this chapter.

Regular Expressions

The term *regex* stands for *regular expression* (RE), which is a character or set of characters designed to match other characters. For example, in utilities that support REs, a **.** (dot, or period) character will match a single character of any type, whereas **[a-z]** will match any single lowercase letter.

There are two types of REs: basic and extended. Basic REs are the original, and extended REs are the newer additions. Utilities that use REs normally support basic REs by default and have some switch or feature to enable extended REs. Although documentation may refer to basic REs as obsolete, they are still used by most modern utilities.

Table 13.3 describes the commonly used basic REs.

TABLE 13.3 **Commonly Used Regular Expressions**

RE	Description
^	Matches the beginning of a line.
$	Matches the end of a line.
*	Matches the preceding character zero or more times.
.	Matches exactly one character.
[ ]	Matches exactly one character that is within the **[ ]** characters; a list of characters (for example, **[abc]**) or a range of characters (for example, **[a-c]**) is permitted.
[^]	Matches exactly one character that is *not* within the **[ ]** characters; a list of characters (for example, **[^abc]**) or a range of characters (for example, **[^a-c]**) is permitted.
****	Escapes the special meaning of a regular expression; for example, the pattern **\.*** would match the value **".*"**.

Table 13.4 describes the commonly used extended REs.

TABLE 13.4 **Commonly Used Extended Regular Expressions**

RE	Description
( )	Groups sets of characters together to form an expression—for example, **(abc)**.
X|*Y*	Matches either *X* or *Y*.
+	Matches the preceding character or expression one or more times.
{*X* **}**	Matches the preceding character or expression *X* times.
{*X***,}**	Matches the preceding character or expression *X* or more times.
{*X***,***Y* **}**	Matches the preceding character or expression *X* to *Y* times.
?	Indicates that the previous character or expression is optional.

ExamAlert

Regular expressions are a big topic. Focus on the information provided in this chapter for the Linux+ XK0-005 exam.

Standard Stream Redirection

Each command is able to send two streams of output (standard output and standard error) and can accept one stream of data (standard input). In documentation, these terms can also be described as follows:

- Standard output = stdout or STDOUT
- Standard error = stderr or STDERR
- Standard input = stdin or STDIN

By default, stdout and stderr are sent to the terminal window, whereas stdin comes from keyboard input. In some cases, you want to change these locations, and this can be accomplished through a process called *redirection*.

Table 13.5 describes the methods used to perform redirection.

TABLE 13.5 **Methods Used for Redirection**

Method	Description
cmd < file	Overrides STDIN so the input comes from the file specified.
cmd > file	Overrides STDOUT so the output goes into the file specified, overwriting the existing contents of the file.
cmd >> file	Overrides STDOUT so the output goes into the file specified, appending to the end of the existing contents of the file.

Method	Description
cmd 2> file	Overrides stderr so the output goes into the file specified.
cmd &> file	Overrides both STDOUT and stderr so the output goes into the file specified.
cmd1 **|** *cmd2*	Overrides STDOUT from *cmd1* so it goes into *cmd2* as STDIN.

In the following example, STDOUT of the **cal** program is sent to a file named **month**:

```
[student@OCS ~]$ cal > month
```

It is common to redirect both STDOUT and stderr into separate files, as demonstrated in this example:

```
[student@OCS ~]$ find /etc -name "*.cfg" -exec file {} \;
> output 2> error
```

Redirecting STDIN is fairly rare because most commands can accept a filename as a regular argument; however, the **tr** command, which performs character translations, requires STDIN to be redirected:

```
[student@OCS ~]$ cat /etc/hostname
localhost
[student@OCS ~]$ tr 'a-z' 'A-Z' < /etc/hostname
LOCALHOST
```

ExamAlert

Be ready for Linux+ XK0-005 questions that focus on the difference between appending (**>>**) to a file and overwriting (**>**) a file's contents.

|

The process of piping (called piping because the | character is referred to as a "pipe") the output of one command to another command results in a more powerful command line. For example, the following example takes the standard output of the **ls** command and sends it into the **grep** command to filter files that were changed on April 16:

```
[student@OCS ~]$ ls -l /etc | grep "Apr 16"
-rw-r--r-- 1 root root 321 Apr 16 2014 blkid.conf
drwxr-xr-x 2 root root 4096 Apr 16 2014 fstab.d
```

In this example, lines 41–50 of the copyright file are displayed (though note that some lines were truncated to fit within the margins of this book):

```
[student@OCS ~]$ head -50 copyright | tail
 b) If you have received a modified Vim that was...
    mentioned under a) you are allowed to further...
    unmodified, as mentioned at I). If you make...
    the text under a) applies to those changes.
 c) Provide all the changes, including source code, with every
    copy of the modified Vim you distribute. This...
    the form of a context diff. You can choose what...
    for new code you add. The changes and their license must not
    restrict others from making their own changes...
    version of Vim.
 d) When you have a modified Vim which includes changes as
    mentioned
```

You can add additional commands as demonstrated here:

```
[student@OCS ~]$ head -50 copyright | tail | nl
 1 b) If you have received a modified Vim that was...
 2     mentioned under a) you are allowed to further...
 3    unmodified, as mentioned at I). If you make...
 4    the text under a) applies to those changes.
 5 c) Provide all the changes, including source code...
 6    copy of the modified Vim you distribute. This...
 7    the form of a context diff. You can choose...
 8    for new code you add. The changes and their...
 9    restrict others from making their own changes...
 10    version of Vim.
 11 d) When you have a modified Vim which includes changes as
 12    mentioned
```

Note that the order of execution makes a difference. In the previous example, the first 40 lines of the copyright file are sent to the **tail** command. Then the last 10 lines of the first 40 lines are sent to the **nl** command for numbering. Notice the difference in output when the **nl** command is executed first:

```
[student@OCS ~]$ nl copyright | head -50 | tail
```

```
36 b) If you have received a modified Vim that was...
37    mentioned under a) you are allowed to further...
38    unmodified, as mentioned at I). If you make...
39    the text under a) applies to those changes.
40 c) Provide all the changes, including source code...
41    copy of the modified Vim you distribute. This...
42    the form of a context diff. You can choose...
43    for new code you add. The changes and their...
44    restrict others from making their own changes...
45    version of Vim.
45 d) When you have a modified Vim which includes changes as
47    mentioned
```

||

The || characters can be used to create a simple **if**-like statement that checks whether a command returns an exit status of **false**. For example, the following would display **bob is not there** if the **bob** account does not exist in the **/etc/passwd** file:

```
grep "^bob:" /etc/passwd || echo "bob is not there"
```

In this example, the **grep** command produces a return status of **true** if it can find **bob:** at the beginning of any line in the **/etc/passwd** file. If **grep** returns **true**, the statement after the || characters is not executed. If **grep** can't find **bob:** at the beginning of any line in the **/etc/passwd** file, it returns **false**, and the shell executes the **echo** command that appears after the || characters.

ExamAlert

CompTIA places this topic under the "Standard stream redirection" category in the Linux+ XK0-005 exam objectives even though this topic is not redirection feature. Don't let its placement in the exam objectives throw you off regarding the purpose of this topic.

>

See the "Standard Stream Redirection" section, earlier in this chapter.

>>

See the "Standard Stream Redirection" section, earlier in this chapter.

<

See the "Standard Stream Redirection" section, earlier in this chapter.

<<

See the "Here Documents" section, later in this chapter.

&

When you execute a program or command via the command line, the shell is referred to as the *parent process*, and the command that is executed is called the *child process*. Normally when the child process is executed in the shell, it runs in the foreground, which means that as long at the child process is running, you can't use the parent process. You can change this behavior by placing a command in the background. To do this, run the command with the **&** character at the end of the command:

```
$ find /etc -name sample.cfg &
```

This is typically a good solution for commands that take a long time to run. You might not want to wait until the command finishes before typing other commands; however, if the child process produces output, this can be confusing because this output just displays on the screen as it is generated by the child process. Therefore, consider redirecting all output to files, as in this example, to avoid this confusion:

```
$ find /etc -name sample.cfg >/tmp/output 2>/tmp/error &
```

ExamAlert

CompTIA places this topic under the "Standard stream redirection" category in the Linux+ XK0-005 exam objectives even though this topic is not redirection feature. Don't let its placement in the exam objectives throw you off regarding the purpose of this topic.

&&

The **&&** characters can be used to create a simple **if**-like statement that checks whether a command returns the exit status **true**. For example, the following displays **bob is there** if the **bob** account exists in the **/etc/passwd** file:

```
$ grep "^bob:" /etc/passwd && echo "bob is there"
```

In this example, the **grep** command produces the return status **true** if it can find **bob:** at the beginning of any line in the **/etc/passwd** file. If **grep** returns **true**, then the statement after the **&&** characters is executed. If **grep** can't find **bob:** at the beginning of any line in the **/etc/passwd** file, it returns **false**, and the shell does not execute the **echo** command that appears after the **&&** characters.

ExamAlert

CompTIA places this topic under the "Standard stream redirection" category in the Linux+ XK0-005 exam objectives even though this topic is not redirection feature. Don't let its placement in the exam objectives throw you off regarding the purpose of this topic.

Redirecting: stderr/stdout

See the "Standard Stream Redirection" section, earlier in this chapter.

Here Documents

A *here document* provides a redirection technique that allows you to send a large chunk of data to a command. For example, suppose you want to send the following text to stdout:

> We choose to go to the moon. We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win, and the others, too.

You can use **<<** with this text as follows to create a here document:

```
cat <<EOF
```

```
We choose to go to the moon. We choose to go to the moon in this
decade and do the other things, not because they are easy, but because
they are hard, because that goal will serve to organize and measure
the best of our energies and skills, because that challenge is one
that we are willing to accept, one we are unwilling to postpone, and
one which we intend to win, and the others, too.
EOF
```

Everything between the two EOF tags is sent to the **cat** command as if it came from stdin. This provides a way to display text using WYSIWYG (what you see is what you get). Normally here documents are used in shell scripts, but they can be used on the command line as well.

Exit Codes

See the "**$?**" section, later in this chapter.

Shell Built-in Commands

Most commands that you execute in BASH are actually external commands. This means they aren't part of BASH itself but rather part of another program. You can tell if a command is a shell built-in command by using the **type** command. For example, the **date** command is not a built-in command, while the **read** command is:

```
[bo@mail ~]$ type date
date is /usr/bin/date
[bo@mail ~]$ type read
read is a shell builtin
```

This section explores some of the most useful shell built-in commands.

read

The **read** statement extracts information from STDIN and places the extracted data into one or more variables. For example, the following reads data from STDIN and places the data into the **$name** variable:

```
read name
```

To assign values to different variables, use the following syntax:

```
read var1 var2 var3
```

Use the **-p** option to issue a prompt to the user:

```
$ read -p "Enter your name" name
```

echo

The **echo** command is used to display information. Typically, it is used to display the value of variables.

Example:

```
[student@OCS ~]$ echo $HISTSIZE
1000
```

The **echo** command has only a few options. The most useful one is the **-n** option, which prevents a newline character from printing at the end of the output.

Some special character sequences can be incorporated into an argument to the **echo** command. For example, the command **echo "hello\nthere"** sends the following output:

```
hello
there
```

Table 13.6 describes some useful character sequences for the **echo** command.

TABLE 13.6 **Useful Character Sequences for the echo Command**

Sequence	Description
\a	Terminal bell ringer
\n	Newline character
\t	Tab character
****	A single backslash character

source

Most commands are executed as separate processes that have their own environments. The **source** command allows you to execute a BASH script as if the commands within the script were executed directly on the command line.

Example:

```
[root@OCS ~]$ source ./functions.sh
```

Use the **.** command to perform the same function as the **source** command, as shown here:

```
[root@OCS ~]$ . ./functions.sh
```

This technique is often used within shell scripts to source code from other shell scripts. The idea of "sourcing" code from another script is to include code from another script within the script you are writing. For example, a script might need some variables that are declared in another script.

Common Script Utilities

Because a script is, at its core, a collection of BASH commands, any BASH command could be used within a script. However, there are some commands that are especially useful for scripting. This section covers some of the most popular script utilities (which are also popular in everyday use).

awk

The **awk** command is used to modify text that is in a simple database format. For example, consider the following data:

```
[student@OCS ~]$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

With the **awk** command, you can break the data shown in the previous example into different fields and then perform actions on that data:

```
[student@OCS ~]$ head /etc/passwd | awk -F: '{print $1,$7}'
root /bin/bash
bin /sbin/nologin
```

```
daemon /sbin/nologin
adm /sbin/nologin
lp /sbin/nologin
sync /bin/sync
shutdown /sbin/shutdown
halt /sbin/halt
mail /sbin/nologin
operator /sbin/nologin
```

In this example, the **-F** option is used to specify the field separator. Each field is assigned to a variable. Table 13.7 describes these variables.

TABLE 13.7 **Variables Assigned to Fields Generated by the awk Command**

Variable	Description
$1, **$2**, and so on	The field variables.
$0	The entire line.
NF	The number of fields on the line. (Do not use the **$** character for this variable.)
NR	The current line number.

Table 13.8 describes some important options of the **awk** command.

TABLE 13.8 **awk Command Options**

Option	Description
-F	Used to specify the field separator.
-f	Used to specify a file that contains the **awk** commands to execute.

ExamAlert

awk is a huge topic. For the Linux+ XK0-005 exam, focus on the components of **awk** that are covered in this chapter.

Sed

Use the **sed** utility to make automated modifications to files. The basic format for the **sed** command is **sed** *'s/RE/string/' file*, where the *RE* refers to the term *regular expression*, a feature that uses special characters to match patterns. See the "Regular Expressions" section, earlier in this chapter, for details regarding regular expressions.

Here is an example of the **sed** command:

```
[student@OCS ~]$ head -n 5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
[student@OCS ~]$ head -n 5 /etc/passwd | sed 's/bin/----/'
root:x:0:0:root:/root:/----/bash
----:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/s----:/sbin/nologin
adm:x:3:4:adm:/var/adm:/s----/nologin
lp:x:4:7:lp:/var/spool/lpd:/s----/nologin
```

sed is a very powerful utility with a large number of features. Table 13.9 describes some of the most useful **sed** operations.

TABLE 13.9 **sed Operations**

Feature	Description
'/*RE***/d'**	Deletes lines that match the RE from the output of the **sed** command.
'/*RE***/c********string***'**	Changes lines that match the RE to the value of *string*.
'/*RE***/a********string***'**	Adds *string* on a line after all lines that match the RE.
'/*RE***/i********string***'**	Adds *string* on a line before all lines that match the RE.

The **sed** command has two important modifiers (that is, characters added to the end of the **sed** operation):

- **g:** Means "global." By default, only the first RE pattern match is replaced. When the **g** modifier is used, all replacements are made. Figure 13.1 shows an example.

```
[student@localhost ~]$ head -n 5 /etc/passwd | sed 's/bin/----/'
root:x:0:0:root:/root:/----/bash
----:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/s----:/sbin/nologin
adm:x:3:4:adm:/var/adm:/s----/nologin
lp:x:4:7:lp:/var/spool/lpd:/s----/nologin
[student@localhost ~]$ head -n 5 /etc/passwd | sed 's/bin/----/g'
root:x:0:0:root:/root:/----/bash
----:x:1:1:----:/----:/s----/nologin
daemon:x:2:2:daemon:/s----:/s----/nologin
adm:x:3:4:adm:/var/adm:/s----/nologin
lp:x:4:7:lp:/var/spool/lpd:/s----/nologin
```

FIGURE 13.1 **The g Modifier**

- **i:** Means "case-insensitive." This modifier matches an alpha character regardless of its case. So, the command **sed 's/a/-/i'** would match either **a** or **A** and replace it with the **-** character.

The **sed** command can also change the original file (instead of displaying the modified data to the screen). To change the original file, use the **-i** option.

ExamAlert

sed is a huge topic. For the Linux+ XK0-005 exam, focus on the components of **sed** that are covered in this chapter.

find

The **find** command uses different criteria to search the live filesystem for files and directories. Here's the syntax of the command:

```
find [options] starting_point criteria action
```

where *starting_point* is the directory to start the search from, *criteria* is what to search for, and *action* is what to do with the results.

Table 13.10 describes the options that are designed to modify how the **find** command behaves.

TABLE 13.10 **find Command Options**

Option	Description
-maxdepth *n*	Limits how deeply into subdirectories the search goes; for example, **find-maxdepth 3** limits the search to three subdirectories deep.
-mount	Prevents searches of directories that serve as mount points. This is useful when you're searching from the **/** directory.
-regextype *type*	When REs are used, this option specifies what type of RE will be used; *type* can be **emacs** (default), **posix-awk**, **posix-basic**, **posix-egrep**, or **posix-extended**.

Most criteria-based options allow you to specify a numeric value as an argument. This can be preceded by a **-** or **+** character to indicate "less than" or "greater than." For example, **+5** would mean "more than five." Table 13.11 lists some important criteria-based options.

TABLE 13.11 **General Search Criteria**

Option	Description
-amin *n*	Matches files based on access time; for example, **-amin -3** matches files accessed within the past three minutes.
-group *name*	Matches files that are owned by the *name* group.
-name *pattern*	Matches a file or directory based on the *pattern* provided; the *pattern* can be a regular expression.
-mmin *n*	Matches files based on modification time; for example, **-mmin -3** matches files modified within the past three minutes.
-nogroup	Matches files not owned by a group.
-nouser	Matches files not owned by a user.
-perm *mode*	Matches files that match the permission specified by the *mode* (octal or symbolic); see the examples in Table 13.12.
-size *n*	Matches files based on file size; the value *n* can be preceded by a **+** (more than) or **-** (less than) and by a unit modifier: **c** for bytes, **k** for kilobytes, **M** for megabytes, or **G** for gigabytes.
-type *fstype*	Searches for a file based on file type, where **d** is for directory, **f** is for plain file, **l** is for symbolic link, and so on.
-user *username*	Matches all files owned by the *username* user (for example, **find / home -user bob**).

Table 13.12 provides some examples using the **-perm** option.

TABLE 13.12 **Permission Search Criteria**

Option	Description
-perm 775	Matches files that exactly match the octal permissions 775 (**rwxrwxr-x.**)
-perm u=rw	Matches files that exactly match the symbolic permissions read and write for the owner (**rw-------.**)
-perm -444	Matches files that have the octal permissions 444 but disregards any other permission values. (Possible matches include **rwxrwxr-x**, **r--r--r--**, and **rwxr--r--**.)
-perm ugo=r	Matches files that have the symbolic permission read for all three permission sets but disregards any other permission values. (Possible matches include **rwxrwxr-x**, **r--r--r--**, and **rwxr--r--**.)
-perm /444	Matches files that have the octal permissions 444 and disregards any other permission values. (Possible matches include **r--------**, **rw-rw-r--**, and **rwxr--r--**.)

Once a file is found, an action can be taken on the file. Table 13.13 describes some important action-based options.

TABLE 13.13 **Results Options**

Option	Description
-delete	Deletes all matched files (for example, **find /tmp -name "*.tmp" -delete**).
-exec *command*	Executes a command on each matched file.
-ls	Lists details about each matched file.
-ok	Executes a command on each matched file but prompts the user before each match; the prompt is a yes/no question to determine if the user wants to execute the command.
-print	Prints the filename of each matched file; this is the default action option.

Here's an example of the **-exec** option:

```
[root@OCS ~]# find /etc -name "*.cfg" -exec file {} \;
/etc/grub2.cfg: symbolic link to '../boot/grub2/grub.cfg'
/etc/enscript.cfg: ASCII text
/etc/python/cert-verification.cfg: ASCII text
```

\; is used to build a command line. For example, the command that was executed for the previous **find** example is:

```
file /etc/grub2.cfg; file /etc/enscript.cfg; file /etc/python/cert-
verification.cfg
```

The **** before the **;** is required to escape the meaning of the **;** character for the BASH shell, so the **;** character is passed to the **find** command as a regular argument.

The **{}** characters represent where in the command the matching filename is placed. This can be used more than once, as demonstrated in the next example, which makes a copy of each matched file:

```
find /etc -name "*.cfg" -exec cp {} /tmp/{}.bak \;
```

xargs

The **xargs** command takes data from STDIN to build and execute commands. Here is the syntax of the command:

```
input_command | xarg execute_command
```

where *input_command* is designed to provide information for **xargs** to provide as arguments to *execute_command*.

For example, suppose you want to run the **wc -l** command on every file in the **/etc** directory that begins with the letter **e**. You could use this command:

```
[student@OCS ~]$ ls -d /etc/e* | xargs wc -l
 1 /etc/ec2_version
 1 /etc/environment
 2 total
```

Table 13.14 describes some important options of the **xargs** command.

TABLE 13.14 **xargs Options**

Option	Description
-0	Handles whitespace issues. Normally used to handle filenames that have whitespace characters; every character is treated as a literal character.
-d	Used to change the delimiter between arguments (where the default is a space character).

Option	Description
-n *max-args*	Indicates the maximum number of arguments per *execute_command*.
-p	Prompts the user before executing the *execute_command*.
-t	Displays the *execute_command* command before executing the command.

grep

Use the **grep** command to search files for lines that contain a specific pattern. By default, the **grep** command displays the entire line when it finds a matching pattern.

Example:

```
[student@OCS ~]$ grep "the" /etc/rsyslog.conf
# To enable high precision timestamps, comment out the following line.
# Set the default permissions for all log files.
```

Table 13.15 describes some important options of the **grep** command.

TABLE 13.15 **grep Command Options**

Option	Description
-c	Displays a count of the number of matching lines rather than displaying each line that matches.
-color	Displays the text that matches in a different color than the rest of the text.
-E	Allows you to use extended regular expressions in addition to basic regular expressions. See Table 13.4 in this chapter for additional details regarding extended regular expressions.
-f	Allows you to use fixed strings so that all characters in the pattern are treated as regular characters, not regular expression characters.
-e	Used to specify multiple patterns in one **grep** command (for example, **grep -e** *pattern1* **-e** *pattern2 file*).
-f *file*	Uses patterns found within the specified *file*.
-i	Ignores case.
-l	Displays filenames that match the pattern rather than displaying every line in the file that matches. This is useful when you're searching multiple files (for example, **grep "the" /etc/***).
-n	Displays the line number before displaying the line.
-r	Recursively searches all the files in a directory structure.

Option	Description
-v	Finds an inverse match (that is, return all lines that don't contain the pattern specified).
-w	Matches whole words only; for example, the command **grep "the"** *file* matches the letters **the** even when part of a larger word such as **then** or **there**, but the command **grep -w "the"** *file* only matches **the** as a separate word.

ExamAlert

For the Linux+ XK0-005 exam, be ready for questions that involve regular expressions as they are commonly used with **grep**. Also know the difference between **grep** and **egrep** (aka, **grep -E**).

egrep

The **egrep** command performs the same function as the **grep -E** command. See the "**grep**" section, earlier in this chapter, for details.

tee

If you want STDOUT to be sent both to the terminal and to a file, send the output to the **tee** command and provide an argument that indicates the file in which you want to store the information. Here's an example:

```
[student@OCS ~]$ ls
[student@OCS ~]$ cal | tee cal.txt
 January 2017
Su Mo  Tu We Th Fr Sa
 1  2   3  4  5   6 7
 8  9  10 11 12 13 14
15 16  17 18 19 20 21
22 23  24 25 26 27 28
29 30  31

[student@OCS ~]$ ls
cal.txt
```

The **tee** command has only a few options. The most useful is the **-a** option. By default, the **tee** command completely overwrites a file, but the **-a** option tells the **tee** command to append to the file.

wc

The **wc** command is used to display the number of lines, words, or characters of data. By default, all three values are displayed:

```
[student@OCS ~]$ wc sample.txt
 2 4 24 sample.txt
```

Table 13.16 describes some important options of the **wc** command.

TABLE 13.16 **wc Command Options**

Option	Description
-c	Displays only the number of bytes. (For text data, a byte is one character.)
-m	Displays only the number of characters.
-l	Displays only the number of lines.
-w	Displays only the number of words.

cut

The **cut** command is used to display "sections" of data. Table 13.17 describes some important options of the **cut** command.

TABLE 13.17 **cut Command Options**

Option	Description
-b	Used to define a section to print by bytes.
-c	Used to define a section to print by characters.
-d	Used (with the **-f** option) to specify a delimiter character.
-f	Used to specify which fields to display.

Here is an example using fields:

```
[student@OCS ~]$ head -2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
```

```
[student@OCS ~]$ head -2 /etc/passwd | cut -d: -f1,7
root:/bin/bash
bin:/sbin/nologin
```

Here is an example using characters:

```
[student@OCS ~]$ ls -l /etc/passwd
-rw-r--r--. 1 root root 2607 Nov 3 10:15 /etc/passwd
[student@OCS ~]$ ls -l /etc/passwd | cut -c1-10,42-
-rw-r--r-- /etc/passwd
```

tr

The **tr** command is useful for translating characters from one set to another. The syntax of the command is **tr** *SET1* **[***SET2***]**.

This example shows how to capitalize the output of the **date** command:

```
[student@OCS ~]$ date
Sat Dec 3 20:15:05 PST 2016
[student@OCS ~]$ date | tr 'a-z' 'A-Z'
SAT DEC 3 20:15:18 PST 2016
```

Note that in order to use the **tr** command on a file, you must redirect the file into the **tr** command, like so, because the **tr** command does not accept files as arguments:

```
tr 'a-z' 'A-Z' < file
```

Table 13.18 describes some important options of the **tr** command.

TABLE 13.18 **tr Command Options**

Option	Description
-d	Used when the second set is omitted; it deletes the matching characters. For example, the following deletes all numbers from the output of the **date** command: **date | tr -d '0-9'**.
-s	Converts repeated matching characters into a single character before translating. Thus, the command **tr -s 'a' 'A'** converts **aaabc** to **abc** and then translates it to **Abc**.

head

The **head** command displays the top part of text data, as shown in this example:

```
[student@OCS ~]$ ls -l | head -3
total 12
drwxrwxr-x. 2 student student 6 Aug 22 16:51 book
drwxrwxr-x. 2 student student 6 Aug 22 16:51 class
```

By default, the top 10 lines are displayed. Use the **-n** option to display a different number of lines.

tail

The **tail** command displays the bottom part of text data, as shown in this example:

```
[student@OCS ~]$ cal 12 1999
 December 1999
Su Mo Tu We Th Fr Sa
          1   2  3 4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

By default, the top 10 lines are displayed. Use the **-n** option to display a different number of lines.

```
[student@OCS ~]$ cal 12 1999 | tail -n 2
26 27 28 29 30 31
```

Table 13.19 describes some important options.

TABLE 13.19 **tail Command Options**

Option	Description
-f	Displays the bottom part of a file and follows changes, which means it continues to display any changes made to the file as data is written to the file.
-n +*x*	Displays from line number *x* to the end of the file.

Environment Variables

Shell variables are used to store information. This information is used to modify the behavior of the shell itself or external commands. Table 13.20 details some common useful shell variables.

TABLE 13.20 **Useful Shell Variables**

Variable	Description
HOME	The current user's home directory.
ID	The current user's ID.
LOGNAME	The username of the user who logged in to the current session.
OLDPWD	The previous directory location (before the last **cd** command).
PATH	The location where commands are found; see the "**$PATH**" section, later in this chapter, for more details.
PS1	The primary prompt.
PWD	The current directory.

The **PS1** variable, for example, defines the primary prompt, often using special character sequences (for example, **\u** = current user's name, **\h** = hostname, **\W** = current directory):

```
[student@OCS ~]$ echo $PS1
[\u@\h \W]\$
```

Note that variables are defined without a dollar sign character but are referenced using the dollar sign character, as shown in this example:

```
[student@OCS ~]$ PS1="[\u@\h \W \!]\$ "
[student@OCS ~ 93]$ echo $PS1
[\u@\h \W \!]$
```

To see all shell variables, use the **set** command, as demonstrated here:

```
[student@OCS ~ 95]$ set | head -5
ABRT_DEBUG_LOG=/dev/null
AGE=25
BASH=/bin/bash
```

```
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extglob:
extquote:force_fignore:
histappend:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
```

When a variable is initially created, it is only available in the shell where it was created. This variable is referred to as a *local variable*. In some cases, you need to pass a variable into a subprocess. To convert an existing local variable to an environment variable, use the **export** command:

```
[student@OCS ~]$ echo $NAME
Sarah
[student@OCS ~]$ export NAME
```

If the variable doesn't already exist, the **export** command can create it directly as an environment variable:

```
[student@OCS ~]$ export AGE=25
```

When a variable is converted into an environment variable, all subprocesses have this variable set. This is useful when you want to change the behavior of a process by modifying a key variable.

For example, the **crontab -e** command allows you to edit your **crontab** file. To choose the editor that the **crontab** command will use, create and export the **EDITOR** variable:

```
export EDITOR=gedit
```

Figure 13.2 provides a visual example of local versus environment variables.

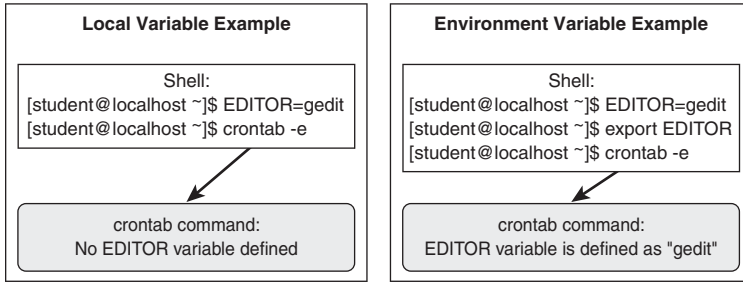

FIGURE 13.2 **Local Versus Environment Variables**

The **export** command can also be used to display all environment variables:

```
export -p
```

The **env** command displays environment variables in the current shell. Local variables are not displayed when the **env** command is executed. Another use of the **env** command is to temporarily set a variable for the execution of a command.

Example:

```
[student@OCS ~]# echo $TZ


[student@OCS ~]# date
Thu Dec 1 18:48:26 PST 2016
[student@OCS ~]# env TZ=MST7MDT date
Thu Dec 1 19:48:31 MST 2016
[student@OCS ~]# echo $TZ


[student@OCS ~]#
```

To unset a variable when executing a command, use the **--unset=VAR** option (for example, **env --unset=TZ date**).

$PATH

Scripts should be located in one of the directories defined by the **$PATH** variable:

```
[root@OCS ~]$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games
```

Often, only the root user can place a file in one of these directories. Regular users can make their own directories and add them to the **$PATH** variable, as shown here:

```
[bob@OCS ~]$ mkdir /home/bob/bin
[bob@OCS ~]$ PATH="$PATH:/home/bob/bin"
[root@OCS ~]$ echo $PATH
```

```
/home/bob/bin: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin:
/usr/games:/usr/local/games
```

$SHELL

The **$SHELL** variable displays the account's login shell. While BASH is the most commonly used Linux shell, it is possible that your account might be using a different shell. Example:

```
[bob@OCS ~]$ echo $SHELL
/bin/bash
```

$?

When a command executes, it returns a success (**0**) or failure (**>1**) value to the shell or script from which it was executed. This status is stored in the **$?** variable. Here's an example:

```
[root@OCS ~]$ ls /etc/skel
[root@OCS ~]$ echo $?
0
[root@OCS ~]$ ls /junk
ls: cannot access /junk: No such file or directory
[root@OCS ~]$ echo $?
2
```

This return value can be used in conditional statements to determine if a command exited successfully:

```
some_command
if [ $? -eq 0 ]
then
 echo "command executed successfully"
else
 echo "command failed"
fi
```

Relative and Absolute Paths

You use a path to refer to a file or directory. If you use a pathname that is relative to the current directory, it is called a *relative path*. Here are some examples:

- **cd test:** Moves to the **test** directory under the current directory.
- **ls abc/xyz:** Lists files in the **xyz** directory, which is under the **abc** directory, which is under the current directory.
- **cd ..:** Moves one level up from the current directory (as the **..** characters represents the directory above the current directory).
- **cp data/abc.txt .:** Copies the **abc.txt** file, which is under the **data** directory, which is under the current directory into the current directory (as a single **.** represent the current directory).

If you use a pathname that is relative to the root directory (**/**), it is called a *relative path*. Here are some examples:

- **cd /etc/skel**
- **ls /usr/bin**
- **cp /etc/hosts /tmp**

Note that an absolute path always begins with the **/** character, and relative paths never begin with the **/** character.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which of the following should be placed on the first line of a BASH script?
   - ❍ **A. /bin/bash**
   - ❍ **B. !/bin/bash**
   - ❍ **C. #/bin/bash**
   - ❍ **D. #!/bin/bash**

2. What concludes a **case** statement?
   - ❍ **A. done**
   - ❍ **B. fi**
   - ❍ **C. esac**
   - ❍ **D.** None of these answers are correct.

3. Which of the following demonstrate valid test conditional statements? (Choose two.)
   - ❍ **A. if [-d /etc]**
   - ❍ **B. while [$name = “Bob”]**
   - ❍ **C. until [$name == “Bob”]**
   - ❍ **D. if [! -d /etc]**

4. Which **sed** feature deletes a line?
   - ❍ **A. ‘/RE\d’**
   - ❍ **B. ‘/RE-d’**
   - ❍ **C. ‘/RE/d’**
   - ❍ **D. ‘/RE=d’**

Cram Quiz Answers

1. **D.** Add this path to the first line of the script by using the following:

   ```
   #!/bin/bash
   ```

2. **C. esac** concludes a **case** statement. **done** concludes **for**, **while**, and **until** loops. **fi** concludes an **if** statement.
3. **B and D.** The answer **if [-d /etc]** is not correct because you need spaces before and after the square bracket characters. The answer **until [$name == “Bob”]** is not correct because **==** is not a valid test operation.
4. **C.** The other answers are not valid syntax.

This page intentionally left blank

CHAPTER 14

Perform Basic Container Operations

This chapter covers the following Linux+ XK0-005 exam objective:

- **3.2:** Given a scenario, perform basic container operations.

Containers are becoming more commonplace in Linux as developers and system administrators are looking for ways to maximize system utilization and provide more robust and secure applications. In this chapter you will learn how to build container images, deploy containers, and manage containers.

This chapter provides information on container management and container image operations.

Container Management

Containers are often compared to virtual machines (VMs). A VM is an operating system that shares a physical system with one or more other operating systems. Access to the system's hardware is managed through special software called a *hypervisor*.

One advantage of a VM is the ability to move a VM from one physical system to another. Another advantage is the ability to clone or duplicate a VM. In order to perform either of these tasks, a VM template is used. A template is a definition of a VM that contains information about the VM.

A container is similar to a VM in that it can appear to act like a separate operating system (OS). However, VMs typically need more hardware resources, whereas containers are more lightweight and run as applications on the host OS. As a result, a system can support many more containers than VMs.

A container is similar to a VM in that they both use images to create a running instance. This image and the running container are managed by a software program, similar to the hypervisor software used to manage VMs.

There are many popular container software solutions, but CompTIA mentions two specifically in the Linux+ XK0-005 exam objectives, under "Linux+ Proposed Hardware and Software List":

- Docker
- Podman

In many ways, Docker and Podman are very similar. The following are some of the main differences between them:

- Docker containers run as the root user, while Podman can run "rootless."
- Docker makes it possible to build container images. This feature isn't available in Podman (where a separate tool called Buildah is required).
- Docker is monolithic in its architecture, and Podman is modular.

ExamAlert

This chapter focuses on Docker as it is, at this writing, much more widely used and more likely to be featured on the Linux+ XK0-005 exam.

ExamAlert

As you have probably realized by now, the sections in each chapter of this book directly mirror the topics on the CompTIA Linux+ XK0-005 exam objectives. Sometimes that makes learning these topics difficult as they are not always in a logical order for learning. If you are learning about containers for the first time, consider skipping to the "Container Image Operations" section, later in this chapter, and reading through that first.

Starting/Stopping

Note

Before you can start an image, you need to deploy it (see the "Deploying Existing Images" section, later in this chapter).

Once a container has been deployed, it can be started and stopped by using the **docker start** and **docker stop** commands, as shown in Figure 14.1.

```
root@ocs.com:~/test$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
   STATUS                      PORTS               NAMES
f084296d59bd        nginx               "/docker-entrypoint.…"   24 seconds ago
   Exited (0) 5 seconds ago                        nginx
d07b50bff81e        ubuntu              "bash"                   35 seconds ago
   Exited (0) 34 seconds ago                       ubuntu
root@ocs.com:~/test$ docker start f084296d59bd
f084296d59bd
root@ocs.com:~/test$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
   STATUS                      PORTS               NAMES
f084296d59bd        nginx               "/docker-entrypoint.…"   35 seconds ago
   Up 2 seconds                0.0.0.0:8080->80/tcp   nginx
d07b50bff81e        ubuntu              "bash"                   46 seconds ago
   Exited (0) 45 seconds ago                       ubuntu
root@ocs.com:~/test$ docker stop f084296d59bd
f084296d59bd
root@ocs.com:~/test$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
   STATUS                      PORTS               NAMES
f084296d59bd        nginx               "/docker-entrypoint.…"   46 seconds ago
   Exited (0) 2 seconds ago                        nginx
d07b50bff81e        ubuntu              "bash"                   57 seconds ago
   Exited (0) 55 seconds ago                       ubuntu
root@ocs.com:~/test$
```

FIGURE 14.1 **Starting and Stopping Containers**

Inspecting

The process of inspecting a container involves gathering detailed information about the container. This is accomplished by using the **docker inspect** command:

```
[root@OCS ~]$ docker inspect 7e18ce199f22 | wc -l
220
```

This command produces a JSON object that contains hundreds of lines of information. While you can read through all of the output to find what you need, it is better to search for the information by using the **grep** command. For example, the output in Figure 14.2 shows the command that is run when the container starts.

```
root@ocs.com:~/test$ docker inspect 7e18ce199f22 | grep Cmd -A 4
            "Cmd": [
                "nginx",
                "-g",
                "daemon off;"
            ],
root@ocs.com:~/test$
```

FIGURE 14.2 **Inspecting Container Details**

For more about JSON format, see Chapter 16, "Common Infrastructure as Code Technologies."

ExamAlert

Given the large number of details provided by the **docker inspect** command, you should not see specific questions about the output that this command provides on the Linux+ XK0-005 exam.

Listing

To list all of the containers that are currently running, run the **docker ps** command, as shown in Figure 14.3.

```
root@ocs.com:~/test$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
      STATUS              PORTS                    NAMES
7e18ce199f22        nginx               "/docker-entrypoint.…"   2 minutes ag
o       Up 2 minutes        0.0.0.0:8080->80/tcp     nginx
```

FIGURE 14.3 **Displaying Only Containers That Are Currently Running**

Output from this command is displayed in columns (often wrapped across multiple lines due to the large amount of information). The columns are:

- **CONTAINER ID:** The unique ID given to this container. This ID can be used to control the container, including starting it, stopping it, and deleting it. In Figure 14.3, this value is **7e18ce199f22**.
- **IMAGE:** The name of the image that was used to create this container. In Figure 14.3, this value is **nginx**.
- **COMMAND:** The command that was executed when the container was started. This is often truncated, as in the example shown in Figure 14.3 ("**/docker-entrypoint. . . .**").
- **CREATED:** When the container was first deployed.
- **STATUS:** Whether the container is running or not and how long it has been in that state.
- **PORTS:** The ports exposed for this container.

- **NAMES:** Alternative ways to refer to the container besides using the CONTAINER ID.

To show all Docker containers, both running and stopped, run the **docker ps -a** command, as shown in Figure 14.4.

```
root@ocs.com:~/test$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
        STATUS                      PORTS                    NAMES
62bfd741ccf7        ubuntu              "bash"                   14 seconds a
go      Exited (0) 12 seconds ago                            ubuntu
7e18ce199f22        nginx               "/docker-entrypoint.…"   2 minutes ag
o       Up 2 minutes                0.0.0.0:8080->80/tcp     nginx
```

FIGURE 14.4 **Displaying All Containers**

ExamAlert

Be very clear about the differences between **docker ps** and **docker ps -a** before taking the Linux+ XK0-005 exam.

Deploying Existing Images

To deploy a container from an image, use the following command:

```
docker run –name NAME -p PORTS IMAGE
```

The arguments for this command are:

- **--name NAME:** This is a name that you choose to give the container. It can be the same as the name of the image.
- **-p PORTS:** The port that you want to expose. For example, if the **PORTS** value is **8080:80**, this means the port 8080 on your localhost would connect to port 80 of the container.
- **IMAGE:** This is the name of the image, as provided when you run the **docker images** command. See the "**list**" section, later in this chapter, for more details about listing images.

Figure 14.5 shows an example of the **docker run** command in action.

```
root@ocs.com:~/test$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
   SIZE
ubuntu              20.04               ff0fea8310f3        7 days ago
   72.8MB
nginx               latest              f2f70adc5d89        7 days ago
   142MB
root@ocs.com:~/test$ docker run --name nginx -p 8080:80 nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to pe
rform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-de
fault.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/con
f.d/default.conf
```

FIGURE 14.5 **docker run Example**

Note that when this command is executed as shown in Figure 14.5, the container deploys in attached mode. This means it runs as a foreground process. The advantage of this is that the container is easy to stop by holding down the Ctrl key and pressing the c key on your keyboard.

If you want to deploy the container in detached mode, use the **-d** option. If you have already deployed the container and want to run it in detached mode, you have two options:

- Stop the container by using Ctrl+c, list the available containers (using **docker ps -a**), and then use the **docker start** command to start the container. See the "Listing" and "Starting/Stopping" sections, earlier in this chapter, for details on these commands.
- Stop the container by using Ctrl+c, delete the container, and then deploy the container again by using the **docker run** command with the **-d** option. Figure 14.6 demonstrates this technique.

```
2022/03/25 05:13:31 [notice] 1#1: worker process 30 exited with code 0
2022/03/25 05:13:31 [notice] 1#1: exit
root@ocs.com:~/test$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
        STATUS                      PORTS               NAMES
fbe903fab8a1        nginx               "/docker-entrypoint.…"   9 minutes ag
o       Exited (0) 19 seconds ago                       nginx
root@ocs.com:~/test$ docker rm fbe903fab8a1
fbe903fab8a1
root@ocs.com:~/test$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
   STATUS              PORTS               NAMES
root@ocs.com:~/test$ docker run -d --name nginx -p 8080:80 nginx
7e18ce199f2264e6cf0c72c144f805ee91527cefc330272d6a681889a1026dd4
root@ocs.com:~/test$
```

FIGURE 14.6 **Stopping, Removing, and Then Deploying a Container in Detached Mode**

Connecting to Containers

You can establish a connection to a running container by using the **docker exec** command in the docker container. This command can be used to open a BASH shell in the container, as shown in Figure 14.7.

```
root@ocs.com:~/test$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                    NAMES
f084296d59bd        nginx               "/docker-entrypoint.…"   About a minute ag
o   Up 3 seconds        0.0.0.0:8080->80/tcp   nginx
root@ocs.com:~/test$ docker exec -it f084296d59bd bash
root@f084296d59bd:/#
```

FIGURE 14.7 **Connecting to a Container**

To break the connection, just exit the BASH shell by entering the **exit** command.

Logging

You can display the logs of a container by using the **docker logs** command, as shown in Figure 14.8.

```
root@ocs.com:~/test$ docker logs f084296d59bd
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
 configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default
.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/d
efault.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d
/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/03/25 05:49:00 [notice] 1#1: using the "epoll" event method
2022/03/25 05:49:00 [notice] 1#1: nginx/1.21.6
2022/03/25 05:49:00 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/03/25 05:49:00 [notice] 1#1: OS: Linux 5.4.0-37-generic
2022/03/25 05:49:00 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/03/25 05:49:00 [notice] 1#1: start worker processes
2022/03/25 05:49:00 [notice] 1#1: start worker process 30
2022/03/25 05:49:00 [notice] 1#1: start worker process 31
2022/03/25 05:49:18 [notice] 1#1: signal 3 (SIGQUIT) received, shutting down
2022/03/25 05:49:18 [notice] 31#31: gracefully shutting down
2022/03/25 05:49:18 [notice] 30#30: gracefully shutting down
```

FIGURE 14.8 **Displaying the Logs of a Container**

Exposing Ports

See the "Deploying Existing Images" section, earlier in this chapter.

Container Image Operations

Container images are used to build containers. While you can create your own images, there are also many images available from locations called *repositories*. In this section, you will learn how build and manage your own images.

build

To build your own Docker image, start by creating a *Dockerfile* like the following:

```
# Use the official Ubuntu 20.04 as base
FROM ubuntu:20.04
# Install nginx and curl
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y nginx curl
```

In this example, the Ubuntu 20.04 basic image will be used as the base for the new image (because of the **FROM** statement). Then **nginx** and **curl** will be added after all of the packages are updated and upgraded (because of the **RUN** statements).

> **ExamAlert**
>
> You don't need to be an expert on Dockerfile for the Linux+ XK0-005 exam, but you should know the statements that are provided in Table 14.1 at the end of this section.

Once the Dockerfile has been created, use the following command to create the image:

```
docker build -t my-image:1.0 .
```

The option to the **-t** command should contain two bits of information: the name of the image (**my-container** in this example) and the version of the image (**1.0** in this case). A colon should separate the two bits of information.

The **.** (dot, or period) character indicates that the image should include all of the files and directories that appear in the current directory. You can also create a **.dockerignore** file and list the files and directories that should not be placed in the image.

When the process is complete, you can verify that the container was built by using the **docker images** command:

```
[root@OCS ~]$ docker images
REPOSITORY            TAG      IMAGE ID       CREATED          SIZE
my-image              1.0      d2c9860a64ae   20 minutes ago   177MB
```

There are several different statements you can place in a Dockerfile. Table 14.1 highlights some of the most important of them.

TABLE 14.1 **Statements You Can Place in a Dockerfile**

Statement	Description
FROM	Specifies a base image.
RUN	Executes a command, such as to install a package.
COPY	Copies files or directories to a specific location.
ENTRYPOINT	Used when the container for an image is started.
CMD	Specifies arguments to pass to the **ENTRYPOINT** command.
EXPOSE	Opens a port to access the container.

push

Recall that Docker images can be stored in a *repository*, also known as a *registry*. The process of pushing an image involves placing the image (or a new version of an image) to the registry. To accomplish this, first tag the image with the registry as shown here, replacing *registry_name* with the hostname of the system that houses the registry and *path_to_registry* with the directory where the image should be stored:

```
docker image tag my-container:1.0
registry_name:5000/path_to_registry/my-image:1.0
```

Then push the image with a command like the following:

```
docker image push registry_name:5000/ path_to_registry/my-image:1.0
```

In the previous commands, **5000** represents the port that the registry server is listening on, and **my-image:1.0** is the image name and tag.

> **Note**
>
> It is common to use the tag **latest** to indicate the latest version of an image.

pull

You can pull an image from a repository by using the **docker pull** command, as shown in the following example:

```
$ docker pull ubuntu
```

list

You can list the Docker images that are available on the current system by using the **docker images** command:

```
[root@OCS ~]$ docker images
REPOSITORY            TAG      IMAGE ID       CREATED          SIZE
my-image              1.0      d2c9860a64ae   20 minutes ago   177MB
```

> **Note**
>
> Another commonly used method, which produces the same output as **docker images**, is the command **docker image ls**.

rmi

To remove a docker image, use the **docker rmi** command, as shown in the following example:

```
[root@OCS ~]$ docker images
REPOSITORY           TAG      IMAGE ID       CREATED          SIZE
my-image             1.0      d2c9860a64ae    20 minutes ago  177MB
ubuntu               latest   ff0fea831of3    6 days ago      72.8MB
[root@OCS ~]$ docker rmi ubuntu:latest
Untagged: ubuntu:latest
```

```
[root@OCS ~]$ docker images
REPOSITORY          TAG     IMAGE ID       CREATED          SIZE
my-image            1.0     d2c9860a64ae   20 minutes ago   177MB
```

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which command provides detailed information about a container?

   - ❍ **A. docker ls**
   - ❍ **B. docker ps -a**
   - ❍ **C. docker inspect**
   - ❍ **D. docker details**

2. Which command displays only the Docker containers that are currently running?

   - ❍ **A. docker ps**
   - ❍ **B. docker ps -a**
   - ❍ **C. docker ps --running**
   - ❍ **D.** None of these answers are correct.

3. What option is missing in the following command?

   ```
   docker run -name test _____ 8080:80 nginx
   ```

   - ❍ **A. --ports**
   - ❍ **B. --port**
   - ❍ **C. -P**
   - ❍ **D. -p**

4. What is missing from the following Dockerfile?

   ```
   # Use the official Ubuntu 20.04 as base
   _____ ubuntu:20.04
   # Install nginx and curl
   RUN apt-get update
   RUN apt-get upgrade -y
   RUN apt-get install -y nginx curl
   ```

   - ❍ **A. BASE**
   - ❍ **B. FROM**

- ❍ **C. START**
- ❍ **D.** None of these answers are correct.

Cram Quiz Answers

1. **C.** The process of inspecting a container involves gathering detailed information about the container. This is accomplished by using the **docker inspect** command. The **docker ps -a** command lists basic information about all Docker containers on the system. The other answers are not valid **docker** commands.
2. **A.** The **docker ps -a** command shows all containers, not just running containers, as the **docker ps** command does. **docker ps –running** is not a valid command.
3. **D.** The **-p** option is used to indicate which ports to expose for the containers. The rest of the options are not valid for the **docker run** command.
4. **B. FROM** is used to specify a base image. The other answers are not valid for a Dockerfile.

CHAPTER 15

Perform Basic Version Control Using Git

This chapter covers the following Linux+ XK0-005 exam objective:

- **3.3:** Given a scenario, perform basic version control using Git.

One of the biggest headaches that developers must deal with is different versions of source code. There are times when you just need to go back to a previous version of code. Maintaining versions manually can be cumbersome and time-consuming.

Compounding the problem, multiple programmers often work together on a single piece of source code. A large program can consist of tens of thousands of lines of code, and different programmers may be responsible for different portions of the code.

Version control software like Git can handle the complicated task of maintaining different versions of source code. This chapter introduces you to Git and common Git operations.

This chapter provides information on the following topics: version control basics, **clone**, **push**, **pull**, **commit**, **add**, **checkout**, **branch**, **tag**, and **gitignore**.

Introduction to Version Control and Git

This section does not specifically cover any exam objectives. However, to understand the rest of this chapter, it is very helpful to have a basic knowledge of version control solutions. If you already have a firm grasp of this topic, consider skipping ahead to the "**clone**" section.

To understand Git and the concept of version control, it is helpful to look at version control from a historical perspective. There have been three generations of version control software (VCS).

The first generation of VCS was very simple. Developers worked on the same physical system and "checked out" one file at a time. This generation of version control software made use of a technique called *file locking*. When a developer checked out a file, it was locked, and no other developer could edit the file. Examples of first-generation VCS include Revision Control System (RCS) and Source Code Control System (SCCS). Figure 15.1 illustrates the concept of this type of version control.

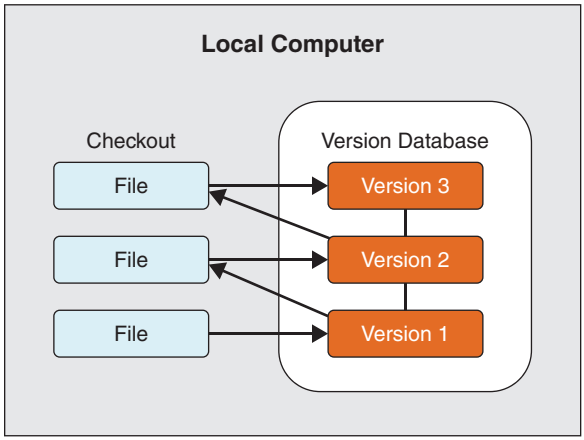

FIGURE 15.1 **First-Generation VCS**

There are several problems with first-generation VCS, including the following:

- Only one developer can work on a file at a time. This creates a bottleneck in the development process.
- Developers have to directly log in to the system that contains the version control software.

These problems were solved in the second generation of version control software. In second-generation VCS, files are stored on a centralized server in a repository. Developers can check out separate copies of a file. When a developer completes work on a file, the file is checked in to the repository. Examples of second-generation version control software include Concurrent Versions System (CVS) and Subversion. Figure 15.2 illustrates the concept of this type of version control.

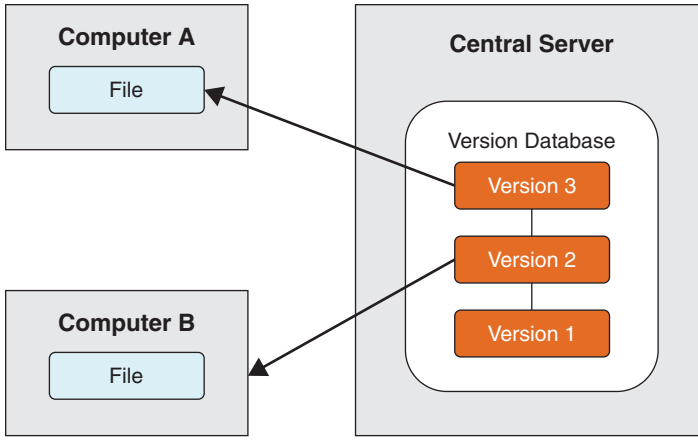

FIGURE 15.2 **Second-Generation VCS**

If two developers check out the same version of a file, there is the potential for issues. This is handled by a process called a *merge*.

The Third Generation

The third generation of VCS is referred to as Distributed Version Control Systems (DVCS). As with the second generation, there is a central repository server that contains all of the files for a project; however, developers don't check out individual files from the repository. Instead, the entire project is checked out, allowing the developer to work on the complete set of files rather than just individual files. Figure 15.3 illustrates the concept of this type of version control.

Git is an example of third-generation VCS. One of the challenges of using Git is understanding the concepts behind it. If you don't understand the concepts, then all of the commands seem like some sort of black magic. This section focuses on critical Git concepts and introduces you to some of the basic commands.

To get a copy of a project from a Git repository, you use a process called *cloning*. It is very important to understand that cloning means checking out an entire project and that most of the work you do is local to the system that you are working on. The files that are checked out are placed in a directory under your home directory.

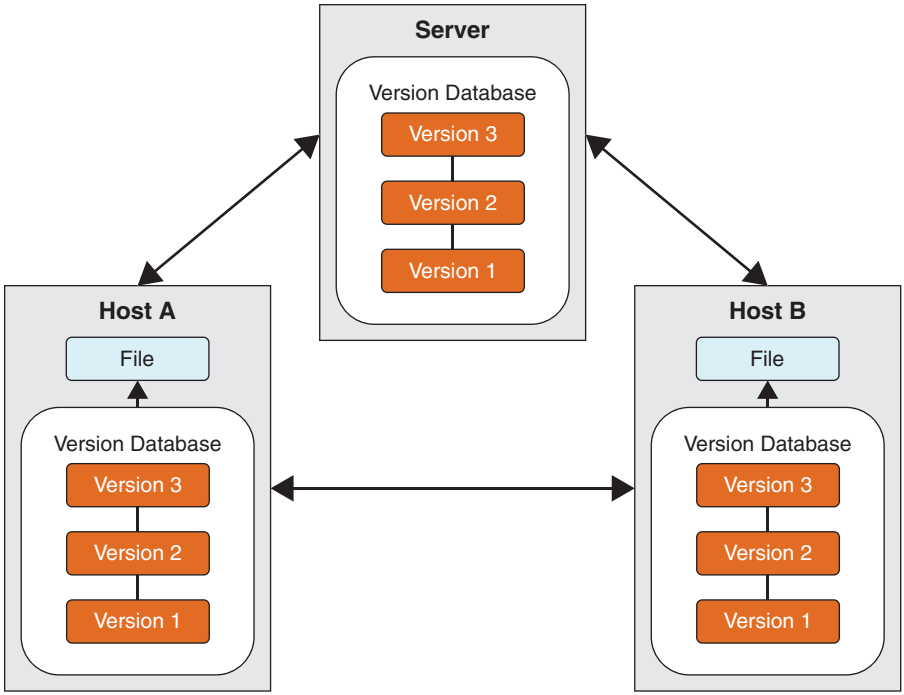

FIGURE 15.3 **Third-Generation VCS**

Cloning doesn't just create a copy of all of the files from the repository. It actually performs three primary functions:

- Cloning creates a local repository of the project under the *project_name*/**.git** directory in your home directory. The files of the project in this location are considered to be "checked out" from the central repository.
- Cloning creates a directory where you can directly see the files. This is called the "working area." Changes made in the working area are not immediately version controlled.
- Cloning creates a staging area. The staging area is designed to store changes to files before you commit them to the local repository.

If you were to clone a project called **Jacumba**, the entire project would be stored in the **Jacumba/.git** directory under your home directory. You should not attempt to modify these files from the project directly. Instead, look directly in the **~/Jacumba** directory, and you will see the files from the project. These are the files that you should change.

Suppose you make a change to a file, but you have to work on some other files before you can commit changes to the local repository. In such a case, you would *stage* the file that you have finished working on; that is, you would prepare it to be committed to the local repository.

After you have made all changes and have staged all files, you can commit them to the local repository. Figure 15.4 illustrates this process.

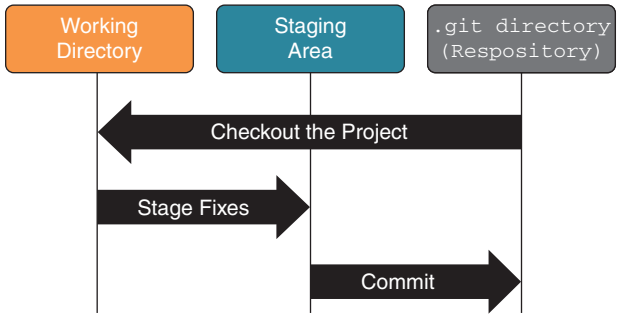

FIGURE 15.4 **Git Stages**

It is important to realize that committing the staged files only sends them to the local repository. This means that only you have access to the changes that have been made. The process of "checking in" the new versions to the central repository is called a *push*.

> **Note**
>
> The steps described in this section are explained in greater detail later in this chapter. Consider this section to be an introduction to the concepts that will help you better understand the process when the Git commands are introduced.

clone

You might need to perform basic configuration of Git before you can work on any projects. The **git config** command is used to configure the **git** utility. Here are two examples showing some commonly used options:

```
$ git config --global user.name "Bo Rothwell"
$ git config --global user.email bo@onecoursesource.com
```

The **--global** option results in configuration options being stored in a configuration file in the user's home directory, as shown here:

```
$ more ~/.gitconfig
[user]
        name = Bo Rothwell
        email = bo@onecoursesource.com
```

The **git** command uses the settings in this file by default. You can also store these settings in the current repository directory (the **.git** directory under the current directory) by using the **--local** option. Local settings override global settings.

There are dozens of additional settings, but most users of the **git** command just set **user.name** and **user.email**.

The **git clone** command creates a local repository from the contents of a remote repository. The argument provided is the location of the remote repository, as shown in this example:

```
$ git clone https://gitlab.com/borothwell/ocs.git
Cloning into 'ocs'...
Username for 'https://gitlab.com': borothwell
Password for 'https://borothwell@gitlab.com':
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

Table 15.1 outlines some important options of the **git clone** command.

TABLE 15.1 **Important git clone Command Options**

Option	Description
-v	Enters verbose mode.
--single-branch	Specifies to clone only a single branch, not the entire repository.

Cloning isn't the only way to create a Git project. To create a new repository in the current local directory, use the **git init** command:

```
$ git init test
Initialized empty Git repository in /tmp/test/.git/
```

ExamAlert

The **git init** command is not specifically mentioned in the Linux+ XK0-005 exam objectives. However, it is very commonly used, and CompTIA includes the following in the "Please note" section of the exam objectives: "The lists of examples provided in bulleted format are not exhaustive lists. Other examples of technologies, processes, or tasks pertaining to each objective may also be included on the exam although not listed or covered in this objectives document."

push

The **git commit** command updates only the local repository. To have changes from the local repository sent to the remote repository, use the **git push** command, as shown in this example:

```
$ git push -u origin main
Username for 'https://gitlab.com': borothwell
Password for 'https://borothwell@gitlab.com':
Counting objects: 4, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 370 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://gitlab.com/borothwell/ocs.git
 12424f5..3b36054 main -> main
Branch main set up to track remote branch main from origin.
```

Note

In the last line of this output, **origin** refers to the remote repository, and **main** refers to the branch that should be updated.

ExamAlert

The **git push** command attempts to merge changes into existing files. If this can't be handled automatically, you may have to perform a manual merge operation. See the "**branch/checkout**" section, later in this chapter, for more details.

pull

If changes have been made to the remote repository and you want to download those changes into the local repository, use the **git pull** command:

```
$ git pull origin main
```

Note

origin refers to the remote repository, and **main** refers to the branch that should be updated.

ExamAlert

The **git pull** command attempts to merge changes into existing files. If this can't be handled automatically, you may have to perform a manual merge operation. See the "**branch/checkout**" section, later in this chapter, for more details.

commit

To have changes made to the working directory placed in the local repository, you first have to add the changes to the staging area and then commit it to the repository, like so:

```
$ git add first.sh
$ git commit -m "added first.sh"
[main 3b36054] added first.sh
 1 file changed, 5 insertions(+)
 create mode 100644 first.sh
```

The **git add** command places the file in the staging area. The **git commit** command takes all of the new files in the staging area and commits them to the local repository. The **-m** option is used to add a message (in this example, the reason for the commit).

add

See the previous section, "commit," for information about **git add**.

branch/checkout

When you first create a project, the code is associated with a branch called **main**. If you want to create a new branch, execute the **git branch** command:

```
$ git branch test
```

This doesn't mean you are suddenly working in the new branch. As you can see from the output of the **git status** command, the **git branch** command doesn't change your current branch:

```
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working directory clean
```

The first line of this output, **On branch main**, indicates that you are still working in the main branch. To switch to the new branch, execute the **git checkout** command:

```
$ git checkout test
Switched to branch 'test'
```

> **ExamAlert**
>
> You can create a branch and switch to it by using the **-b** option to the **git checkout** command: **git checkout -b test**.

Switching actually does two things:

- Causes any future commits to occur on the test branch
- Makes the working directory reflect the test branch

The second item makes more sense with a demonstration. First, observe the following commands, which cause a new version of the **hidden.sh** file to be placed in the test branch repository:

```
$ git add hidden.sh
$ git commit -m "changed hidden.sh"
[test ef2d7d5] changed hidden.sh
 1 file changed, 1 insertion(+)
```

Note what the file looks like in the current working directory:

```
$ more hidden.sh
#!/bin/bash
#hidden.sh


echo "Listing only hidden files:"
ls -ld .* $1
```

If the project is now switched back to the main branch, you can see how the **hidden.sh** file in the working directory is different (note the missing **echo** line, which was added for the test branch only):

```
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)
$ more hidden.sh
#!/bin/bash
#hidden.sh


ls -ld .* $1
```

Suppose you create a new branch to add a new feature to a file. After testing this new feature, you are ready to implement it in the main branch. To do this, you need to merge the content from the new branch (the **feature127** branch in the following example) into the main branch. Start by committing all changes in the **feature127** branch and then switch back to the main branch:

```
$ git commit -a -m "feature added to showmine.sh"
[feature127 2e5defa] feature added to showmine.sh
```

```
 1 file changed,    5 insertions(+), 2 deletions(-)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 3 commits.
   (use "git push" to publish your local commits)
```

You must be in the branch that you want to merge into in order to correctly run the next command. The following **git merge** command merges the changes from the **feature127** branch into the main branch:

```
$ git merge feature127
Updating 4810ca8..2e5defa
Fast-forward
 showmine.sh | 7 +++++--
 1 file changed, 5 insertions(+), 2 deletions(-)
```

This merge process can be rather complex. For example, consider the following command's output:

```
$ git merge main
Auto-merging showmine.sh
CONFLICT (content): Merge conflict in showmine.sh
Auto-merging hidden.sh
CONFLICT (content): Merge conflict in hidden.sh
Automatic merge failed; fix conflicts and then commit the result.
```

You can see that the merge was not completed because the automated merge process ran into some conflicts. You can also see these conflicts by executing the **git status** command:

```
$ git status
On branch test
Your branch is ahead of 'origin/test' by 1 commit.
 (use "git push" to publish your local commits)


You have unmerged paths.
 (fix conflicts and run "git commit")
```

```
Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified: hidden.sh
        both modified: showmine.sh

no changes added to commit (use "git add" and/or "git commit -a")
```

One way to handle these conflicts is to use the **git mergetool** command (but note that you need to install the **git-all** package if you want to use the **git mergetool** command):

```
$ git mergetool showmine.sh

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse dif-
fmerge ecmerge p4merge araxis bc3 codecompare emerge vimdiff
Merging:
showmine.sh

Normal merge conflict for 'showmine.sh':
     {local}: modified file
     {remote}: modified file
Hit return to start merge resolution tool (vimdiff):
```

This tool opens the **vimdiff** utility to allow you to make the necessary changes to the file in order to resolve the conflicts.

ExamAlert

The **git mergetool** command is not specifically mentioned in the official 005 exam guide. However, it is very commonly used in conjunction with merging, and CompTIA includes the following in the "Please note" section of the exam objectives: "The lists of examples provided in bulleted format are not exhaustive lists. Other examples of technologies, processes, or tasks pertaining to each objective may also be included on the exam although not listed or covered in this objectives document."

tag

In a large project, you might find it useful to associate a specific development point with a tag. For example, if you release version 4.1 of your software, you can tag the current state of your project using a command like the following:

```
$ git tag -a v4.1 -m "Release 4.1"
```

The **-a** option is what adds the tag to the current state of the Git repository. The **-m** option allows you to provide a more detailed message.

Use the **git tag** command with no arguments to list the available tags:

```
$ git tag
v1.0
v2.0
v3.0
v4.0
v4.1
```

To see details of a specific tag, use the **git show** command:

```
$ git show v4.1
tag v4.1
Tagger: Bo Rothwell <bo@onecoursesource.com>
Date:   Fri Mar 25 14:27:55 2022 -0600

Release 4.1

commit ca82a6dff817ec66f44342007202690a93763949
Author: Bo Rothwell <bo@onecoursesource.com>
Date:   Mon Mar 21 10:14:13 2022 -0600
```

You can also revert the current view of the project to a tag specific tag:

```
$ git checkout v2.0
```

To return to the latest version of the project (called the HEAD), use the following command:

```
$ git checkout -
```

gitignore

To have **git** commands ignore a file, create a file named **.gitignore** in the working directory and place the filename to ignore inside this file. Here is an example:

```
$ touch notes
~/ocs$ git status -s
?? notes
$ vi .gitignore #added notes as shown below:
$ cat .gitignore
notes
$ git status -s
?? .gitignore
```

Notice that the **.gitignore** file itself must also be placed in the **.gitignore** file, as shown here:

```
$ git status -s
?? .gitignore
$ vi .gitignore #added .gitignore as shown below:
$ cat .gitignore
notes
.gitignore
$ git status -s
```

ExamAlert

You can also use wildcard characters (*****, **?**, and **[***range***]**) in the **.gitignore** file to match a collection of files.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which command is the correct one to use to clone a Git project from a repository?
   - ❍ **A. git init https://gitlab.com/borothwell/ocs.git**
   - ❍ **B. git clone http://gitlab.com/borothwell/ocs.git**

- ❍ **C. git clone https://gitlab.com/borothwell/ocs**
- ❍ **D. git clone https://gitlab.com/borothwell/ocs.git**

2. What is missing from the following command?

```
git pull _____ main
```

- ❍ **A. repo**
- ❍ **B. source**
- ❍ **C. origin**
- ❍ **D. original**

3. Which of the following commands switches the current branch to a branch called **version1**?

- ❍ **A. git branch version1**
- ❍ **B. git change version1**
- ❍ **C. git version1**
- ❍ **D. git checkout version1**

4. In which file can you place file and directory names that **git** should ignore when making changes to the repository?

- ❍ **A. gitignore**
- ❍ **B. .gitignore**
- ❍ **C. ignore**
- ❍ **D. .ignore**

Cram Quiz Answers

1. **D. git init** is incorrect because that command creates a new repository; it does not clone an existing one. The other two answers are incorrect either because of the wrong protocol (**http** instead of **https**) or because of the missing **.git** at the end of the project name.
2. **C.** The other answers are invalid for the **git pull** command.
3. **D.** There is no **git change** command. The **git branch** command creates a new branch; it does not change to a different branch (at least not without another option). The final incorrect answer produces an error message because it is missing a **git** command.
4. **B.** To have **git** commands ignore a file, create a file named **.gitignore** in the working directory and place the filename to ignore inside this file. The other answers have no impact on how **git** modifies the repository.

This page intentionally left blank

CHAPTER 16

Common Infrastructure as Code Technologies

This chapter covers the following Linux+ XK0-005 exam objective:

- **3.4:** Summarize common infrastructure as code technologies.

Imagine a scenario in which your manager tells you that you need to deploy some virtual machines in your organization's infrastructure to give your organization's developers a testing platform for the code they are writing. Initially the developers will need 100 virtual machines, all of which have similar components, but there are some components in each virtual machine that need to be different (user accounts, passwords, software installation, storage space size, and so on).

You are also informed that as the project progresses, the developers will need additional virtual machines—again all similar to one another but with a few customizations. You start to suspect that creating these virtual machines might become a full-time task. In addition to the time it will take to manually configure each virtual machine, you are concerned that you might make mistakes along the way, resulting in mis-configured systems that will cause headaches for the developers. Fortunately for you, there is a better solution.

Infrastructure as code (IaC) is a process in which you can use configuration files or scripts to deploy and manage systems, including virtual machines and containers. Utilities called *orchestration* or *configuration management* tools are used to not only deploy new systems but maintain systems (install new software, patch the operating system, and so on). This chapter focuses on the features of IaC and the utilities that can be used to implement IaC solutions.

ExamAlert

It is important to realize for the Linux+ XK0-005 exam that the objective that matches this chapter asks you to "Summarize common infrastructure as code technologies." For the exam, you should not be asked any questions on configuration or administration of these technologies. IaC and the utilities that manage systems are very large topics, and there are several certification exams that focus on just these technologies. An individual who is Linux+ certified should understand the concepts behind these technologies but should not necessarily be an expert. As a result, the coverage of this topic in this chapter is designed to match the awareness of the topic that is required to pass the Linux+ exam.

This chapter provides information on the following topics: file formats, utilities, continuous integration/continuous deployment (CI/CD), and advanced Git topics.

File Formats

IaC technologies often make use of text files to store data and configurations. The most commonly used file formats are JavaScript Object Notation (JSON) and YAML Ain't Markup Language (YAML), which are covered briefly in this section.

JavaScript Object Notation (JSON)

Some infrastructure as code technologies make use of JavaScript Object Notation (JSON) file format to store or transfer data. JSON is a simple structure that makes use of the following primary data formats: objects (a key/value pairs), arrays (also called lists), strings, and numbers. There are also three keyword values: **true**, **false**, and **null**.

Here is a sample JSON file:

```
{
     "name": "Sue Smith",
     "phone": ["555-555-5555", "555-555-5556"],
     "address": {
             "street": "123 Main St.",
              "city": "Anytown",
              "state": "CA"
      }
}
```

YAML Ain't Markup Language (YAML)

YAML is similar to JSON, and some IaC technologies make use of the YAML file format to store data. It has a similar data structure to JSON, but there are some noticeable difference. JSON uses characters such as **{ }** and **[ ]** to define data structures like objects and arrays, whereas YAML uses indention to create these structures.

Here is a sample YAML file:

```
---
name: Sue Smith
phone:
- 555-555-5555
- 555-555-5556
address:
 street: 123 Main St.
 city: Anytown
 state: CA
```

You can contrast this example with the example in the "JavaScript Object Notation (JSON)" section of this chapter to see how the two file formats differ.

> **Note**
>
> YAML was originally an acronym that stood for "Yet Another Markup Language." XML and HTML are other examples of markup languages. Over time, the goal of YAML changed to being a "human-friendly data serialization standard," so YAML was converted into a backronym that now stands for "YAML Ain't Markup Language."

Utilities

Configuration management has long been a standard process that system administrators have manually handled. Because it was not automated, this process was often time-consuming and resulted in misconfigured systems.

The purpose of automated configuration management is to ensure that software, services, and systems are maintained in a consistent, predictable state. By using automated configuration management tools, a large amount of the workload is handled by automated processes. This also results in fewer errors compared to using manual configuration management operations.

You will often see the terms *orchestration* and *automation* when working with configuration management utilities. Infrastructure automation is a component of orchestration. While many people use the terms *orchestration* and *automation* interchangeably, there is a difference between them. You can certainly have orchestration without automation, but often a primary goal of orchestration is to automate a process.

Consider the following orchestration scenario: A user fills in fields of data in a web-based form. The data is used to populate attributes. An *attribute* is used to define parameters that are used to customize the automation procedure. These attributes contain data that can (and likely will) be different for each orchestration process. For example, suppose the procedure is designed to set up a system that requires access to the network. Attributes would need to be defined for things like the system's IP address, subnet mask, gateway, and router.

The user then clicks an OK button, and a procedure is performed to orchestrate a service. This is orchestration without automation.

Now consider this scenario: An agent is monitoring a system when a threshold is reached (say, maximum CPU usage in an hour). This triggers a procedure using attributes that are populated automatically from a database to spawn a second instance of the services. No human interaction took place; the orchestration process handed everything automatically. This is orchestration with automation.

This section covers the basics of commonly used configuration management utilities.

Ansible

Ansible is a very popular utility that can be used for orchestrating new systems. Some of the features of Ansible are as follows:

- **Agentless:** Ansible is an agentless solution, which means no software is installed on a system. An agentless system may either receive data from the system or conduct remote queries to determine if a change has taken place.
- **Playbook:** An Ansible playbook is a set of instructions to perform that is stored in a file. A playbook file is in YAML format.
- **Ad hoc commands:** Ansible allows you to execute individual commands without having to create a playbook. These are called ad hoc commands.
- **Roles:** Roles can be established to limit the functionality of Ansible operations.
- **Variables:** Variables are used to allow for greater flexibility within playbooks.

Puppet

You can use Puppet to manage servers and automate the process of configuring servers. With Puppet you can define a desired state of a system and then have Puppet perform the operations necessary to get the system to the desired state. Some of the key concepts and features of Puppet include:

- **Agent-based:** In order to use Puppet, client applications (agents) need to be installed on client systems.
- **Puppet master:** This is the software that is installed on the main Puppet server.
- **IaC:** Puppet is an IaC technology, which is a DevOps practice of treating the infrastructure of an organization as you would treat code.
- **Idempotency:** This concept means that you should be able to apply operations and be assured of the end results. Puppet uses this concept to ensure that if a system is manually modified, Puppet can bring the system back to a "normal" state automatically.
- **Puppet manifests:** Puppet has its own language for describing what actions to take on systems. This code is written in files called manifests.

Chef

As with many of the tools discussed in this section, Chef is used for configuration management of systems and provides the means to automate this management process. It uses an IaC design to accomplish these goals. Some of the key concepts and features of Chef include:

- **Cookbooks:** A cookbook is a file in which you describe recipes, which in turn describe the state of resources (nodes). A cookbook can also include other features, such as Chef libraries, metadata, and attributes.
- **Node:** A node is a device for which Chef maintains the configuration. Chef can support many different node types, including network devices, on-premises and cloud devices, and virtual or physical devices.
- **Chef client:** This is an application that must run on each node in order for Chef to maintain the configuration of the node.
- **Chef server:** A Chef server is a system where you store cookbooks. A Chef server works with a Chef client to maintain the configuration of nodes.

- **Chef workstation:** If you don't want to configure a Chef server for a single node, you can install a Chef workstation, which provides the features of Chef on a single system.
- **Chef supermarket:** This is a repository of cookbooks that are created by the Chef community and free to use.

SaltStack

SaltStack is another configuration management system that is used to maintain remote nodes based on defined states. It contains the following components and features:

- **Salt master:** This is the service that runs on the primary Salt server.
- **Salt minion:** This is a server that performs actions that are defined by the Salt master.
- **Target:** A list of one or more minions. The Salt master uses targets to issue commands to the minions.
- **Salt state:** A method of configuring minions by indicating what state a minion should be in. Salt states are defined in YAML files.

Terraform

As with the other configuration management tools in this section, Terraform is a tool that can be used to maintain the state of a system. Terraform is based on a life cycle that consists of the following:

- **Init:** The initialization process uses a working directory that contains the configuration files.
- **Plan:** A plan of action is created to ensure that the configuration can be carried out.
- **Apply:** The plan is executed.
- **Destroy:** This step in the life cycle is where old resources are deleted.

Continuous Integration/Continuous Deployment (CI/CD)

Many software vendors use a traditional software release approach, in which software is released at specific intervals. For example, an organization might

have a yearly release cycle so that each year a new major release of the software occurs. During the year, there might also be minor releases or bug fix releases, depending on potential flaws or security holes that are discovered in the software. The advantage of this method of releasing software is that normally the organization gets time to test the software before it is released, which should result in a more stable program/product. However, the disadvantage of this release method is that new features can take up to a year before they are released, and a competitor may be able to capture more market share if their software is released more often.

Continuous integration (CI) and continuous deployment (CD) are two processes that together result in rapid and continuous release of software. The two terms are often confused, likely because the word "continuous" appears in both of them. They differ in these important ways:

- Continuous integration is a process followed by developers (typically software developers). A software program that uses a version control utility (see Chapter 15, "Perform Basic Version Control Using Git") employs a main branch of development that is used when the software is released to customers. Developers don't work directly on the main branch; rather, they all work on their own branches. With continuous integration, developers merge the changes in their branch back into the main branch as often as possible (not exactly continuously, although that is the goal), and the changes are validated through some automated testing process. This results in the main release branch being continuously validated and updated.
- While continuous integration means the main branch of development is often updated, it doesn't mean that customers have access to the changes. Continuous deployment is a process that extends continuous integration to deploying new releases as they are made available on the main branch (after testing has occurred). A similar process, called continuous delivery, is used to deploy new releases rapidly, but this process requires a human to approve the release.

Advanced Git Topics

Because Git (discussed in Chapter 15) is such a robust tool, complete coverage of Git would require a complete book. Chapter 15 provides some of the basics of Git, and this section dives into a few of the more advanced Git features that are covered on the Linux+ XK0-005 exam.

merge

See the "**branch/checkout**" section of Chapter 15.

rebase

The **git rebase** command applies all the changes from a branch to the current branch in a process called *patching*.

The idea of patching is that you will discover situations in which it won't be easy to perform a simple merge between two different branches. There are several reasons this might be true, including the following:

- You might want to implement changes from a branch that hasn't been made available on the central repository.
- You might want to implement changes from a specific version of a file.

There are actually several different techniques that can be used to perform patching. The most basic (and common) method is to use the **git diff** command and redirect the output to a file, as in this example:

```
git diff A B > file.patch
```

This patch file will be used to modify an existing checked-out file to include the differences. This modification is performed by using the **git apply** command, as in this example:

```
$ git apply file.patch
```

Often a patch file is generated on one system and then copied to another system before it is applied. It is also typically a good idea to use the **--check** option to test a patch before actually applying it, as shown in this example:

```
git apply --check file.patch
git apply file.patch
```

Pull Requests

The **git pull** command downloads the latest version history of a file from the remote repository to the local repository, and this command also performs a merge operation and updates the content of your working directory. See the "**pull**" section of Chapter 15.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which line of the following JSON file contains an error?

```
{
     "name": "Sue Smith",
     "phone": ["555-555-5555", "555-555-5556"],
     "address": {
             "street": "123 Main St."
              "city": "Anytown",
              "state": "CA"
      }
}
```

- ❍ **A.** Line 2
- ❍ **B.** Line 3
- ❍ **C.** Line 4
- ❍ **D.** Line 5

2. Which of the lines in the following YAML file contains an error?

```
---
name: Sue Smith
phone:
- 555-555-5555
- 555-555-5556
address
 street: 123 Main St.
 city: Anytown
 state: CA
```

- ❍ **A.** Line 4
- ❍ **B.** Line 5
- ❍ **C.** Line 6
- ❍ **D.** Line 7

3. What is the name of the file that Ansible uses to provide a set of instructions to configure a system?
   - ❍ **A.** Manifest
   - ❍ **B.** Playbook
   - ❍ **C.** Cookbook
   - ❍ **D.** Target

4. The **git** _____command applies all the changes from a branch to the current branch.
   - ❍ **A. init**
   - ❍ **B. pull**
   - ❍ **C. merge**
   - ❍ **D. rebase**

Cram Quiz Answers

1. **D.** Line 5 contains an error as it is missing a comma after **123 Main St.**.
2. **C.** Line 4 contains an error as it is missing a colon after **address**.
3. **B.** Manifests are used with Puppet. Cookbooks are used with Chef. Targets are used with SaltStack.
4. **D.** The **git rebase** command applies all the changes from a branch to the current branch. The **git init** command creates a new repository. The **git pull** command downloads the latest version history of a file from the remote repository to the local repository. The **git merge** command combines changes from two versions of a **git** file into a single new version.

CHAPTER 17

Container, Cloud, and Orchestration Concepts

This chapter covers the following Linux+ XK0-005 exam objective:

- **3.5:** Summarize container, cloud, and orchestration concepts.

Recall that you were introduced to the concept of containers in Chapter 14, "Perform Basic Container Operations." You learned what a container is as well as how to perform basic container operations, including starting, stopping, and listing containers. While knowing these operations is very important when managing containers that reside on a single host, if you decided to take containers to the next level, you will soon realize that the next level involves managing containers across multiple hosts.

You were also introduced to utilities that allow for the orchestration of systems in Chapter 16, "Common Infrastructure as Code Technologies." In that chapter, you learned that tools like Ansible, Puppet, Chef, SaltStack, and Terraform can be used to orchestrate containers. In this chapter you will learn about a specialized container orchestration tool called Kubernetes. You will learn about some of the key features of Kubernetes as well as some more advanced container components, such as container persistent storage, container networks, and container registries.

ExamAlert

It is important to realize for the Linux+ XK0-005 exam that the objective that matches this chapter asks you to "Summarize container, cloud, and orchestration concepts" For the exam, you should not be asked any questions on configuration or administration of these technologies. Kubernetes and the utilities that orchestrate containers are very large topics, and there are several certification exams that focus just on these technologies. An individual who is Linux+ certified should understand the concepts behinds these technologies but should not necessarily be an expert. As a result, the coverage of this topic in this chapter is designed to match the awareness of the topic that is required to pass the Linux+ exam.

This chapter provides information on the following topics: Kubernetes benefits and application use cases; single-node, multicontainer use cases; container persistent storage; container networks; service mesh; bootstrapping; and container registries.

Kubernetes Benefits and Application Use Cases

Imagine that you have created a container that performs a critical function in your organization. Say that it provides a function that is related to your website (such as providing access to a key database or storing customer information). The main idea is that this container is critical to your website performing correctly.

Now imagine that one day something happens to that container, and it stops responding to requests from your web server. What do you do in this situation? Should you ever be in a position that you need to do something manually?

One benefit of Kubernetes is that it can provide fault tolerance by recognizing when a container fails and automatically launching a new container. It can also be used to launch additional containers when demand increases and spread the load between containers.

The benefits of Kubernetes include the following:

- It can monitor the health of containers and take action when a container isn't performing or responding.
- It can be configured to run multiple containers concurrently.
- It can work both on premises and in cloud environments.
- It can be used for load balancing between containers.
- Kubernetes can provide horizontal scaling (adding more containers as needed to handle larger system loads).
- It can be used to update containers via a rollout process and even perform rollback operations, if needed.

Pods

Let's return to that scenario in which you have a container that provides some key function to your web server. You have decided to use Kubernetes in order to best ensure that your web server will be able to reach at least one of a group

of containers that provide the necessary functionality. However, you realize that the data that the containers are providing (or storing) can't just be duplicated between different containers as keeping this data completely in sync would be a nightmare.

The solution: Use Kubernetes pods. A *pod* is one or more containers that are connected via shared resources, such as shared storage or network resources. The term used to describe these containers is "tightly coupled."

> **Note**
>
> The term *pod* in Kubernetes is related to the term for a group of whales. In addition, the Docker logo is a whale that looks like a ship with shipping containers on the top of it.

The advantage of a Kubernetes pod is that when resources are shared, such as for storage, each container of the pod is able to access exactly the same resource. As more containers are brought online to handle additional system load, they can also access this shared resource.

Note that a pod does not necessarily have to include multiple containers. It is fairly common in simple Kubernetes solutions to have a pod with a single container. If you decide to have multiple containers running within a pod, then you need to use a sidecar, described next.

Sidecars

A *sidecar* is a container in Kubernetes that is used to make changes to the shared storage of a pod and perform other functions for the pod. For example, a sidecar could handle authentication to the pod containers, and it could also handle logging of activities in the pod.

Note that one of the primary uses of sidecars is to create a service mesh. See the "Service Mesh" section, later in this chapter, for more information.

Ambassador Containers

A type of sidecar container called an Ambassador container is designed to act as a client proxy to the containers in a pod. Suppose that the containers in your pod need to access another resource, such as a remote database. While you could configure the container to perform this connection, the Ambassador container could act as a proxy for the containers in the pod to gain access to the remote database.

The advantage of using an Ambassador container is that containers are meant to be single-purpose software components. Remember that containers aren't full virtual machines; a container should ideally be designed to perform a single task very well. As a result, having containers perform their primary task and also perform client connections breaks the principle of single-task containers.

Single-Node, Multicontainer Use Cases

See the "Pods" section, earlier in this chapter.

Compose

Docker Compose is a tool that is very similar to Kubernetes in that both Docker Compose and Kubernetes are designed to run collections of containers. The big difference between the two is that Docker Compose is designed to run containers on only a single host system, while Kubernetes can run containers either on a single host or across multiple hosts.

Container Persistent Storage

A persistent volume (PV) is storage space that can be used by an OS. This PV "points to" a physical storage device (called a "block" device), which can be a traditional SATA hard disk, a solid-state drive (SSD), or other storage media.

The reason this system is considered "persistent" is that the underlying physical storage device could change, but the OS would not be aware of or impacted by this change. The OS would always use the PV, but it could just be mapped to a different block device than it originally was.

The advantage of this system is that older block devices can be seamlessly upgraded without any impact on the operation of the OS. A block storage device is a physical storage device and is typically the back-end storage for cloud-based storage systems. Examples include the following:

- Traditional SATA drives
- SSDs
- RAID drives

- Storage area networks (SANs), which can include Fibre Channel Protocol (FCP), Internet Small Computer System Interface (iSCSI), ATA over Ethernet (AoE), and Fibre Channel over Ethernet (FCoE)

There are additional block storage devices, such as optical discs, but most of them are not suitable for VM or cloud solutions.

Container Networks

This section focuses on networking components you should consider implementing when working with cloud technologies.

Overlay Networks

An overlay network is a network that is built on top of another network. This might be necessary in situations where two different network protocols are used. For example, in an optical network that uses optical protocols, there might be a need for IP (Internet Protocol)-based communications. Because the optical protocols don't support IP, an overlay network is used.

With an overlay network, network packets are encapsulated within other network packets. So, the data being transported on the optical network could include encapsulated IP packets.

Many overlay network technologies are commonly used on the Internet, including peer-to-peer protocols, Tor, VPN, and VoIP.

Bridging

In cloud computing, it is common to end up with multiple network segments. One way of establishing a connection between these different segments is by using a router that forwards network traffic between the different network segments.

Bridging is an alternative that provides a different solution than routing. A bridge device merges different network segments into a single segment. There are different types of bridges:

- **Simple:** This type of bridge connects two network segments.
- **Multiport:** This type of bridge connects multiple networks.

- **Transparent:** Also known as a learning bridge, this type of bridge builds routing tables dynamically.
- **Source route:** The source of the traffic builds the routing.

Network bridges offer several advantages. They can be used to merge dissimilar network transmissions (think of a physical network merged with a wireless network to form a single network). They can also be used to maximize bandwidth.

Network Address Translation (NAT)

Network address translation (NAT) is a technology in which the network packets from a private IP network are translated so they can be routed to a public network. There are a few different forms of NAT:

- **DNAT (destination NAT):** DNAT is used when you want to place servers behind a firewall and still provide access from an external network. DNAT rules are placed on the **PREROUTING** filtering point. Further discussion of this topic is beyond the scope of this book.
- **SNAT (source NAT):** SNAT is used when you have an internal network with *statically assigned* private IP addresses. Using SNAT, you can funnel access to the Internet via a single machine that has a live IP address (that is, an address that is routable on the Internet). This system is configured with SNAT, which is used to map an internal address with external communication. SNAT rules are placed on the **POSTROUTING** filtering point. Further discussion of this topic is beyond the scope of this book.
- **MASQUERADE:** This type of NAT is used with an internal network that has *dynamically assigned* private IP addresses (for example, using DHCP). Using **MASQUERADE**, you can funnel access to the Internet via a single machine that has a live IP address (an address that is routable on the Internet). This system is configured with **MASQUERADE**, which is used to map an internal address with external communication. **MASQUERADE** rules are placed on the **POSTROUTING** filtering point.

ExamAlert

The most likely question regarding NAT that you will see on the Linux+ XK0-005 exam relates to the different types of NAT. Make sure you are familiar with the three types of NAT described here before taking the exam.

Host

In some cases, you might want to permit networking only between a container and the local host. This might be because incoming network requests are sent to the host operating system and then forwarded to the container.

Such a solution is referred to as a *host networking solution*. With such a solution, the software that manages the container creates a virtual network interface on the local host and the container. Network configuration settings are applied to these virtual network interfaces.

Service Mesh

As mentioned previously, one of the defining features of a container is that it is designed to perform a single task very well. This concept allows container developers to focus on a single feature, but often that single feature, by itself, doesn't serve a purpose. As a result, you will often see a collection of containers or pods, each performing a specific task, deployed as a solution. In such a scenario, these containers or pods are typically referred to as *microservices*.

Microservices often need to communicate with one another. This can pose a challenge as each microservice container or pod will not have a full understanding of the complete solution. A *service mesh* can be used to manage communication between microservices.

A service mesh provides the means for communication between microservices, and it also can be used to handle errors in communication. A service mesh also makes for a single point of administration and visibility of communication between microservices.

Note that a service mesh is established by using sidecars. See the "Sidecars" section, earlier in this chapter, for more information about sidecars.

Bootstrapping

In general, bootstrapping refers to any process that self-starts without the need for external assistance. For example, all operating systems use bootstrapping to boot the system. When you power on a system, it starts executing code that comes from either the BIOS or UEFI. That code finds and starts the bootloader software (GRUB or GRUB2 on Linux systems), and the bootloader in turn starts the kernel. The kernel is responsible for starting the rest of the operating system.

However, bootstrapping isn't just a term for starting operating systems. The term is also applied to automating the process of installing an OS. This is a critical component of cloud and virtualization technologies, and this section focuses on a bootstrapping tool used for these technologies.

Cloud-init

As its name implies, cloud-init is a tool that can be configured to provide the initialization of a cloud instance. This tool makes use of preprovisioned images, and it is flexible enough to use with different solutions. Typical operations you might conduct with cloud-init are to add SSH keys, perform networking configurations, execute scripts that will run after the installation, and perform other similar tasks.

Container Registries

A *container registry*, or *container repository*, is a central location where container images are stored. Examples of popular container registries include Docker Hub, GitHub Package Registry, Google Artifact Registry (GAR), and Amazon Elastic Container Registry.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. _____ are containers that are connected via shared resources, such as shared storage or network resources.

   - ❍ **A.** Pods
   - ❍ **B.** Herds
   - ❍ **C.** Colonies
   - ❍ **D.** Swarms

2. A service mesh is implemented using _____ containers.
   - ❍ **A.** wheelchair
   - ❍ **B.** carriage
   - ❍ **C.** helper
   - ❍ **D.** sidecar

3. An _____ network is one that is built on top of another network.
   - ❍ **A.** overload
   - ❍ **B.** overlay
   - ❍ **C.** overlook
   - ❍ **D.** overpass

4. Which type of bridge builds routing tables dynamically?
   - ❍ **A.** Source route
   - ❍ **B.** Multiport
   - ❍ **C.** Transparent
   - ❍ **D.** Simple

Cram Quiz Answers

1. **A.** All of these answers describe different collections of animals, but only pod matches the definition.
2. **D.** One of the primary uses of sidecars is to create a service mesh. The other answers are not valid container types in Kubernetes.
3. **B.** An overlay network is a network that is built on top of another network. The other answers are not valid terms.
4. **C.** A transparent bridge builds routing tables dynamically. A simple bridge connects two network segments. A multiport bridge connects multiple networks. A source route bridge is built by the source of the traffic.

This page intentionally left blank

CHAPTER 18

Analyze and Troubleshoot Storage Issues

This chapter covers the following Linux+ XK0-005 exam objective:

- **4.1:** Given a scenario, analyze and troubleshoot storage issues.

In this chapter you will learn how to deal with storage-related issues. You will learn about typical storage problems, such as latency and throughput issues, as well as input/output operations per second (IOPS) and capacity issues. In this chapter you will also explore some filesystem issues as well as issues that are inherent in specific storage types. Finally, you will learn about mount option problems.

This chapter provides information on the following topics: high latency, IOPS scenarios, capacity issues, filesystem issues, I/O scheduler, device issues, and mount option problems.

High Latency

Storage latency is a measurement of how long it takes for data to travel from a storage device to the operating system. Large latency values have a major impact on the performance of services and on the user experience. In this section you will learn how to monitor and manage high latency issues.

Input/Output (I/O) Wait

The **ioping** command can be used to perform simple latency tests on a disk. *Latency* is a measure of the delay in data transfer. Typically you want to perform multiple tests on a device, such as in the following

example, which ends up performing 100 tests on the filesystem for the current directory (though the **tail** command is used here to limit the output):

```
[root@OCS ~]# ioping -c 100 . | tail
4 KiB <<< . (ext4 /dev/sda1): request=96 time=210.6 us
4 KiB <<< . (ext4 /dev/sda1): request=97 time=280.7 us
4 KiB <<< . (ext4 /dev/sda1): request=98 time=249.2 us
4 KiB <<< . (ext4 /dev/sda1): request=99 time=230.1 us
4 KiB <<< . (ext4 /dev/sda1): request=100 time=210.9 us

--- . (ext4 /dev/sda1) ioping statistics ---
99 requests completed in 22.6 ms, 396 KiB read, 4.38 k iops,
 17.1 MiB/s
generated 100 requests in 1.65 min, 400 KiB, 1 iops, 4.04 KiB/s
min/avg/max/mdev = 143.1 us / 228.1 us / 369.1 us / 40.4 us
```

Input/Output Operations per Second (IOPS) Scenarios

The primary speed calculation of a storage devices is the *IOPS*, which stands for *input/output operations per second*. You need to take this value into consideration when choosing which underlying storage type you want to use for a storage resource. As you might expect, the IOPS for HDD devices is less than for SDD devices.

This section describes how you can view the IOPS of a storage device to determine if you need to take action to resolve a low IOPS issue.

Low IOPS

The **iostat** command provides input/output statistics on devices, including storage IOPS. When executed with the **-d** option, it provides a variety of information, as shown in this example:

```
#iostat -d egrep "Device|sd"
Device tps   kB_read/s  kB_wrtn/s kB_read kB_wrtn
sda    1.62   17.05     12.08     969749  686696
sdb    0.02   3.76       0.00     213643  88
sdc    0.01   0.11       3.59     6239    203860
```

The values in this output are as follows:

- **Device:** The filename of the storage device
- **tps:** Transactions per second
- **kB_read/s:** Kilobytes of data read from the device per second
- **kB_wrtn/s:** Kilobytes of data written to the device per second
- **kB_read:** Total kilobytes of data read from the device
- **kB_wrtn:** Total kilobytes of data written to the device

This data can be useful when trying to determine if the read/write actions on a device cause a load that is too great. You need to monitor this information over time, as it is difficult to determine if there are issues by reading this output once. Output must be compared to data from other points in time to determine if there is an issue.

Capacity Issues

Capacity issues for storage devices can have several negative impacts. If a storage device lacks enough free space, software installation could fail. Users might not be able to save their files if there isn't enough space or if there aren't enough inodes left on a storage device. This section demonstrates how you can determine if you have a storage capacity issue.

Low Disk Space

The **df** command displays usage of partitions and logical devices:

```
# df
Filesystem   1K-blocks   Used     Available Use% Mounted on
udev          2019204    12       2019192     1% /dev
tmpfs          404832    412       404420      1% /run
/dev/sda1    41251136    6992272  32522952   18% /
none         4           0         4          0% /sys/fs/cgroup
none         5120        0        5120        0% /run/lock
none         2024144     0        2024144     0% /run/shm
none         102400      0         102400     0% /run/user
```

Table 18.1 details important options for the **df** command.

TABLE 18.1 **df Command Options**

Option	Description
-h	Displays values in human-readable size.
-i	Displays inode information.

The **du** command shows the estimated amount of disk space usage in a directory structure. For example, the following command displays the amount of space used in the **/usr/lib** directory:

```
# du -sh /usr/lib
791M /usr/lib
```

Table 18.2 details important options for the **du** command.

TABLE 18.2 **du Command Options**

Option	Description
-h	Displays values in a human-readable size. (Instead of always displaying in bytes, it displays in more understandable values, such as megabytes or kilobytes, depending on the overall size of the file.)
-s	Displays a summary rather than the size of each subdirectory.

ExamAlert

For the Linux+ XK0-005 exam, know when to use **df** (for partition-based issues) and when to use **du** (for directory-based issues).

Inode Exhaustion

A file or directory consists of several components. Many of these components, such as the owner and permissions, are stored in a filesystem element called an inode.

Everything about a file besides the data in the file and the filename is stored in the inode. Each file is given an inode number that is unique for the filesystem in which the file resides.

The inode of a file contains the following information:

- Unique inode number
- User owner
- Group owner
- Mode (permissions and file type)
- File size
- Timestamps:
  - Last time the file contents were modified
  - Last time the inode data was modified
  - Last time the file was accessed
- Pointers (references to the data block locations that contain the file data)

You can see inode information by using the **stat** command:

```
[root@OCS ~]$ stat /etc/passwd
 File: '/etc/passwd'
 Size: 2597 Blocks: 8 IO Block: 4096 regular file
Device: fc01h/64513d Inode: 33857 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid:
 ( 0/ root)
Access: 2022-10-12 12:54:01.126401594 -0700
Modify: 2022-09-08 12:53:48.371710687 -0700
Change: 2022-09-08 12:53:48.371710687 -0700
 Birth: -
```

Each file must have a unique inode, which is used to keep track of file metadata. Each filesystem has a limited number of inodes. Normally this limit is very high, and running out of inodes is rare, but a filesystem with many small files might pose a problem. The **--inodes** option to the **df** command is useful in troubleshooting this problem as it displays the total number of inodes, how many are used, how many inodes are available, and a percentage of inodes that are currently in use:

```
[student@OCS ~]$ df --inodes | grep /tmp
Tmpfs              505992 602 505390 1% /tmp
```

ExamAlert

Understanding what inodes are used for helps you better understand several Linux features, including hard links (see Chapter 2, "Manage Files and Directories") and quotas (see Chapter 21, "Analyze and Troubleshoot User Access and File Permissions"). Make sure you have a good understanding of this topic as you might see multiple exam questions related to inodes.

Filesystem Issues

A filesystem issue is a problem with either the filesystem itself or with how the filesystem is being mounted. This section discusses common filesystem issues that you should be aware of for the Linux+ XK0-005 exam.

Corruption

When changes are made to a filesystem, some of these changes are made in RAM initially and then synced to the filesystem at regular intervals. This process makes filesystems faster, but it can also result in filesystem corruption.

Corruption occurs when a filesystem isn't properly unmounted, such as during a power loss or other system interruptions. The data about filesystem changes that is stored in RAM is not synced to the storage device, resulting in a corrupted filesystem.

The **fsck** utility (which you need to run as the root user) is designed to find filesystem problems on unmounted filesystems. Consider this example:

```
# fsck /dev/sdb1
fsck from util-linux 2.20.1
e2fsck 1.42.9 (4-Feb-2014)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/sdb1: 11/12544 files (0.0% non-contiguous), 6498/50176 blocks
```

This utility is fairly straightforward. It calls the correct filesystem check utility based on a probe of the filesystem and then prompts the user when errors are

found. To fix an error, answer **y** or **yes** to the prompts. Because **yes** is almost always the appropriate answer, the **fsck** utility supports a **-y** option, which automatically answers **yes** to each prompt.

The **fsck** command executes filesystem-specific utilities. In the case of ext2/ext3/ext4 filesystems, the **fsck** command executes the **e2fsck** utility.

> **ExamAlert**
>
> Knowing how to fix filesystem issues with the **fsck** command is important, but knowing what causes filesystem corruption is also important for the Linux+ XK0-005 exam.

Mismatch

In Chapter 3, "Configure and Manage Storage Using the Appropriate Tools," you learned about the **/etc/fstab** file, which is designed to mount filesystems automatically at boot. Recall that each line in this file describes one mount process. The following is an example of one of these lines:

```
/dev/sda1 / ext4 defaults 1 1
```

One common problem that might occur when manually creating or modifying entries in the **/etc/fstab** file is providing the wrong filesystem type. In the previous example, **ext4** represents the filesystem type. If the wrong filesystem is provided, a "mismatch filesystem" error occurs, and the **/etc/fstab** file needs to be modified.

I/O Scheduler

Kernel parameters can be used to optimize the I/O (input/output) scheduler. Several parameters can be set to change the behavior of the scheduler. This section covers the parameters that are important for the Linux+ XK0-005 exam.

To see the current scheduler, view the contents of the **/sys/block/**<*device*>**/queue/scheduler** file (where <*device*> is the device name). Here's an example:

```
[root@OCS ~]# cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
```

The value within the square brackets is the default. To change this value, use the **echo** command, as shown here:

```
[root@OCS ~]# echo "cfq" > /sys/block/sda/queue/scheduler
[root@OCS ~]# cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
```

There are several types of schedule, including the following:

- **cfq:** The Completely Fair Queuing schedule has a separate queue for each process, and each queue is served in a continuous loop.
- **noop:** This schedule follows the first in, first out (FIFO) principle.
- **deadline:** This is the standard scheduler. This scheduler creates two queues—a read queue and a write queue. It also puts a timestamp on each I/O request to ensure that requests are handled in a timely manner.

ExamAlert

Know the three different types of schedules and be ready for questions on the Linux+ XK0-005 exam that test your understanding of why you should use one over the others.

Device Issues

Storage devices may pose issues that need to be diagnosed. This section focuses on some common device issues.

Non-volatile Memory Express (NVMe)

When SSD devices first appeared on the market, they were faster than spinning disks, but they were connected to the motherboard and used older protocols (SAS and SATA) to transfer data. Initially this wasn't a problem, but the faster the SSD devices became, the more often the older protocols created bottlenecks. There was a need for a faster transfer protocol, and the solution was NVMe.

Why is NVMe faster than the older protocols? In a nutshell, it provides more "lanes" of I/O, which allows for higher overall throughput.

To troubleshoot NVMe on SSD drives, use the **nvme** command. For example, to list devices that have NVMe enabled, use the **nvme list** command. Table 18.3 lists additional **nvme** command options.

TABLE 18.3 **nvme Command Options**

Option	Description
error-log <*device_name*>	Provides information about problems for the specified NVMe-enabled SSD device.
smart-log <*device_name*>	Displays the SMART log for the specified NVMe-enabled SSD device. (SMART is a health assessment tool that can determine problems with storage devices.)

Solid-State Drive (SSD)

Over time, storage devices degrade and fail (sometimes suddenly). To determine if there are any problems with a drive before it fails, consider taking the following proactive measures:

- **Search for bad blocks:** An increase in the number of bad blocks on a device can indicate degraded storage. To test for bad blocks, use the **badblocks** command:

```
root@OCS:~# badblocks -v /dev/sdb1
Checking blocks 0 to 1047551
Checking for bad blocks (read-only test): done


Pass completed, 0 bad blocks found. (0/0/0 errors)
```

- **Use the SMART tool:** Self-Monitoring, Analysis, and Reporting Technology (SMART) is a tool available on hard disks and SSDs that monitors the health of a drive. Not all disks support this utility, but most modern ones do. The **smartctl** command is used to display information about the hard disk. For example, to perform a health check, use the **smartctl -HH <***device_name***>** command.
- **Disk quotas:** If quotas are enabled on a filesystem, users may be limited in their ability to create new files. To determine whether a quota is the cause of a problem, consider using the following commands:
  - System administrators should use the **repquota** command to display quotas for all users on a filesystem.
  - Regular users should use the **quota** command to display quota limitations for their own accounts.

See Chapter 21 for more information about quotas.

SSD Trim

On SSD devices, only data cells that are completely empty can be used to store new data. Most filesystems don't actually delete data when a file is deleted; rather, they just remove the inode pointers to the file. This behavior can have a negative performance impact on SSD storage devices as data might need to be deleted before a file's data can be saved.

SSD trim is a tool that can be used to clean up data cells that contain data but that are no longer used by the filesystem. There are a couple of ways of using this tool:

- By executing the **fstrim** command
- By adding the **discard** mount option to the **/etc/ftab** command

RAID

Software RAID devices created in Linux are typically very robust. There are a few scenarios, however, in which a RAID device can be compromised:

- The physical drive that the RAID device is based on is removed.
- A corruption occurs in the partition table.
- A partition that the RAID device is based on is deleted.

See Chapter 3 for a complete discussion of RAID. When troubleshooting, you primarily use the **mdadm** command and the **/proc/mdstat** command.

LVM

See Chapter 3 for a complete discussion of Logical Volume Manager (LVM).

I/O Errors

Bad blocks can result in I/O errors. To test for bad blocks, use the **badblocks** command:

```
root@OCS:~# badblocks -v /dev/sdb1
Checking blocks 0 to 1047551
Checking for bad blocks (read-only test): done


Pass completed, 0 bad blocks found. (0/0/0 errors)
```

Mount Option Problems

When you're mounting a partition or volume, the *mount point* (or *mount directory*) must already exist. If it doesn't, the following error will appear (with **/data** replaced by your mount point):

```
mount: mount point /data does not exist
```

To resolve this error, create the mount point:

```
root@OCS:~# mkdir /data
```

Recall the format of the **/etc/fstab** file:

```
/dev/sda1 / ext4 defaults 1 1
```

Additional possible mount option problems include the following:

- The wrong device name (**/etc/sda1** in the example above). Use the **df** command (covered in the "Low Disk Space" section, earlier in this chapter) to double-check the device name.
- Mismatched filesystem type (see the "Mismatch" section, earlier in this chapter).
- Incorrect mount options, which are often the result of misspellings (**default** instead of **defaults**, for example) or extra spaces with multiple mount options (such as **rw, sync, nouser** instead of **rw,sync,nouser**).

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which option to the **ioping** command performs 100 I/O latency tests?
   - ❍ **A. -c 100**
   - ❍ **B. -s 100**
   - ❍ **C. -t 100**
   - ❍ **D. -T 100**

2. Which command can be used to determine a storage device's IOPS?
   - ❍ **A. ioping**
   - ❍ **B. df**
   - ❍ **C. iostat**
   - ❍ **D. du**

3. Which of the following is not stored in an inode entry?
   - ❍ **A.** File size
   - ❍ **B.** User owner
   - ❍ **C.** Filename
   - ❍ **D.** File mode

4. Which I/O scheduler follows the FIFO (first in, first out) principle?
   - ❍ **A. fifo-sched**
   - ❍ **B. deadline**
   - ❍ **C. cfq**
   - ❍ **D. noop**

Cram Quiz Answers

1. **A.** The **-c** option specifies the number of tests. The **-s** option specifies the size of the tests. **-t** and **-T** specify the minimum and maximum request times, respectively.
2. **C.** The **iostat** command display IOPS information. The **ioping** command is used to view latency information. The **du** and **df** commands are used to display disk and directory space utilization.
3. **C.** The filename is not stored in the inode. The other answers are values stored in the inode.
4. **D.** The **noop** schedule follows the FIFO (first in, first out) principle. The Completely Fair Queuing (**cfq**) schedule has a separate queue for each process, and the queues are served in a continuous loop. The deadline scheduler creates two queues: a read queue and a write queue. **fifo-sched** is not a valid schedule.

CHAPTER 19

Analyze and Troubleshoot Network Resource Issues

This chapter covers the following Linux+ XK0-005 exam objective:

- **4.2:** Given a scenario, analyze and troubleshoot network resource issues.

Regardless of which operating systems you have worked on, network-related issues are something that you have probably already experienced. This chapter focuses on the tools and configurations that you should be aware of when dealing with network resource issues.

You will first explore the common network configuration issues that you are likely to encounter. You will also learn about tools that help you handle interface errors and bandwidth issues. The chapter concludes by discussing name resolution issues and testing of remote systems.

This chapter provides information on the following topics: network configuration issues, firewall issues, interface errors, bandwidth limitations, name resolution issues, and testing of remote systems.

Network Configuration Issues

As you may suspect, a misconfigured network leads to issues connecting to the network. This section highlights the most commonly misconfigured network settings.

Subnet

An invalid subnet setting causes a system to not be able to communicate with other devices in the local network. It also causes routing to fail as routing relies on communicating with the router on the local subnet.

Recall from Chapter 5, "Use the Appropriate Networking Tools or Configuration Files," that one of the tools that can be used to display and configure a subnet is the **ip** command. For example, the following output displays the subnet, in CIDR notation, highlighted in gray:

```
[root@OCS ~]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
     inet 127.0.0.1/8 scope host lo
     valid_lft forever preferred_lft forever
     inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
     link/ether 08:00:27:b0:dd:dc brd ff:ff:ff:ff:ff:ff
     inet 192.168.1.26/24 brd 192.168.1.255 scope global dynamic
enp0s3
     valid_lft 2384sec preferred_lft 2384sec
     inet 192.168.1.24/16 brd 192.168.255.255 scope global enp0s3
     valid_lft forever preferred_lft forever
     inet6 fe80::a00:27ff:feb0:dddc/64 scope link
     valid_lft forever preferred_lft forever
```

Routing

Routing is the process of sending network data from one Internet Protocol (IP) network to another via a gateway. A *gateway* is a system that has multiple network interfaces and transfers data between the networks.

Use the **route**, **ip**, or **netstat** command to determine routing on a host. For example, the **route** command displays the routing table:

```
[root@OCS ~]# route
Kernel IP routing table
Destination Gateway     Genmask      Flags Metric Ref Use Iface
```

```
default     192.168.1.1  0.0.0.0     UG    100      0   0  enp0s3
192.168.0.0 0.0.0.0      255.255.0.0  U     100      0   0  enp0s3
192.168.1.0 0.0.0.0      255.255.0.0  U     100      0   0  enp0s3
```

The most common troubleshooting scenario in which you would want to use the **route** command is when you are able to connect to systems within the local network but can't access systems outside the local network. This is normally the result of a misconfigured router, which can be determined by using the **route** command.

These commands are also covered in Chapter 5.

Firewall Issues

There are many possible issues that can affect network access when using firewalls. For example, an incorrect rule might be applied to an interface or rules might be placed in the wrong order in the firewall list.

Firewall issues are normally placed into one of two categories:

- Access is granted when it shouldn't be granted.
- Access is denied when it should be granted.

For example, imagine a scenario in which you have a private website that shouldn't be accessible outside your local network. You have firewall rules that are designed to block access from outside the local network but discover that the website is still accessible from remote systems.

ExamAlert

To prepare for this Linux+ XK0-005 exam topic, make sure you understand firewall rules and how firewalls function. See Chapter 10, "Implement and Configure Firewalls," for the essentials that you should know about firewalls for the exam.

Interface Errors

The network interface might itself be a cause of networking issues. There might be a problem with a network device, problems with the physical network connection, issues with network cabling, or traffic issues on the network

(for example, too much traffic for the network bandwidth). This section covers some of the essentials of interface errors.

Dropped Packets

A packet drop occurs when a remote system doesn't respond to an inbound network packet. It could be the result of firewall rules, saturation, or a misconfigured network.

To view dropped packet information, consider using the **netstat** command. For example, Figure 19.1 shows information about each network interface, with the receiving dropped packets (RX-DRP) and transmitting dropped packets (TX-DRP) highlighted.

```
root@ocs ~]$ netstat -i
Kernel Interface table
Iface       MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eno16780   1500 761804883      0    1256 0      122658847      0      0      0 BMRU
lo        65536 592413709      0       0 0      592413709      0      0      0 LRU
```

FIGURE 19.1 **The netstat Command Displaying Dropped Packets**

The **netstat** command is also covered in Chapter 5.

Collisions

On a busy network, two or more network devices sometimes try to send network packets at the same time. This results in collisions, which are not normally much of a problem. A collision makes a packet "bound back" to the original network device, which then tries to send the packet again after waiting a small, random period of time.

A useful command for displaying collisions is the **netstat** command. For example, Figure 19.2 demonstrates the use of the **netstat -ie** command to display network information, including the highlighted area where collisions are displayed.

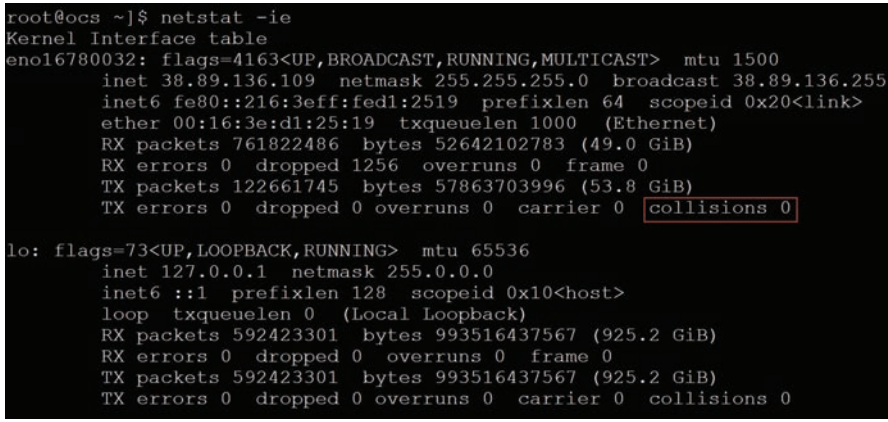

FIGURE 19.2 **The netstat Command Displaying Collisions**

Notice in this figure that the collisions value is 0. This is very commonly a low number. If the number on your system is high, consider increasing the bandwidth or segmenting your network further.

ExamAlert

Collisions are a big red flag that there is not enough bandwidth.

Link Status

Link status is really a question of whether the system is connected to the network. One consideration that isn't Linux specific is simply checking to see if there is a physical connection to the network. Most network interfaces have a light that turns on when a physical connection is established.

You also should verify that an interface is up, which you can do by using the **ip** command. The following example demonstrates that the primary network interface is up (note the **UP** highlighted in gray):

```
[root@OCS ~]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
     inet 127.0.0.1/8 scope host lo
     valid_lft forever preferred_lft forever
     inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
```

```
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
     link/ether 08:00:27:b0:dd:dc brd ff:ff:ff:ff:ff:ff
     inet 192.168.1.26/24 brd 192.168.1.255 scope global dynamic
enp0s3
     valid_lft 2384sec preferred_lft 2384sec
     inet 192.168.1.24/16 brd 192.168.255.255 scope global enp0s3
     valid_lft forever preferred_lft forever
     inet6 fe80::a00:27ff:feb0:dddc/64 scope link
     valid_lft forever preferred_lft forever
```

Another useful tool for determining the link status of a network device is **ethtool**. The **ethtool** command is used to display and configure network device settings, such as the transmission speed and duplex value. Typically these settings are automatically configured through a process called *auto-negotiation*. With auto-negotiation, two network devices determine the best speed and duplex value and use that value automatically; however, these settings can also be set manually. The **ethtool** command also displays or modifies other useful network device settings.

Syntax:

```
ethtool [option] device
```

The following example shows how to get a summary of information with this command:

```
[root@OCS ~]# ethtool eth0
Settings for eth0:
         Supported ports: [ TP ]
         Supported link modes:     10baseT/Half 10baseT/Full
                                   100baseT/Half 100baseT/Full
                                   1000baseT/Full
         Supported pause frame use: No
         Supports auto-negotiation: Yes
         Advertised link modes: 10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
         Advertised pause frame use: No
         Advertised auto-negotiation: Yes
         Speed: 1000Mb/s
```

```
Duplex: Full
Port: Twisted Pair
PHYAD: 0
Transceiver: internal
Auto-negotiation: on
MDI-X: off (auto)
Supports Wake-on: umbg
Wake-on: d
Current message level: 0x00000007 (7)
                           drv probe link
Link detected: yes
```

The **ethtool** command provides a lot of output. Table 19.1 describes some of the most useful output from this command.

TABLE 19.1 **Useful Information from the ethtool Command Output**

Output	Description
Supported link modes	This output shows the speed and duplex values that this device is able to use.
Supports auto-negotiation	If this device and the connected-to device both support auto-negotiation, the speed and duplex settings don't need to be manually set.
Advertised link modes	This output shows the speed and duplex values that this device is telling other devices it can use.
Speed	This output shows the current speed setting the device is using.
Duplex	This output shows the duplex value the device is currently using.
Link detected	If this item is set to "yes," the device is currently connected via the network to another device.

Table 19.2 describes some useful options for the **ethtool** command.

TABLE 19.2 **ethtool Command Options**

Option	Description
-i	Provides driver information for this device.
-S	Displays statistics for this device.
-p	Identifies this device by causing its link light to blink. This is very useful when a system has multiple network interface cards.

Option	Description
speed *value*	Allows you to manually set the speed value of the device, where *value* is the speed value to set the device to.
duplex *value*	Allows you to manually set the duplex value of the device; *value* should be **full** or **half**.
autoneg *value*	Enables auto-negotiation; *value* should be either **yes** or **no**.

For wireless devices, consider using the **iwconfig** command to display or set information about wireless network interfaces.

Syntax:

```
iwconfig [interface] [parameters]
```

If no arguments are provided, this command displays all network interfaces:

```
[root@OCS ~]# iwconfig
lo no wireless extensions.
eth0 no wireless extensions.
wlan0 IEEE 802.11bgn ESSID:"test1"
       Mode:Managed Frequency:2.412 GHz Access Point:
         Not-Associated
       Tx-Power=20 dBm
       Retry min limit:7 RTS thr:off Fragment thr=2352 B
       Power Management:off
       Link Quality:0 Signal level:0 Noise level:0
       Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
       Tx excessive retries:0 Invalid misc:0 Missed beacon:0
```

Parameters can be used to configure the wireless interface. For example, to change the device mode to ad hoc, execute the following command:

```
[root@OCS ~]# iwconfig wlan0 mode Ad-Hoc
```

Table 19.3 lists some useful parameters for the **iwconfig** command.

TABLE 19.3 **iwconfig Command Parameters**

Parameter	Description
essid *value*	Sets the ESSID name (that is, the network name).
nwid *value*	Sets the network ID.

Parameter	Description
mode *value*	Sets the mode (*value* can be **Ad-Hoc**, **Managed**, **Master**, and so on).
ap *value*	Forces the network card to use the specified access point.

Bandwidth Limitations

Bandwidth is the maximum amount of data that can travel through media. The media could be network cables, wireless, or even the network interface itself.

Throughput is the actual amount of data that passes through media. This value is limited by many factors, including bandwidth, latency, and interference.

The **iperf** command provides a means to create tests of the throughput between two systems. It requires installation and configuration of the tool on two systems: a client and a server.

ExamAlert

Complete coverage of the **iperf** command is beyond the scope of the Linux+ XK0-005 exam. You should be aware of the purpose of the command and the fact that it requires both client and server setup.

Saturation occurs when throughput often (or constantly) reaches the value of the bandwidth. Saturation is likely in just about every network from time to time, but when it becomes a regular occurrence, the network appears to be slow or sluggish. This increases problems with latency and makes users frustrated. Use network monitoring tools like Wireshark (covered in Chapter 5) to determine if saturation is occurring too often.

High Latency

Think of *latency* as any sort of delay in communication (for example, the time it takes for a network packet to travel from one host to another host). Sometimes latency is measured by the roundtrip time of communication (that is, how long it takes to send a network packet and receive a response). Use tools like the **ping** and **traceroute** commands (covered in Chapter 5) to determine if there are latency issues.

Name Resolution Issues

If you can connect to a system by using its IP address but you can't connect by using its hostname, then you might have an issue with name resolution. This section discusses some of the steps you can take to resolve DNS issues.

Domain Name System (DNS)

DNS and name resolution are covered in Chapter 5. This section does not cover all of these tools and configurations again but instead focuses on handling DNS and name resolution errors. If you don't recall how these tools work, consider reviewing the "Name Resolution" section in Chapter 5 before reading further in this chapter.

When troubleshooting name resolution problems, recall the difference between these commands:

- **dig:** This command is useful for performing DNS queries on specific DNS servers.
- **nslookup:** This command is designed to perform simple queries on DNS servers; however, it generally has fewer options than the **dig** command has.
- **host:** This command is normally used to perform simple hostname-to-IP-address translation operations. By default, it uses the configuration information of the local system rather than querying a DNS server that you specify with the command (although you can specify a server as an argument).
- **resolvectl:** This command can also perform DNS lookups. It is slightly different from the **host** command as it only does DNS lookups, but it still uses the local system configurations to determine which DNS servers should perform the queries.

As a rule of thumb, you should use the **dig** or **nslookup** command to test a specific DNS server. Use the **host** or **resolvectl** command to test the DNS client configuration. If you can perform a query successfully using the **dig** or **nslookup** command but a query on the same remote host fails when using the **host** or **resolvectl** command, the problem is likely your configuration information on the client system.

Also, remember where to look to verify configuration information:

- The **/etc/nsswitch.conf** file contains information on where the local client should look to perform name resolutions (local **/etc/hosts** file, DNS server file, and so on).
- The localhost file (**/etc/hosts**) should only be used for name resolutions of local network devices (that is, systems in the same local network as the client system).
- The **/etc/resolv.conf** file contains DNS server information. Remember that this file can be populated by a DHCP server.

ExamAlert

Understanding the different name resolution files is critical for understanding the name resolution process. Expect to see Linux+ XK0-005 exam questions that test your ability to understand how these files affect name resolution.

Testing Remote Systems

This section demonstrates a couple of commands that can be used to perform testing on remote systems.

nmap

The **nmap** command is used to probe a remote system to determine which network ports are reachable on a system. To use the **nmap** command, provide either the IP address or the hostname of the system that you want to scan. Here is an example:

```
# nmap 192.168.1.1
Starting Nmap 5.51 ( http://nmap.org ) at 2015-10-31 23:22 PDT
Nmap scan report for 192.168.1.1
Host is up (2.9s latency).
Not shown: 987 closed ports
PORT STATE SERVICE
23/tcp open telnet
25/tcp open smtp
53/tcp open domain
80/tcp open http
```

```
110/tcp open pop3
119/tcp open nntp
143/tcp open imap
465/tcp open smtps
563/tcp open snews
587/tcp open submission
993/tcp open imaps
995/tcp open pop3s
5000/tcp open upnp
Nmap done: 1 IP address (1 host up) scanned in 4.89 seconds
```

As mentioned previously, the purpose of the **nmap** command is to determine whether a network port is open. This tool can be used to troubleshoot access to a service. For example, a user might complain that they can't reach a specific website on a server. You can use the **nmap** command to determine if that server is listening on port 80 to determine if that network port is open or not. This helps narrow down the problem because if this port is open from a remote system, then the problem is most likely related to the web server software rather than a network connection issue (such as a firewall rule blocking access).

ExamAlert

Be aware that using the **nmap** utility—or any other network probing utility—on a system that you don't have written authorization to probe can be considered a hacking attempt.

openssl s_client

Secure communication over SSL is an important component of networking. In order to troubleshoot SSL issues, you might need to make use of the **openssl** client utility. For example, to test an SSL connection, you can execute the following command:

```
openssl s_client -connect <hostname>:<port>
```

where <*hostname*> is the SSL server's hostname and <*port*> is the network port that the server is listening on.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which of the following cannot be used to display network route information?

   - ❍ **A. route**
   - ❍ **B. ip**
   - ❍ **C. netstat**
   - ❍ **D. arp**

2. Which command is being used in the following example?

```
[root@OCS ~]# _____ eth0
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:     10baseT/Half 10baseT/Full
                                  100baseT/Half 100baseT/Full
                                  1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes: 10baseT/Half 10baseT/Full
                               100baseT/Half 100baseT/Full
                               1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: off (auto)
        Supports Wake-on: umbg
        Wake-on: d
        Current message level: 0x00000007 (7)
                               drv probe link
        Link detected: yes
```

- ❍ **A. ip**
- ❍ **B. ethtool**
- ❍ **C. netstat**
- ❍ **D. dig**

3. Which command displays wireless network device information?
   - ❍ **A. wlconfig**
   - ❍ **B. iwconfig**
   - ❍ **C. wireconfig**
   - ❍ **D. wiconfig**

4. Which of the following commands is designed to probe for open ports on a remote system?
   - ❍ **A. nmap**
   - ❍ **B. netstat**
   - ❍ **C. netscan**
   - ❍ **D. netmap**

Cram Quiz Answers

1. **D.** The **arp** command does not display route information. **route**, **ip**, and **netstat** are used to display route information.
2. **B.** The **ethtool** command displays the output that is shown.
3. **B.** For wireless devices, the **iwconfig** command is used to display or set information about wireless network interfaces. **wlconfig**, **wireconfig**, and **wiconfig** are not valid commands.
4. **A.** The **nmap** command is used to probe a remote system to determine which network ports are reachable from the local system. The **netstat** command displays network statistics for the network interfaces; it does not probe remote systems. **netscan** and **netmap** are not valid commands.

CHAPTER 20

Analyze and Troubleshoot Central Processing Unit (CPU) and Memory Issues

This chapter covers the following Linux+ XK0-005 exam objective:

- 4.3: Given a scenario, analyze and troubleshoot central processing unit (CPU) and memory issues.

This chapter focuses on different tools and files that help you troubleshoot issues related to the CPU (central processing unit) and memory. You will learn how to display memory and CPU usage information and determine which process might be causing issues.

This chapter provides information on the following topics: runaway processes, zombie processes, high CPU utilization, high load average, high run queues, CPU times, CPU process priorities, memory exhaustion, out of memory (OOM), swapping, and hardware.

Runaway Processes

A *runaway process* is a process that is "running away" with a large amount of CPU or RAM resources. Typically, it is CPU resources that are consumed to the point of making a system very sluggish, but sometimes it might be RAM that a process is overconsuming.

To find runaway processes that are taking too many CPU resources, use the **top** command, which is covered in Chapter 4, "Configure and Use the Appropriate Processes and Services." To find processes that are taking too many RAM resources, use the **ps -augx** command (which also helps find processes that are consuming too much CPU resources)

and look for a high value in the **%MEM** column, as in the first process shown in the following example:

```
[root@ocs ~]$ ps -augx
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME
COMMAND
root          1  0.0  77.9  47576  3536 ?        Ss    2021  27:57 /
usr/lib/system
root          2  0.0  0.0      0     0 ?        S     2021   0:01
[kthreadd]
root          3  0.0  0.0      0     0 ?        S     2021   2:00
[ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S<    2021   0:00
[kworker/0:0H]
root          7  0.0  0.0      0     0 ?        S     2021   2:08
[migration/0]
root          8  0.0  0.0      0     0 ?        S     2021   0:00
[rcu_bh]
root          9  0.0  0.0      0     0 ?        S     2021  43:14
[rcu_sched]
root         10  0.0  0.0      0     0 ?        S     2021   2:01
[migration/1]
root         11  0.0  0.0      0     0 ?        S     2021   1:23
[ksoftirqd/1]
root         13  0.0  0.0      0     0 ?        S<    2021   0:00
[kworker/1:0H]
root         14  0.0  0.0      0     0 ?        S     2021   2:04
[migration/2]
```

Zombie Processes

See Chapter 4 for details regarding zombie processes.

High CPU Utilization

One of the most useful tools for monitoring the CPU is **iostat**. This tool is also used for monitoring disk I/O, but this section focuses on the CPU monitoring features.

When you execute the **iostat** command with the **-c** option, it provides you with statistics regarding CPU utilization since the last time the system was booted, as shown in the following example:

```
[root@OCS ~]# iostat -c
Linux 3.10.0-229.el7.x86_64 (localhost.localdomain)     09/05/2015
_x86_64_(1 CPU)
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.84    0.00    1.18    0.33    0.00   96.65
```

The beginning of the output provides a summary of the system, including the kernel version, the hostname, the date of the report, the kernel type, and how many CPUs the system has. The second part of the output contains CPU statistics; if you have more than one CPU, this will be an average of all of the CPUs. The values provided include:

- **%user:** This value represents the percentage of CPU utilization when applications were running at the user level (that is, processes running **user** as a normal user account).
- **%nice:** Regular users can execute commands using the **nice** command to alter process CPU priority. This value represents the CPU utilization for these processes.
- **%system:** This value represents the percentage of CPU utilization of kernel-based processes.
- **%system%iowait:** This value represents the percentage of CPU utilization when the CPU was waiting for disk I/O operations to complete before performing the next action.
- **%system%steal:** This value only pertains to virtual CPUs. In some cases a virtual CPU must wait for the hypervisor to handle requests from other virtual CPUs. This value indicates the percentage of time waiting for the hypervisor to handle the virtual CPU's request.
- **%system%idle:** This value represents the percentage of time the CPU is not handling any requests.

It is unlikely that you will be asked about specific **iostat** options on the Linux+ XK0-005 exam; however, there are two useful arguments to the command, as shown in this syntax:

```
iostat interval count
```

interval is always the first argument and indicates how many seconds to wait between running **iostat** reports. *count* is how many reports to run.

For example, review the output in the following example, which shows the **iostat** command displaying statistical information once every second for three times during a period when a process peaks in system usage:

```
[root@OCS ~]# iostat -c 1 3
Linux 3.10.0-229.el7.x86_64 (localhost.localdomain)     09/05/2015
_x86_64_(1 CPU)


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           6.30    0.40    0.98    0.42    0.00   91.89


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           5.05    0.00   94.95    0.00    0.00    0.00


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           5.05    0.00   94.95    0.00    0.00    0.00
```

Another command that provides the same statistics as the **iostat** command is the **sar** command. However, the **sar** command displays this information as it occurs over time (typically at 10-minute intervals). For example, look at the output in the following example:

```
[root@OCS ~]# sar | head
Linux 3.10.0-229.el7.x86_64 (localhost.localdomain)     09/06/2015
_x86_64_(1 CPU)


12:00:01 AM     CPU     %user     %nice   %system   %iowait    %steal
%idle
12:10:01 AM     all      0.11      0.00      0.10      0.01      0.00
99.78
12:20:01 AM     all      0.08      0.00      0.07      0.02      0.00
99.84
12:30:01 AM     all      0.10      0.00      0.11      0.01      0.00
99.79
12:40:01 AM     all      0.06      0.00      0.04      0.00      0.00
99.89
12:50:01 AM     all      0.08      0.00      0.04      0.00      0.00
99.88
01:00:02 AM     all      0.10      0.00      0.11      0.01      0.00
99.79
01:10:01 AM     all      0.10      0.00      0.05      0.00      0.00
99.85
```

The **iostat** and **sar** commands are typically used to troubleshoot high CPU utilization by observing these values over time. Typically after the installation of the operating system, these values are low, but as more software and users are added to the system, these values become larger over time. Normally the following values are used to determine where an issue may be present:

- **%user:** Can indicate too many users or users that are using too many resources. Consider moving users to other systems.
- **%system:** Indicates an issue with system software using too many CPU resources. Consider moving some of the software to other systems.
- **%iowait:** Indicates an issue with storage. Consider upgrading the storage on the system.

High Load Average

The **uptime** command shows how long a system has been running. More importantly for system monitoring, it provides a quick snapshot of how many users are on the system and the system load average over the past 1, 5, and 15 minutes:

```
[root@OCS ~]# uptime
16:16:00 up 1 day,  1:05,  6 users,  load average: 0.60, 0.51, 0.25
```

The load average often causes confusion for administrators. This value is designed to describe the CPU load. For a system with a single CPU, a load average of 0.50 would mean the CPU was used for 50% of that time period. A load average of 1.50 means that the CPU was overtasked; process requests were stuck on the queue as the CPU was busy handling other requests. This is not an ideal situation for a system if it happens often.

If you have two CPUs, then you must look at this data differently. A load average of 0.50 would mean the CPUs were used for 25% of that time period. A load average of 1.50 means that the CPUs were in use for 75% of that time period. In other words, the load average as a percentage is calculated by taking the load value, dividing by the number of CPUs, and multiplying by 100.

Consistently high uptime values (CPU usage over 75%) can indicate too many users or too much software on the system. Consider moving users or software to other systems to address this issue.

High Run Queues

When processes are waiting for the CPU in order to perform an operation, the process request is placed into a queue. You can see the current queue by running the **vmstat** command, as highlighted in Figure 20.1.

```
root@ocs ~]$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 2  0 192860 1617300 292080 3337584    0    0     0    17    0    0  1  0 99  0
```

FIGURE 20.1 **Viewing the Process Queue with the vmstat Command**

The value under **r** (which is **2** in the example in Figure 20.1) indicates how many process requests are in the CPU queue. If system performance is sluggish, you can use this command to determine if there are a lot of requests in the queue. If there are, it might be time to upgrade the hardware of the system or consider offloading some of the process to another system.

CPU Times

For information about CPU times and the related issues **steal**, **user**, **system**, **idle**, and **iowait**, see the "High CPU Utilization" section, earlier in this chapter.

CPU Process Priorities

Process priorities are covered in detail in the "Setting Priorities" section of Chapter 4. Please refer to that section for more details.

nice

The **nice** command is covered in detail in the "Setting Priorities" section of Chapter 4. Please refer to that section for more details.

renice

The **renice** command is covered in detail in the "Setting Priorities" section of Chapter 4. Please refer to that section for more details.

Memory Exhaustion

Memory exhaustion happens when the operating system runs out of available (or free) memory (that is, RAM). This section explores some of the key aspects of memory exhaustion.

Free Memory vs. File Cache

See the discussion about the **free** and **vmstat** commands in the "**/proc/meminfo**" section, later in this chapter.

Out of Memory (OOM)

When dealing with Out of Memory issues, you should consider memory leaks and the process killer. These topics are covered in this section.

See the "Process Killer" section, later in this chapter.

Memory Leaks

A *memory leak* occurs when a process is allocated memory but that memory is not freed after it is used. There are some third-party tools available for Linux to search for memory leaks; however, for the Linux+ XK0-005 exam, you just need to be aware of what a memory leak is; you should not need to know how to find and troubleshoot memory leaks. In order to actually troubleshoot a memory leak, you would have to examine the code of the program, and that is beyond the scope of the Linux+ exam.

Process Killer

What happens when a system uses too much memory? The Linux kernel has a feature called the *OOM Killer* (Out of Memory Killer), also known as the *Process Killer*, that kills processes to clear up memory. Without this feature, the system could grind to a halt, and new processes would fail to execute.

The OOM Killer determines which process to kill by assigning a score (called a "badness score") to each process and killing the worst one. Often this is the process that is using the most memory and, very likely, a key process on a server, such as the mail service or the web server.

You can search log files (typically either **/var/log/messages** or **/var/log/kern.log**) to find evidence that the OOM Killer has struck. Error messages look like this:

```
host kernel: Out of Memory: Killed process 1466 (httpd).
```

ExamAlert

There are methods of configuring the OOM Killer (using kernel parameters), but they are beyond the scope of the Linux+ XK0-005 exam. For the exam, you should be aware of the function of the OOM Killer and how you can determine if the OOM Killer has taken action.

Swapping

Swapping is a process that the operating system uses to free up RAM when too much RAM is in use. A *swap space* is an area on a storage device where the system can place data that is normally located in RAM. The data that is swapped is normally data that hasn't been recently used by the operating system but has not yet been freed from RAM.

To see your currently active swap devices, execute the **swapon** command with the **-s** option:

```
[root@OCS ~]# swapon -s
Filename  Type      Size    Used Priority
/dev/dm-1 partition 1048568 27100 -1
```

From the output of the **swapon -s** command, you can see the device name (**/dev/dm-1**) that holds the swap filesystem, the size (in bytes) of the swap filesystem, and how much of this space has been used. The priority indicates which swap filesystem should be used first.

If you have an existing swap device, such as a swap file, you can add it to the currently used swap devices by using the **swapon** command. For example, the following command adds the **/var/swap** file:

```
[root@OCS ~]# swapon /var/swap
[root@OCS ~]# swapon -s
Filename  Type      Size    Used  Priority
/dev/dm-1 partition 1048568 27484 -1
/var/swap file      51192    0    -2
```

To have this swap device enabled each time you boot the system, add a line like the following to the **/etc/fstab** file:

```
/var/swap swap     swap     defaults    0     0
```

If you decide you want to manually remove a device from current swap space, use the **swapoff** command:

```
[root@OCS ~]# swapon -s
Filename   Type      Size    Used  Priority
/dev/dm-1  partition 1048568 27640 -1
/var/swap  file 51192 0 -2
[root@OCS ~]# swapoff /var/swap
[root@OCS ~]# swapon -s
Filename   Type      Size     Used   Priority
/dev/dm-1 partition 048568    27640   -1
```

If you have already created a new partition with the tag type **82** (**/dev/sdb1** in this example), you can format it as a swap device by executing the following command:

```
[root@OCS ~]# mkswap /dev/sdb1
Setting up swapspace version 1, size = 203772 KiB
no label, UUID=039c21dc-c223-43c8-8176-da2f4d508402
```

To create a swap file, you first need to create a large file. This is most easily accomplished by using the **dd** command. The following example demonstrates how to create a 200MB file named **/var/extra_swap** and then enable it as swap space:

```
[root@OCS ~]# dd if=/dev/zero of=/var/extra_swap bs=1M count=200
200+0 records in
200+0 records out
209715200 bytes (210 MB) copied, 4.04572 s, 51.8 MB/s
[root@OCS ~]# mkswap /var/extra_swap
mkswap: /var/extra_swap: warning: don't erase bootbits sectors
        on whole disk. Use -f to force.
Setting up swapspace version 1, size = 204796 KiB
no label, UUID=44469984-0a99-45af-94b7-f7e97218d67a
[root@OCS ~]# swapon /var/extra_swap
```

```
[root@OCS ~]# swapon -s
Filename  Type      Size    Used  Priority
/dev/dm-1 partition 1048568 37664 -1
/dev/sdb1 partition  203768  0    -2
/var/extra_swap file 204792  0    -3
```

Hardware

This section demonstrates some of the files and commands that you can use to display CPU and RAM information.

lscpu

The **lscpu** command can be used to display useful information about the CPU, as shown in the following example:

```
[root@ocs ~]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             4
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 62
Model name:            Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz
Stepping:              4
CPU MHz:               2700.000
BogoMIPS:              5400.00
Hypervisor vendor:     VMware
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
```

```
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0-3
```

Note

CPU information primarily comes from the **/proc/cpuinfo** file. See the "**/proc/cpuinfo**" section in this chapter for more information.

ExamAlert

Don't memorize the output of the **lscpu** command. You should not be asked any specific questions about CPU properties on the Linux+ XK0-005 exam. Rather, you need to know how to view this information.

lsmem

The **lsmem** command can be used to display information about memory (RAM), as shown in the following example:

```
[root@ocs ~]$ lsmem
Address range                       Size (MB)  State    Removable
Device
==========================================================================
==========
0x0000000000000000-0x000000000fffffff        256  online   no
0
0x0000000010000000-0x000000002fffffff        512  online   yes
1-2
0x0000000030000000-0x000000003fffffff        256  online   no
3
0x0000000040000000-0x000000006fffffff        768  online   yes
4-6
0x0000000070000000-0x00000000ffffffff       2304  offline  -
7-15


Memory device size  : 256 MB
Memory block size   : 256 MB
```

```
Total online memory : 1792 MB
Total offline memory: 2304 MB
```

ExamAlert

Don't memorize the output of the **lsmem** command. You should not be asked any specific questions about CPU properties on the Linux+ XK0-005 exam. Rather, you need to know how to view this information.

/proc/cpuinfo

One way of discovering information about CPUs is by viewing the contents of the **/proc/cpuinfo** file. This file contains detailed information about each CPU, as demonstrated in this example:

```
[root@OCS ~]# cat /proc/cpuinfo
processor     : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 13
model name    : QEMU Virtual CPU version (cpu64-rhel6)
stepping      : 3
microcode     : 0x1
cpu MHz       : 2666.760
cache size    : 4096 KB
physical id   : 0
siblings      : 1
core id       : 0
cpu cores     : 1
apicid         : 0
initial apicid : 0
fpu            : yes
fpu_exception  : yes
cpuid level    : 4
wp             : yes
flags : fpu de pse tsc msr pae mce cx8 apic
 sep mtrr pge mca cmov pse36 clflush mmx fxsr sse sse2
 syscall nx lm nopl pni cx16 hypervisor lahf_lm
```

```
bugs :
bogomips        : 5333.52
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 13
model name      : QEMU Virtual CPU version (cpu64-rhel6)
stepping        : 3
microcode       : 0x1
cpu MHz         : 2666.760
cache size      : 4096 KB
physical id     : 1
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 4
wp               : yes
flags : fpu de pse tsc msr pae mce cx8 apic sep
 mtrr pge mca cmov pse36 clflush mmx fxsr sse sse2 syscall nx
 lm nopl pni cx16 hypervisor lahf_lm
bugs :
bogomips         : 5333.52
clflush size     : 64
cache_alignment  : 64
address sizes    : 40 bits physical, 48 bits virtual
power management :
```

ExamAlert

Don't memorize the output of the `cat` command. You should not be asked any specific questions about CPU properties on the Linux+ XK0-005 exam. Rather, you need to know how to view this information.

/proc/meminfo

The **free** command provides a summary of virtual memory (RAM and swap space utilization):

```
[root@OCS ~]# free
     total   used   free   shared buff/cache available
Mem: 3883128 774752 777300 18200 2331076 2784820
Swap: 839676    0   839676
```

The **Mem:** line describes RAM, and the **Swap:** line describes virtual memory. The columns of output are described here:

- **total:** The total amount of memory on the system
- **used:** The amount of memory currently being used
- **free:** The amount of memory available
- **shared:** How much memory is used by **tmpfs**, which is a filesystem that appears to be normal hard disk space but is really storing data in memory
- **buff/cache:** A temporary storage location
- **available:** How much memory is available for new processes

Table 20.1 describes useful options for the **free** command.

TABLE 20.1 **free Command Options**

Option	Description
-b or **--bytes**	Shows information in bytes.
-m or **--mega**	Shows information in megabytes.
-g or **--giga**	Shows information in gigabytes.
-h or **--human**	Displays "human-readable sizes."

If you need more detail than the **free** command provides, you can execute the **vmstat** command. Consider the output shown in Figure 20.2.

```
root@ocs ~]$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 2  0 192860 1617300 292080 3337584    0    0     0    17    0    0  1  0 99  0
```

FIGURE 20.2 **The vmstat Command**

It is not critical to memorize each line of the output shown in Figure 20.2, but you should be familiar with what output the **vmstat** command can provide. For example, the output in the preceding example includes a column labeled **-----io----**. Under this column are two subcolumns:

- **bi:** Also called "blocks in," this value represents how many blocks have been received from a block device (such as a hard disk).
- **bo:** Also called "blocks out," this value represents how many blocks have been sent to a block device (such as a hard disk).

These values can be used to determine whether a performance issue is memory based or disk based. If the **bi** and **bo** values are high, this could mean processes are being blocked on I/O.

Most of the data displayed by the **free** and **vmstat** commands actually comes from the **/proc/meminfo** file, which can be viewed directly (but may be difficult to understand in its raw format).

ExamAlert

Don't memorize the contents of the **/proc/meminfo** file. You should not be asked any specific questions about memory properties on the Linux+ XK0-005 exam. Rather, you need to know how to view this information.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which command can display process CPU and RAM utilization?

   - ❍ **A. ps**
   - ❍ **B. iostat**
   - ❍ **C. nice**
   - ❍ **D. renice**

2. Which of the following values of the **iostat -c** command only applies to virtual CPUs?

   - ❍ **A. %system**
   - ❍ **B. %idle**
   - ❍ **C. %steal**
   - ❍ **D. %nice**

3. In the following output, the value 0.51 represents the load average over the past _____ minute(s).

```
[root@OCS ~]# uptime
16:16:00 up 1 day,  1:05,  6 users,  load average: 0.60, 0.51,
0.25
```

   - ❍ **A.** 1
   - ❍ **B.** 3
   - ❍ **C.** 5
   - ❍ **D.** 15

4. Which option to the **swapon** command displays the currently active swap devices?

   - ❍ **A. -s**
   - ❍ **B. -b**
   - ❍ **C. -c**
   - ❍ **D. -g**

Cram Quiz Answers

1. **D.** The **ps -augx** command can display how much CPU and RAM a process is using. The **iostat** command can provide useful summary of CPU usage (and other resources), but it cannot show CPU and RAM usage per process. The **nice** and **renice** commands control process priority; they do not display usage.
2. **B.** The **%steal** value only pertains to virtual CPUs. In some cases, a virtual CPU must wait for the hypervisor to handle requests from other virtual CPUs. This value indicates the percentage of time waiting for the hypervisor to handle a virtual CPU's request. The other values apply to both physical and virtual CPUs.
3. **C.** The last three numbers (**0.60**, **0.51**, and **0.25** in the output for this question) display the system load averages over the past 1, 5, and 15 minutes.
4. **A.** To see your currently active swap devices, execute the **swapon** command with the **-s** option. The other answers are not valid **swapon** options.

This page intentionally left blank

CHAPTER 21

Analyze and Troubleshoot User Access and File Permissions

This chapter covers the following Linux+ XK0-005 exam objective:

- **4.4:** Given a scenario, analyze and troubleshoot user access and file permissions.

Users often face challenges when working on Linux systems. One challenge is related to the process of logging in to the system. Several components are used to determine if a user can log in, and many of these components have been covered in previous chapters. This chapter introduces some additional user access features and provides useful advice on how to troubleshoot user access issues.

Another common problem that users face is accessing files and directories. As with user access, many topics related to file access, such as permissions, FACLs (file access control lists), and context-based systems (such as SELinux and AppArmor) are covered in previous chapters. This chapter also introduces file attributes and how to troubleshoot file access.

This chapter provides information on the following topics: user login issues, user file access issues, password issues, privilege elevation, and quota issues.

User Login Issues

Two components must be considered when a user attempts to access a system, such as when they attempt to log in: the access itself and authentication. This section focuses on issues that could arise when a user attempts to access a system, and the "User File Access Issues"

section, later in this chapter, focuses on issues that arise when a user account is authenticated.

Local

Accessing a system locally means physically sitting down in front of the system and attempting to log in directly. Fewer problems can arise with local access than with remote access. The following are some of the issues that can come up with local access:

- Hardware malfunctions.
- Entry of incorrect user account information. Consult log files to discover these sorts of errors.
- Pluggable Authentication Modules (PAM) restrictions. (See Chapter 8, "Security Best Practices in a Linux Environment," for more information about PAM.)

With remote access, a user attempts to connect to a system via the network. In addition to the problems just listed for local access, remote access can involve the following problems:

- **Misconfiguration of the service:** For example, if the user is connecting via SSH, the SSH server may be misconfigured or not even running.
- **Service-based security restrictions:** Each service typically has its own security features. For example, SSH servers can be configured to not allow the root user to log in directly. (See Chapter 11, "Configure and Execute Remote Connectivity for System Management," for details regarding SSH servers.)
- **Network-based issues:** If the remote system isn't accessible via the network, the user won't be able to access it due to this network issue.
- **Firewall restrictions:** Firewalls can block access to remote systems, making it impossible for a user to access the system via the network. (See Chapter 10, "Implement and Configure Firewalls," for more information about firewalls.)
- **TCP Wrappers rules:** One of the techniques to filter access to services on a host is to use a library called TCP Wrappers. This library uses simple configuration files (the **/etc/hosts.allow** and **/etc/hosts.deny** files) to

either allow or deny access from specific hosts or networks. This topic is covered in more detail next.

Only services that use the TCP Wrappers library are affected by the **/etc/hosts.allow** and **/etc/hosts.deny** files. You can determine if a program uses this library by using the **ldd** command:

```
[root@OCS ~]# which sshd
/usr/sbin/sshd
[root@OCS ~]# ldd /usr/sbin/sshd | grep libwrap
 libwrap.so.0 => /lib64/libwrap.so.0 (0x00002b003df03000)
```

TCP Wrappers uses the following steps to determine whether access should be allowed or denied:

1. If a match is made in the **/etc/hosts.allow** file, access is granted. If not, the next step is consulted.
2. If a match is made in the **/etc/hosts.deny** file, access is denied. If not, the next step is consulted.
3. If no matches are made in either file, access is granted.

The format of the **/etc/hosts.allow** and **/etc/hosts.deny** files is as follows:

```
service: hosts|network
```

The following command explicitly allows access to the SSHD server for the 192.168.1.1 hosts and the 192.168.99.0/24 network:

```
[root@OCS ~]# cat /etc/hosts.allow
sshd: 192.168.1.1
sshd: 192.168.99.0/24cat
```

Table 21.1 describes how the *service* parameter can be specified.

TABLE 21.1 **Specifying the service Parameter**

Parameter Value	Description
service	Matches a single service.
service,*service*	Matches any of the services listed. (Note that there's no space between services.)
ALL	Matches all services.

Table 21.2 describes how the *hosts* or *network* parameter can be specified.

TABLE 21.2 **Specifying the hosts or network Parameter**

Parameter Value	Description
IP	An IP address
Hostname	A hostname that can be resolved
@group	A NIS group
Network	A network, in any of these formats: 192.168.1, 192.168.99.0/24, 192.168.99.0/255.255.255.0, or .example.com

ExamAlert

TCP Wrappers is not mentioned specifically in the Linux+ XK0-005 exam objectives. However, it is sometimes the cause of remote access failure and has been featured in previous versions of the Linux+ exam.

User File Access Issues

One of the most common actions that users experience when working in Linux is accessing files and directories. These resources can be secured in a variety of ways, which makes it difficult to determine why a user can't access resources. This section describes different file access issues that you need to understand when troubleshooting.

Group

Recall that each user account belongs to at least one group. When accessing files or directories, either the user account or the groups that the user is a member of can provide access to the file or directory. See the "Permission" section, later in this chapter, for additional details.

Context

In terms of user file access, *context* refers to either SELinux or AppArmor security context. These topics are covered in detail in Chapter 12, "Apply the Appropriate Access Controls."

Permission

Sometimes it can be really frustrating to determine why you cannot access a file or directory. For example, consider the following example:

```
bob@OCS:~# cp sample /etc/skel
cp: cannot create regular file '/etc/skel/sample:
 Permission denied
```

What is the problem here? Keep in mind that several permissions are checked, such as the read permission on the **sample** file; the execute permissions on the **/**, **etc**, and **skel** directories; and the write permissions on the **skel** directory.

To discover the error, first look closely at the output of the error message. In this case, the problem seems to be with creating the file in the **/etc/skel** directory, not with the **sample** file itself (which would have resulted in an error like "cannot open 'sample1' for reading").

Next, determine if you can get into all of the directories by either looking at the permissions for each one or using the **cd** command to move into each directory. Finally, look for the write permission on the destination directory.

Directory permissions can be tricky, making it difficult to troubleshoot access to a directory. For example, consider the following example:

```
[julia@OCS mydata]$ ls -l info
-rwxrwxrwx 1 julia julia 240 Feb 4 22:16 info
[julia@OCS mydata]$ rm info
rm: cannot remove 'info': Permission denied
```

It doesn't seem to make any sense that the **julia** user isn't able to delete the **info** file. The file is owned by the **julia** account, and the **julia** account has all permissions on the file. The problem here is that the permission required to delete a file is the write permission in the directory the file is in, as shown here:

```
[julia@OCS mydata]$ ls -ld .
dr-xrwxr-x 2 julia julia 4096 Feb 4 22:16 .
[julia@OCS mydata]$ chmod u+w .
[julia@OCS mydata]$ ls -ld .
drwxrwxr-x 2 julia julia 4096 Feb 4 22:16 .
[julia@OCS mydata]$ rm info
```

When you deal with permissions problems, consider what permissions you need to complete the task at hand. Consider writing out the command using a full path. For example, in this case, instead of thinking about the **rm info** command, think about the **rm /home/julia/data/info** command. Review the permissions on each directory to determine where the problem might be.

ACL

Recall that file ACLs can provide additional access to files and directories. When reviewing why a user or group might not have access to a file or directory, don't forget to run the **getfacl** command on the resource to see if there is a specific rule that blocks access.

Attribute

While not technically permissions, file attributes do affect how users access files and directories, so they logically belong in any discussion regarding permissions. With file attributes, the system administrator can modify key features of file access.

For example, a useful file attribute is one that makes a file immutable. An immutable file is completely unchangeable; it cannot be deleted or modified by anyone, including the root user. To make a file immutable, use the **chattr** command:

```
[root@OCS ~]$ chattr +i /etc/passwd
```

Note that now no user can change the **/etc/passwd** file, which means no new users can be added to the system (and existing users cannot be removed). This might seem like an odd thing to do, but imagine a situation in which a system is publicly available (like a kiosk in a mall). There is no need for new users, and you do not want anyone to remove users either.

To see the attributes for a file, use the **lsattr** command:

```
[root@OCS ~]$ lsattr /etc/passwd
----i--------e-- /etc/passwd
```

The **-** characters indicate file attributes that are not set. A complete list of attributes can be found in the man page for the **chattr** command. Table 21.3 describes the file attributes that are important for system security.

TABLE 21.3 **File Attributes That Are Important for System Security**

Attribute	Description
a	Specifies append-only mode so that only new data can be placed in the file.
A	Prevents modification of the access timestamp. This timestamp can be important for security reasons to determine when key system files have been accessed. However, for non-critical files, disabling the access time can make the system faster as it results in fewer hard drive writes.
e	Specifies extent format, which allows for key features such as SELinux (discussed later in this chapter).
i	Makes the file immutable, which means the file cannot be deleted or modified.
u	Makes the file undeletable, which means the file cannot be deleted, but its contents can be modified.

To remove the immutable file attribute, use the following command:

```
[root@OCS ~]$ chattr -i /etc/passwd
[root@OCS ~]$ lsattr /etc/passwd
-------------e-- /etc/passwd
```

Table 21.4 describes important options for the **chattr** command.

TABLE 21.4 **chattr Command Options**

Option	Description
-R	Recursively applies changes to an entire directory structure.
-V	Displays verbose messages when producing output demonstrating the changes made.

ExamAlert

Remember for the Linux+ XK0-005 exam that only the root user can change file attributes. A regular user can see the attributes but cannot change them, even if that user owns the file.

Password Issues

Passwords are at the heart of the authentication process. The authentication process involves determining whether a user is permitted to access a system. For example, is the user providing the correct credentials, such as the correct password? Or is the user logging in during the correct time and day? This section focuses on how to determine whether a problem arises during the authentication process.

Local user authentication typically relies on password information stored in the **/etc/shadow** file. A user could have local authentication issues for the following reasons:

- The user forgot the account password.
- The account has passed its expiration date.
- The account has password aging restrictions, and the user has violated these restrictions. For example, an account can be configured to require a new password every 60 days. If the user doesn't change the account password within this timeframe, the account can be locked out.
- An administrator may have manually locked a user account.
- A PAM restriction may limit user authentication.

Here are some helpful hints related to troubleshooting local authentication issues:

- Check the security log file for error messages.
- Review the user account settings in the **/etc/shadow** file. See Chapter 9, "Implement Identity Management," for details about this file.
- Review PAM restrictions. See Chapter 8, "Security Best Practices in a Linux Environment," for more information about PAM.

ExamAlert

For the Linux+ XK0-005 exam, be sure you know the fields of the **/etc/shadow** file. You are likely to see a question that asks you specifics about the contents of this file.

Privilege Elevation

Recall that privilege elevation is the use of either PolicyKit rules, **sudo**, **su**, or **pkexec** to gain privilege (typically root account access) when logged in to a non-privileged account (that is, a regular user account). Chapter 11, "Configure and Execute Remote Connectivity for System Management," provides complete coverage of these topics.

Some suggestions when troubleshooting privilege elevation issues include the following:

- Consult the log files to gather additional information. Files like the **/var/log/secure** log often contain entries that describe why an attempt failed.
- Ask the user who is attempting to gain privilege elevation to provide the output of the command that failed.
- Review the configuration file for the tool being used to determine if it is configured correctly.
- Review PAM configurations to ensure that there isn't a PAM rule that blocks privilege elevation attempts for the user. See Chapter 8 for more information about PAM.

Quota Issues

Quotas provide an administrator with the ability to limit how much disk space can be used by individuals or groups. Quotas are per filesystem. This section focuses on how to enable and view quotas.

To enable user quotas, you must mount the filesystem with the **usrquota** mount option. This can be accomplished by adding **usrquota** to the mount option field of the **/etc/fstab** file, as shown here:

```
/dev/sdb1 / ext4 usrquota 1 1
```

Note

To enable group quotas, use the **grpquota** boot option. The **usrquota** and **grpquota** options can be used together (**usrquota,grpquota**) to enable both user and group quotas on a filesystem.

Then you can remount the filesystem with the following command (executed by the root user):

```
[root@OCS ~]$ mount -o remount /
```

See the "**/etc/fstab**" and "**mount**" sections in Chapter 3, "Configure and Manage Storage Using the Appropriate Tools," for details regarding these tasks.

After mounting the filesystem with the **usrquota** option enabled, you need to create the initial quota databases by executing the **quotacheck** command:

```
[root@OCS ~]$ quotacheck -cugm /dev/sdb1
```

This results in new files in the mount point directory of the filesystem:

```
[root@OCS ~]$ ls /aquota*
/aquota.group /aquota.user
```

Table 21.5 describes some important options for the **quotacheck** command.

TABLE 21.5 **quotacheck Command Options**

Option	Description
-c	Creates database file(s).
-g	Only creates the **aquota.group** file, which means only group quotas will be enabled unless the **-u** option is also used.
-m	Prevents unmounting of the filesystem while creating the quota file(s).
-u	Only creates the **aquota.user** file, which means only user quotas will be enabled unless the **-g** option is also used.

To create or edit a user's quotas, execute the **edquota** command followed by the username (as the root user):

```
[root@OCS ~]$ edquota sarah
```

Note

Use the **-g** option to the **edquota** command to edit a group quota.

The **edquota** command enters an editor (vi is typically the default) and displays all of the user's quotas. The output will look something like the following:

```
Disk quotas for user sarah (uid 507):
 Filesystem blocks soft hard inodes soft hard
 /dev/sdb1  550060 0    0     29905  0    0
```

Table 21.6 describes the fields of the quota.

TABLE 21.6 **Quota Fields**

Key	Description
Filesystem	The partition that contains the filesystem with quotas enabled.
blocks	How many blocks the user currently uses in the filesystem.
soft	A value that represents a soft quota for blocks; if the user creates a file that results in exceeding this block limit, a warning is issued.
hard	A value that represents a hard quota for blocks; if the user creates a file that results in exceeding this block limit, an error is issued, and no additional files can be created in the filesystem.
inodes	How many files the user currently has in the filesystem.
soft	A value that represents a soft quota for files; if the user creates a file that results in exceeding this file limit, a warning is issued.
hard	A value that represents a hard quota for files; if the user creates a file that results in exceeding this file limit, an error is issued, and no additional files can be created in the filesystem.

ExamAlert

The grace period can be set by executing the **edquota -t** command.

The **quota** command can be executed by a user to display the quotas for the account:

```
[sarah@OCS ~]$ quota
Disk quotas for user sarah (uid 507):
    Filesystem blocks quota   limit grace files quota limit grace
     /dev/sda1   20480    30000     60000    1     0     0
```

Note the output when a user has exceeded a soft quota. In the following example, the user sarah is above the soft limit for block size:

```
[sarah@OCS ~]$ quota
Disk quotas for user sarah (uid 507):
    Filesystem blocks quota limit grace files quota limit grace
 /dev/sda1   40960*  30000 60000 7days    2     0    0
```

Once the user has exceeded a soft quota, a grace period is provided. The user must reduce the space used in the filesystem to be below the soft quota within the grace period or else the current usage converts to a hard quota limit.

Table 21.7 describes some important options for the **quota** command.

TABLE 21.7 **quota Options**

Option	Description
-g	Displays group quotas instead of specific user quotas.
-s	Displays information in human-readable sizes rather than block sizes.
-l	Displays quota information only for local filesystems (rather than network-based filesystem quotas).

The root user can use the **repquota** command to display quotas for an entire filesystem:

```
[root@localhost ~]$ repquota /
*** Report for user quotas on device /dev/sda1
Block grace time: 7days; Inode grace time: 7days
 Block limits File limits
User used  soft hard grace used soft hard grace
----------------------------------------------------------------------
---
root       -- 4559956  0   0    207396  0  0
daemon     -- 64       0   0    4       0  0
man        -- 1832     0   0    145     0  0
www-data    -- 4       0   0    1       0  0
libuuid    -- 40       0   0    6       0  0
syslog     -- 3848     0   0    23      0   0
messagebus -- 8        0   0    2       0  0
landscape  -- 8        0   0    2       0  0
```

```
pollinate  -- 4         0   0    1       0   0
vagrant    -- 550060    0   0    29906   0   0
colord     -- 8         0   0    2       0   0
statd      -- 12        0   0    3       0   0
puppet     -- 44        0   0    11      0   0
ubuntu     -- 36        0   0    8       0   0
sarah     +- 40960   30000   60000   6days   2   0   0
```

Table 21.8 describes some important options for the **repquota** command.

TABLE 21.8 **repquota Command Options**

Option	Description
-a	Displays quotas for all filesystems with quota mount options specified in the **/etc/fstab** file.
-g	Displays group quotas instead of specific user quotas.
-s	Displays information in human-readable sizes rather than block sizes.

The **quotaon** command is used to turn on quotas for a filesystem. Normally when the system is booted, it turns on quotas automatically. However, you can turn off quotas by executing the **quotaoff** command followed by the name of the filesystem (as the root user):

```
[root@OCS ~]$ quotaoff /dev/sdb1
[root@OCS ~]$ quotaon /dev/sdb1
```

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which files are used by TCP Wrappers? (Choose two.)

   - ❍ **A. /etc/hosts.reject**
   - ❍ **B. /etc/hosts.allow**
   - ❍ **C. /etc/hosts.deny**
   - ❍ **D. /etc/hosts.permit**

2. Which of the following provide context-based access to files and directories? (Choose two.)
   - ❍ **A.** ACLs
   - ❍ **B.** Permissions
   - ❍ **C.** SELinux
   - ❍ **D.** AppArmor

3. Which file attribute prevents modification of the access timestamp?
   - ❍ **A. a**
   - ❍ **B. A**
   - ❍ **C. e**
   - ❍ **D. i**

4. Which command can be used by a regular user to display the user's disk quota?
   - ❍ **A. edquota**
   - ❍ **B. quota**
   - ❍ **C. quotacheck**
   - ❍ **D. repquota**

Cram Quiz Answers

1. **B and C.** TCP Wrappers uses simple configuration files (the **/etc/hosts.allow** and **/etc/hosts.deny** files) to either allow or deny access from specific hosts or networks. The other answers are not valid files for TCP Wrappers.
2. **C and D.** In terms of user file access, context refers to either SELinux or App-Armor security context. ACLs and permissions are user-based access controls, not context-based controls.
3. **B.** The **A** attribute prevents modification of the access timestamp. The **a** attribute only allows new data to be placed in the file. The **e** attribute allows for key features such as SELinux. The **i** attribute makes the file immutable.
4. **B.** The **quota** command can be executed by a user to display the quotas for the account. The root user can use the **repquota** command to display quotas for an entire filesystem. The other two commands cannot be executed by a regular user and are not designed to display quotas.

CHAPTER 22

Use systemd to Diagnose and Resolve Common Problems with a Linux System

This chapter covers the following Linux+ XK0-005 exam objective:

- **4.5:** Given a scenario, use **systemd** to diagnose and resolve common problems with a Linux system.

This last chapter of the book is a bit unusual because of the manner in which it is addressed in the Linux+ objectives. The **systemd** service has been discussed briefly in a couple other chapters, but many of the topics in this chapter have not been covered previously. Given that the objective for this chapter is "Given a scenario, use **systemd** to diagnose and resolve common problems with a Linux system," to pass the Linux+ XK0-005 exam, you need to have a solid understanding of **systemd**.

The goal of this chapter is to introduce how **systemd** impacts several components of the operating system. **systemd** is involved in booting the system, controlling system processes, mounting storage devices, scheduling processes, and additional features that are covered in this chapter. While learning these **systemd** features, you will also be introduced to techniques for troubleshooting problems that arise with **systemd**.

This chapter provides information on unit files and common problems.

Unit Files

A unit file is used to define a service. (Services are discussed later in this section.) To display which unit files are defined on a system, use the **list-units** option to the **systemctl** command:

```
# systemctl list-units | head
UNIT FILE STATE
proc-sys-fs-binfmt_misc.automount  static
-.mount generated
dev-hugepages.mount                          static
dev-mqueue.mount                             static
proc-sys-fs-binfmt_misc.mount                static
snap-core-5662.mount                         enabled
snap-core-5742.mount                         enabled
snap-core-6130.mount                         enabled
snap-gnome\x2d3\x2d26\x2d1604-64.mount       enabled
snap-gnome\x2d3\x2d26\x2d1604-70.mount       enabled
```

To view the contents of a unit file, use the **cat** option to the **systemctl** command:

```
# systemctl cat cups
# /lib/systemd/system/cups.service
[Unit]
Description=CUPS Scheduler
Documentation=man:cupsd(8)

[Service]
ExecStart=/usr/sbin/cupsd -l
Type=simple
Restart=always

[Install]
Also=cups.socket cups.path
WantedBy=printer.target
```

The first line of this output provides the location of the unit file. The `[Unit]` section of the output describes the service. The `[Service]` section specifies what process should be started as well as some options for how the process should be configured.

The `[Install]` section includes the `WantedBy` setting, which indicates which target "wants" to start this service. It may also contain an `Also` setting that is used to indicate additional services that should be started.

> **ExamAlert**
>
> The term *service* is used here, but unit files can define many different **systemd** components, including mount points, devices, sockets, and more.

You shouldn't worry about how to create your own unit file, but you should be familiar with the function of such a file and the settings described in the following sections of this chapter.

Additional unit file commands include the following:

- **systemctl list-dependencies** *unit***:** Lists what other features or services are required by this feature or service.
- **systemctl show** *unit***:** Lists properties of a unit.
- **systemctl edit** *unit***:** Used to edit a unit file.

Service

A *service* is a **systemd** configuration that describes a process or an operating system feature that is controlled by **systemd**. You can see a list of the services that a system manages by viewing the files that end in **.service** in the **/lib/systemd/system** directory. The following is a partial list on a typical CentOS system:

```
[root@ocs ~]$ ls /lib/systemd/system/*.service | head
/lib/systemd/system/abrt-ccpp.service
/lib/systemd/system/abrtd.service
/lib/systemd/system/abrt-oops.service
/lib/systemd/system/abrt-pstoreoops.service
/lib/systemd/system/abrt-vmcore.service
/lib/systemd/system/abrt-xorg.service
```

```
/lib/systemd/system/acpid.service
/lib/systemd/system/arp-ethers.service
/lib/systemd/system/atd.service
/lib/systemd/system/auditd.service
```

Networking Services

The networking services are controlled by service files such as the following:

- **/lib/systemd/system/NetworkManager.service**
- **/lib/systemd/system/NetworkManager-wait-online.service**
- **/lib/systemd/system//lib/systemd/system/NetworkManager.service**

These service unit files manage the different NetworkManager processes, as discussed in Chapter 5, "Use the Appropriate Networking Tools or Configuration Files."

ExecStart/ExecStop

The **ExecStart** and **ExecStop** settings in a **systemd** service unit file describe how to start or stop a service. For example, consider the following file, which describes the NetworkManager service:

```
[root@ocs ~]$ cat /lib/systemd/system/NetworkManager.service
[Unit]
Description=Network Manager
Wants=network.target
Before=network.target network.service

[Service]
Type=dbus
BusName=org.freedesktop.NetworkManager
ExecStart=/usr/sbin/NetworkManager --no-daemon
# NM doesn't want systemd to kill its children for it
KillMode=process

[Install]
WantedBy=multi-user.target
```

```
Alias=dbus-org.freedesktop.NetworkManager.service
Also=NetworkManager-dispatcher.service
```

If **systemd** attempts to start this service, the **/usr/sbin/NetworkManager --no-daemon** command is executed because that is what is defined by **ExecStart**. If a process needs to be stopped by **systemd**, an **ExecStop** setting is used.

Before/After

The **Before** and **After** settings in the **systemd** service unit files are used to control the order in which services are started. For example, consider the highlighted lines in the following service unit file that is used to manage the firewall daemon:

```
[root@ocs ~]$ cat /lib/systemd/system/firewalld.service
[Unit]
Description=firewalld - dynamic firewall daemon
Before=network.target
Before=libvirtd.service
Before=NetworkManager.service
Conflicts=iptables.service ip6tables.service ebtables.service

[Service]
EnvironmentFile=-/etc/sysconfig/firewalld
ExecStart=/usr/sbin/firewalld --nofork --nopid $FIREWALLD_ARGS
ExecReload=/bin/kill -HUP $MAINPID
# supress to log debug and error output also to /var/log/messages
StandardOutput=null
StandardError=null
Type=dbus
BusName=org.fedoraproject.FirewallD1

[Install]
WantedBy=basic.target
Alias=dbus-org.fedoraproject.FirewallD1.service
```

This indicates that the firewall must be started before the **NetworkManager.service** and **libvirtd.service** services are started. For the Linux+ XK0-005 exam, you don't need to worry about why a specific service should be started

before another service; you just need to understand the relationship. In this case, it is pretty clear that the firewall should be up before NetworkManager activates the network. This is because if the network is active first, then some network communication that should have been blocked by the firewall might gain access to the system.

Note that there is a third **Before** value, **network.target**, in the previous example. Targets are covered later in this chapter, but for now consider targets as a way to define functional states in **systemd**. For example, the **network.target** file is used to define the functional state of the network service being enabled. See the "Target" section, later in this chapter, for more information on targets.

Type

The term *type* in relation to **systemd** can be a bit confusing as it may refer to either of two things:

- **Type of unit:** This is the type of **systemd** configuration. For example, **systemd** configuration files that end in **.service** are of the service type, while files that end in **.target** are of the target type. Other common types include mount, device, and timer.
- **Type setting in a service configuration file:** This defines how the process is started in a **systemd** service unit configuration file. This can be a value like **simple**, **exec**, **forking**, **oneshot**, **dbus**, **notify**, or **idle**. These types are all similar in that they all define how a service is started, but they have subtle differences. For example, with **simple**, **systemd** considers the process to be started once the process *forks* (that is, initiates a new process and starts executing), while with **notify**, **systemd** expects the process to send a message once it has started.

ExamAlert

It is unlikely that the Linux+ XK0-005 exam will ask you specific questions about the differences in how processes are started by **systemd**. The differences are subtle, and normally only individuals who create service unit files need to worry about the differences.

User

The **User** setting indicates which user account should be used to start a service. For example, the process started by the following **postgresql.service** file should be started using the **postgres** user account:

```
[root@ocs ~]$ grep -A2 -B3 "User" /lib/systemd/system/
postgresql.service
[Service]
Type=forking

User=postgres
Group=postgres
```

Requires/Wants

The **Requires** value in a service unit file indicates what resources are required for the service to be started. This is similar to the **After** value, but **After** is used to define the order in which services need to be started, whereas **Requires** is used to indicate that the service needs a particular resource before it starts.

The **Requires** value can be different types, such as another service, a target, or another resource type, such as a socket (which is a resource used for network communication). For example, the highlighted text in the following example demonstrates that **nfslock.service** is dependent on the **rpcbind.service** and the **network.target** resources being enabled:

```
[root@ocs ~]$ cat /lib/systemd/system/nfslock.service
[Unit]
Description=NFS file locking service.
Requires=rpcbind.service network.target
After=network.target named.service rpcbind.service
Before=remote-fs-pre.target

[Service]
Type=forking
StandardError=syslog+console
EnvironmentFile=-/etc/sysconfig/nfs
ExecStartPre=/usr/libexec/nfs-utils/scripts/nfs-lock.preconfig
ExecStart=/sbin/rpc.statd $STATDARG
```

```
# Make sure lockd's ports are reset
ExecStopPost=-/sbin/sysctl -w fs.nfs.nlm_tcpport=0 fs.nfs.
nlm_udpport=0


[Install]
WantedBy=nfs.target
```

Note that this example also includes a **WantedBy** setting. This is used to indicate another resource that "wants" this resource. In other words, the **nfs.target** wants the **nfslock.service** to be enabled in order to reach the **nfs.target** system state. You could determine that other services are wanted by **nfs.target** by running a command like the following:

```
[root@ocs ~]$ grep "WantedBy=nfs.target"
/lib/systemd/system/*.service
/lib/systemd/system/nfs-blkmap.service:WantedBy=nfs.target
/lib/systemd/system/nfs-idmap.service:WantedBy=nfs.target
/lib/systemd/system/nfs-lock.service:WantedBy=nfs.target
/lib/systemd/system/nfslock.service:WantedBy=nfs.target
/lib/systemd/system/nfs-mountd.service:WantedBy=nfs.target
/lib/systemd/system/nfs-rquotad.service:WantedBy=nfs.target
/lib/systemd/system/nfs-secure-server.service:WantedBy=nfs.target
/lib/systemd/system/nfs-secure.service:WantedBy=nfs.target
/lib/systemd/system/nfs-server.service:WantedBy=nfs.target
/lib/systemd/system/nfs.service:WantedBy=nfs.target
/lib/systemd/system/rpcgssd.service:WantedBy=nfs.target
/lib/systemd/system/rpcidmapd.service:WantedBy=nfs.target
/lib/systemd/system/rpcsvcgssd.service:WantedBy=nfs.target
```

Timer

In Chapter 4, "Configure and Use the Appropriate Processes and Services," you learned how you can schedule the execution of processes by using the **crontab** and **at** services. The **systemd** service also enables you to schedule processes to run at specific intervals. This section explores how to use the **systemd** timer to schedule process execution.

A timer unit file is used to schedule the execution of processes based on times. Depending on what software you have installed, there might already be some timer unit files on your system. This is how you use the timer:

```
root@ocs ~]$ ls /lib/systemd/system/*.timer
/lib/systemd/system/systemd-readahead-done.timer
/lib/systemd/system/systemd-tmpfiles-clean.timer
/lib/systemd/system/unbound-anchor.timer
/lib/systemd/system/yum-makecache.timer
```

The following is an example of a timer unit file:

```
[root@ocs ~]$ cat /lib/systemd/system/yum-makecache.timer
[Unit]
Description=yum makecache timer
ConditionKernelCommandLine=!rd.live.image

[Timer]
OnBootSec=10min
OnUnitInactiveSec=1h
Unit=yum-makecache.service

[Install]
WantedBy=basic.target
```

Some of the key settings in this file include the following:

- **OnBootSec:** Defines when to run the process, based on how long it has been since the system was booted. In this example, the process will be executed 10 minutes after system boot.
- **OnUnitInactiveSec:** Defines a time to execute the process since the last time the process executed. In this example, the process is defined to execute 1 hour after the last time it was executed. In other words, it will run every hour.
- **Unit:** This is the service unit file that defines the process to execute. In order to determine the actual process that will be executed, you need to look at the service unit file for the **ExecStart** value. For example, this

example indicates **Unit=yum-makecache.service**, and looking at the **ExecStart** value of the **yum-makecache.service** file shows what process will be executed by the timer unit file:

```
root@ocs ~]$ cat /lib/systemd/system/yum-makecache.service
[Unit]
Description=yum makecache

[Service]
Type=oneshot
Nice=19
IOSchedulingClass=2
IOSchedulingPriority=7
ExecStart=/usr/bin/yum makecache fast
```

Another common method of defining when a service should execute is to use the **OnCalendar** setting, as shown in the following example:

```
root@ocs ~]$ cat /lib/systemd/system/unbound-anchor.timer
[Unit]
Description=daily update of the root trust anchor for DNSSEC
Documentation=man:unbound-anchor(8)

[Timer]
# Current DNSKEY TTL in root zone is 172800 seconds, i.e.
172800/60/60/24 = 2 days.
# It means that unboud-anchor should be run at least once a day.
OnCalendar=daily
Persistent=true
AccuracySec=24h

[Install]
WantedBy=timers.target
```

The setting **OnCalendar=daily** indicates that the process will be executed once per day. See the following section, "Time Expressions," for additional examples of values that can be used with the **OnCalendar** setting.

Time Expressions

The expressions that can be used in conjunction with the **OnCalendar** setting are very diverse. For example, you could use the following value to execute a process on Sunday, May 1, 2022, at 13:30:00 (*hour*:*minute*:*second* in 24-hour format, so 1:30 p.m.):

```
OnCalendar=Sun 2022-05-01 13:30:00
```

An asterisk character indicates "all valid values," so to run a process every hour on Sunday, May 1, 2022, at half past the hour, use the following:

```
OnCalendar=Sun 2022-05-01 *:30:00
```

A number of keywords are already defined, including the following:

- **minutely:** The same as ***-*-* *:*:00**
- **hourly:** The same as ***-*-* *:00:00**
- **daily:** The same as ***-*-* 00:00:00**
- **monthly:** The same as ***-*-01 00:00:00**
- **weekly:** The same as **Mon *-*-* 00:00:00**
- **yearly:** The same as ***-01-01 00:00:00**
- **quarterly:** The same as ***-01,04,07,10-01 00:00:00**
- **semiannually:** The same as ***-01,07-01 00:00:00**

Ranges can be indicated by using **..** between values. For example, the following would indicate to run a process at 2 a.m. every workday (Monday through Friday):

```
OnCalendar=Mon..Fri *-*-* 02:00:00
```

Multiple times can be indicated by using **,** between values. For example, the following would indicate to run a process at 8 a.m., 2 p.m., and 6 p.m. every workday (Monday through Friday):

```
OnCalendar=Mon..Fri *-*-* 08,14,18:00:00
```

Mount

Automounting is the process of mounting a resource when a process accesses the mount point. After the process stops using the mount point, the resource

is unmounted. Typically automounting is set up for mounting remote network shares or for mounting removable media, such as CD-ROMs and USB disks.

You create an automount by creating the unit files, as shown here:

```
$ cat /etc/systemd/system/mnt-test.automount
[Unit]
Description=Automount Test

[Automount]
Where=/mnt/test

[Install]
WantedBy=multi-user.target

$ cat /etc/systemd/system/mnt-test.mount
[Unit]
Description=Test

[Mount]
What=nfs.example.com:/export/test
Where=/mnt/test
Type=nfs

[Install]
WantedBy=multi-user.target
```

ExamAlert

The automount unit filename must match the mount point name (**/mnt/test** = **mnt-test.automount**).

When you have created new unit files, you need to execute the **systemctl daemon-reload** command.

Here is how you disable the mount unit and enable the automount unit:

```
$ systemctl is-enabled mnt-test.mount
disabled
```

```
$ systemctl is-enabled mnt-test.automount
enabled
```

To switch to the automount mount point, use the **cd** command.

In addition to enabling automounting, **systemd** can perform traditional Linux mounting. This can either be defined in the **/etc/fstab** file, which is described in Chapter 3, "Configure and Manage Storage Using the Appropriate Tools," or by using mount unit files, as described in the following sections.

Naming Conventions

The following unit file demonstrates the naming conventions you see when creating a mount unit file:

```
[Unit]
Description=Storage drive

[Mount]
What=/dev/disk/by-uuid/72b3acbf-eb3f-457b-8442-55c17044d88b
Where=/data
Type=ext4
Options=defaults

[Install]
WantedBy=multi-user.target
```

The components **What**, **Where**, **Type**, and **Options** are described in the sections that follow.

What

The **What** component defines the device to mount. Note that this should be provided using the device's UUID (universally unique identifier). If you don't know the UUID of a device, you can use the **lsblk -f** command to display all the UUIDs for all devices:

```
[root@ocs ~]$ lsblk -f
NAME   FSTYPE LABEL UUID                                 MOUNTPOINT
sda
├─sda1 ext4         72b3acbf-eb3f-457b-8442-55c17044d88b /
└─sda2 swap         e02f16c6-fceb-44c0-919f-c520045a0a60 [SWAP]
```

Where

The **Where** component defines where to mount the device, also known as the *mount point*. This should be an empty directory.

Type

The **Type** component defines the type of filesystem that is being mounted. Typically this is a value like **ext4** or **xfs**. If you are not sure of the filesystem type of a device, use the **lsblk -f** command to display this information:

```
[root@ocs ~]$ lsblk -f
NAME   FSTYPE LABEL UUID                                 MOUNTPOINT
sda
├─sda1 ext4         72b3acbf-eb3f-457b-8442-55c17044d88b /
└─sda2 swap         e02f16c6-fceb-44c0-919f-c520045a0a60 [SWAP]
```

Options

The **Options** component is used to define what options to use when mounting the filesystem. There are many mount options that can be used, and exactly which options are available depends on the filesystem type that is being mounted. Table 22.1 describes the mount options that are supported by most filesystems.

TABLE 22.1 **Mount Options**

Option	Description
rw	Mounts the filesystem as read/write. To mount the filesystem as read-only, use the option **ro**.
suid	Enables the SUID permission set on files in the filesystem. (See Chapter 12, "Apply the Appropriate Access Controls," for details regarding the SUID permission.) Use the **nosuid** option to disable the SUID permission set. Disabling the SUID permission set on filesystems is normally done to prevent a user from creating a SUID program, which could result in a security risk. Note that the **/** and **/usr** filesystem should never be mounted with the **nosuid** option.
dev	Permits device files to be used on the filesystem. Normally device files are only placed under the **/dev** directory, so for security purposes, using the **nodev** option on all filesystems (besides the **/** filesystem) would result in a more secure system.

Option	Description
exec	Enables executable files in the filesystem. Use the **noexec** option to disable executable files. Disallowing executable files might be a good idea on servers to prevent users from introducing new programs to the server.
auto	Used along with the **-a** option to the **mount** command. The **-a** option means "mount all filesystems that have the **auto** option in the **/etc/fstab** file." Keep in mind that when the system boots, the **mount -a** command is used to mount filesystems, so using the **noauto** option would also mean the filesystem is not mounted automatically during the boot process.
nouser	Prevents a regular user from mounting the filesystem. This is typically the best option for security reasons, but there are some cases in which you might want a user to mount a removable device. In those cases, use the **user** option. If you employ the **user** option, then once the filesystem is mounted, only that user can unmount the filesystem. The **user** option allows any user to mount the filesystem, and it also allows any user to unmount the filesystem.
async	Causes metadata to be stored temporarily in memory. The process of performing a sync is to ensure that all data has been written to the hard drive. Normally when files are created or modified, the contents of the file are written directly to the hard drive, but the metadata (information about the file) is stored temporarily in memory. This reduces excessive hard drive writes and prevents thrashing, a situation that occurs when the hard drive is overtaxed. At regular intervals, this metadata is written to the hard drive, in a process called *syncing*. In some cases, temporarily storing metadata in memory can cause issues because this metadata could be lost if the system is improperly powered off. If this metadata is critical, consider using the **sync** option but realize that it could slow down your system considerably. Note that if you are using the **async** option, you can write the metadata to the hard drive by executing the **sync** command.
relatime	Enables updating of the access timestamp. Each file has three timestamps: the last time the contents of the file were modified, the last time the metadata was changed, and the last time the file was accessed. The file access time is normally not critical, but it can cause a lot of system writes, especially if you execute a command that recursively executes on a directory structure (such as **grep -r**). If updating the access timestamp causes a performance issue, consider changing the option to **norelatime**.

Note that when **defaults** is provided for the **mount** command, it means that the following options are used: **rw**, **suid**, **dev**, **exec**, **auto**, **nouser**, and **async**. As previously mentioned, these are not the only options; there are in fact

many more. To discover these additional options, explore the man page for the **mount** command. First look at the section titled "Filesystem Independent Mount Options," which covers options that work for most filesystems. Then search for the filesystem that you are using. The following output shows a portion of the **mount** man page related to ext4 filesystems:

```
Mount options for ext4
       The ext4 filesystem is an an advanced level of the
       ext3 filesystem which incorporates scalability
       and reliability enhancements for supporting large
       filesystem.

       The options journal_dev, noload, data, commit,
       orlov, oldalloc, [no]user_xattr [no]acl, bsddf,
       minixdf, debug, errors, data_err, grpid, bsdgroups,
       nogrpid sysvgroups, resgid, resuid, sb, quota,
       noquota, grpquota, usrquota and [no]bh are backwardly
       compatible with ext3 or ext2.

       journal_checksum
              Enable checksumming of the journal transactions.
              This will allow the recovery code in e2fsck and
              the kernel to detect corruption in the kernel. It
              is a compatible change and will be ignored by
              older kernels.

       journal_async_commit
              Commit block can be written to disk without waiting
              for descriptor blocks. If enabled older kernels
              cannot mount the device.
```

Target

A *target* is a functional state of a system. Each target has specific services that start. Figure 22.1 shows an example of a typical **systemd** boot sequence.

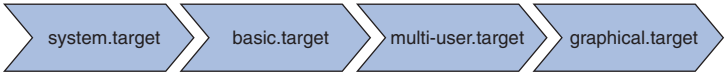

FIGURE 22.1 **Overview of the systemd Boot Process**

Targets are defined in the **/usr/lib/systemd/system** directory. Consider the following example of a target file, which describes the functional state of having the graphical user interface running:

```
# cat /usr/lib/systemd/system/graphical.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License
  as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target
 display-manager.service
AllowIsolate=yes
```

A few of the key elements of this file are:

- **Requires:** The target or service that must be enabled for this target to be started. In this example, the system must first bring the state to **multi-user.target** before bringing the state to the target that starts the graphical user interface.
- **Wants:** The service that this target wants started in order to reach the functional state of this target. In this case, **display-manager.service** is the service that starts the graphical user interface.

The default target is defined by a symbolic link from **/etc/systemd/system/default.target** to the target in the **/usr/lib/systemd/system** directory, as in this example:

```
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 36 Jun 11 20:47
```

```
/etc/systemd/system/default.target ->
/lib/systemd/system/graphical.target
```

Use the following command to display the default target:

```
# systemctl get-default
multi-user.target
```

Use the **systemctl list-unit-files --type=target** command to list the available targets. This command provides a large amount of output, and the following example uses the **head** command to limit the output:

```
# systemctl list-unit-files --type=target | head
UNIT FILE STATE
basic.target              static
bluetooth.target          static
cryptsetup-pre.target     static
cryptsetup.target         static
ctrl-alt-del.target       disabled
cvs.target                static
default.target            enabled
emergency.target          static
final.target              static
```

Use the following command to set the default target:

```
# systemctl set-default graphical-user.target
rm '/etc/systemd/system/default.target'
ln -s '/usr/lib/systemd/system/graphical-
 user.target' '/etc/systemd/system/default.target'
```

Multiuser

The **multiuser** target defines a system state in which multiple users can log in to the system. At this target, networking should be enabled, and the graphical user interface is not enabled.

Network-online

The **network-online** target is used to define a system state in which networking is enabled.

Graphical

The **graphical** target is used to define a system state in which the graphical user interface is enabled.

Common Problems

This section describes common problems that you might experience when working with **systemd**.

Name Resolution Failure

On systems on which it is enabled, **systemd-resolved.service** handles name resolution. This service is configured by modifying the **/etc/systemd/resolved.conf** file, which might look like the following:

```
[Resolve]
DNS=10.0.0.1 10.0.0.2
Domains=~example.com
```

If you are having issues with name resolution, review the **/etc/systemd/resolved.conf** file and ensure that the following settings are correct in this file:

- **DNS:** The list of DNS servers that provide name resolution.
- **Domains:** The list of domains that can be used for non-fully qualified hostnames. (For example, **ping test** would be converted to **ping test.example.com**.)

ExamAlert

There are other settings in the **/etc/systemd/resolved.conf** file, but the ones described here are the critical ones for answering troubleshooting questions on the Linux+ XK0-005 exam.

Also see the "**systemd**" section of Chapter 5 to learn about the **hostnamectl** and **resolvectl** commands.

Application Crash

If a **systemd**-managed application tends to crash, then you can use the following highlighted settings to restart it after a crash:

```
[Service]
Type=simple
ExecStart=/path/to/executable
RemainAfterExit=no
Restart=on-failure
RestartSec=5s
```

This should restart the application five seconds after it crashes.

Time-zone Configuration

Use the **timedatectl** command to display the system clock, including the time zone.

Syntax:

```
timedatectl [option] [value]
```

Example:

```
[root@OCS ~]# timedatectl
       Local time   :       Wed 2018-10-10 14:41:41 PDT
       Universal time:      Wed 2018-10-10 21:41:41 UTC
       RTC time:            Wed 2018-10-10 09:51:09
      Timezone:             America/Los_Angeles (PDT, -0700)
      NTP enabled:          yes
       NTP synchronized:     yes
       RTC in local TZ:      no
       DST active:           yes
        Last DST change:    DST began at
                            Sun 2018-03-11 01:59:59 PST
                            Sun 2018-03-11 03:00:00 PDT
```

```
      Next DST change:  DST ends (the clock jumps one hour backwards)
at
                                Sun 2018-11-04 01:59:59 PDT
                                 Sun 2018-11-04 01:00:00 PST
```

As the root user, you can use this command to set the system clock. Table 22.2 demonstrates the most commonly used methods of changing the system clock.

TABLE 22.2 **Methods of Changing the System Clock**

Method	Description
set-time [*time*]	Sets the system clock to the specified *time*.
set-timezone [*zone*]	Sets the system time zone to the specified *zone*.
set-ntp [0|1]	Enables (**1**) or disables (**0**) Network Time Protocol.

Boot Issues

To determine what **systemd** issues were encountered during a boot process, use the **journalctl -b** command, which provides the following output:

```
-- Logs begin at Sat 2018-02-24 16:42:01 PST, end at Sun 2022-05-01
14:28:53 PDT
Nov 14 10:30:16  example.com postfix/smtpd[2863]: warning: hostname
Nov 14 10:30:16  example.com postfix/smtpd[2863]: connect from unkno
Nov 14 10:30:16  example.com sshd[2873]: Invalid user localadmin fro
Nov 14 10:30:16  example.com sshd[2873]: input_userauth_request: inv
Nov 14 10:30:16  example.com sshd[2873]: pam_unix(sshd:auth): check
Nov 14 10:30:16  example.com sshd[2873]: pam_unix(sshd:auth): authen
Nov 14 10:30:17  example.com saslauthd[501]: pam_unix(smtp:auth): ch
Nov 14 10:30:17  example.com saslauthd[501]: pam_unix(smtp:auth): au
Nov 14 10:30:18  example.com sshd[2873]: Failed password for invalid
Nov 14 10:30:18  example.com saslauthd[501]: DEBUG: auth_pam: pam_au
Nov 14 10:30:18  example.com saslauthd[501]: do_auth         : auth
Nov 14 10:30:18  example.com postfix/smtpd[2863]: warning: unknown[8
Nov 14 10:30:18  example.com sshd[2873]: Received disconnect from 19
Nov 14 10:30:44  example.com sshd[2883]: pam_unix(sshd:auth): authen
Nov 14 10:30:47  example.com sshd[2883]: Failed password for root fr
Nov 14 10:30:47  example.com sshd[2883]: Received disconnect from 13
Nov 14 10:30:50  example.com postfix/smtpd[2885]: connect from mail-
Nov 14 10:30:50  example.com postfix/smtpd[2885]: Anonymous TLS conn
```

```
Nov 14 10:30:50  example.com postfix/trivial-rewrite[2887]: warning:
Nov 14 10:30:50  example.com postfix/smtpd[2885]: 84BFFEE03FF: clien
Nov 14 10:30:50  example.com postfix/cleanup[2888]: 84BFFEE03FF: mes
```

Each line provides a timestamp, the hostname, the process that is started (for example, **postfix/cleanup**) along with the process ID (for example, **2888**), and a description that can be used to learn where problems might exist.

By default, only the journal message for the most recent boot is displayed. You can see which previous boot logs exist by executing the following command:

```
root@ocs ~]$ journalctl --list-boots
-3 9cd60f76c27044ff951c5a28a1e4056d Sat 2018-02-24 16:42:01 PST—Sat
2018-02-24 1
-2 1430ae61899447c2925c084019a49eca Fri 2018-06-22 03:46:28 PDT—Fri
2018-06-22 0
-1 3276e666c8d647da8a2befe13c31dfa6 Thu 2020-05-21 19:20:05 PDT—Thu
2020-05-21 1
 0 77abb0fc787343c4bc85099d3e15048e Sun 2021-11-14 10:30:16 PST—Sun
2022-05-01 1
```

Then you can use the unique identifier to view the boot message of a previous boot attempt:

```
root@ocs ~]$ journalctl -b 3276e666c8d647da8a2befe13c31dfa6
{output omitted}
```

Journal Issues

On modern Linux systems, the logging process is handled by the **systemd-journald** service. To query **systemd** log entries, use the **journalctl** command:

```
[root@OCS ~]# journalctl | head
-- Logs begin at Tue 2017-01-24 13:43:18 PST, end
 at Sat 2017-03-04 16:00:32 PST. --
Jan 24 13:43:18 localhost.localdomain
 systemd-journal[88]: Runtime journal is using 8.0M
 (max allowed 194.4M, trying to leave 291.7M free
 of 1.8G available → current limit 194.4M).
Jan 24 13:43:18 localhost.localdomain
 systemd-journal[88]: Runtime journal is using 8.0M
```

```
 (max allowed 194.4M, trying to leave 291.7M free of 1.8G
 available → current limit 194.4M).
Jan 24 13:43:18 localhost.localdomain kernel:
 Initializing cgroup subsys cpuset
Jan 24 13:43:18 localhost.localdomain kernel:
 Initializing cgroup subsys cpu
Jan 24 13:43:18 localhost.localdomain kernel:
 Initializing cgroup subsys cpuacct
Jan 24 13:43:18 localhost.localdomain kernel:
 Linux version 3.10.0-327.18.2.el7.x86_64
    (builder@kbuilder.dev.centos.org) (gcc version
 4.8.3 20140911 (Red Hat 4.8.3-9) (GCC) ) #1 SMP
 Thu May 12 11:03:55 UTC 2016
Jan 24 13:43:18 localhost.localdomain kernel: Command
 line: BOOT_IMAGE=/vmlinuz-3.10.0-327.18.2.
 el7.x86_64 root=/dev/mapper/centos-root ro
 rd.lvm.lv=centos/root rd.lvm.lv=centos/swap
 crashkernel=auto rhgb quiet LANG=en_US.UTF-8
Jan 24 13:43:18 localhost.localdomain kernel:
 e820: BIOS-provided physical RAM map:
Jan 24 13:43:18 localhost.localdomain kernel:
 BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
```

Table 22.3 describes some important options for the **journalctl** command.

TABLE 22.3 **journalctl Command Options**

Option	Description
--all or **-a**	Shows all fields in full format.
-r	Reverses the log order so newest entries are displayed first.
-k	Shows only kernel messages.
--priority=*value*	Shows only messages that match the priority *value* (**emerg**, **alert**, **crit**, **err**, **warning**, **notice**, **info**, or **debug**).

The **/etc/systemd/journald.conf** file is used to configure the **systemd-journald** service. Typically, this file contains all commented-out values by default. Table 22.4 describes some important settings for the **/etc/systemd/journald.conf** file.

TABLE 22.4 **/etc/systemd/journald.conf File Settings**

Setting	Description
Storage=*value*	Indicates where to store the journal date; *value* can be **volatile**, **persistent**, **auto**, or **none**.
compress=[1|0]	If set to **1** (true), indicates compression of journal entries before writing to the file.

The **/var/log/journal** directory is where the **systemd-journald** service stores journal entries if the **Storage=persistent** setting is placed in the **/etc/systemd/journald.conf** file.

Services Not Starting on Time

Services failing to start in a timely manner can have an impact on the system booting. For example, the network service won't start until the firewall service has started. If the firewall service doesn't start in a timely manner, this can either cause the network to not start during boot or result in a very long boot process.

The following are some suggestions for troubleshooting when services fail to start in a timely manner:

- Review the configuration of the service. Each service normally has a configuration file in the **/etc** directory or some directory under the **/etc** directory. Review the configuration file for any recent changes.
- If updates have been applied lately, consider reverting to an older version of the software and testing to see if the problem still occurs.
- Review the output of **journalctl -b** for a message from the service that may indicate what the problem is.
- Review the service unit configuration to determine if there is an error in the way the service is starting up. Manual changes to these files may be the cause of the problem.
- Review services that are related to the service that isn't starting up in a timely manner.
- Review services that may be in conflict with the service that isn't starting up in a timely manner.

Cram Quiz

Answer these questions. The answers follow the last question. If you cannot answer these questions correctly, consider reading this chapter again until you can.

1. Which of the following directories contains the **systemd** service unit files?
   - ❍ **A. /lib/system/systemd**
   - ❍ **B. /etc/systemd/system**
   - ❍ **C. /lib/systemd/system**
   - ❍ **D. /etc/system/systemd**

2. Which of the following is an invalid **OnCalander** setting?
   - ❍ **A. OnCalendar=Sun 2022-05-01 13:30:00**
   - ❍ **B. OnCalendar=Sun 2022-05-31 13:30:00**
   - ❍ **C. OnCalendar=Sun 2022-05-01 13:30:45**
   - ❍ **D. OnCalendar=Sun 2022-13-01 13:30:00**

3. Which command displays information such as a storage device's UUID?
   - ❍ **A. uuid --show**
   - ❍ **B. lsblk -f**
   - ❍ **C. fdisk -uuid**
   - ❍ **D. show-uuid**

4. Which service unit file setting is used to specify the user account to which a process should be executed?
   - ❍ **A. Account**
   - ❍ **B. UserAcct**
   - ❍ **C. RunAs**
   - ❍ **D. User**

Cram Quiz Answers

1. **C.** You can see a list of the services that a system manages by viewing the files that end in **.service** in the **/lib/systemd/system** directory. The other answers are not valid files for **systemd**.
2. **D. OnCalendar=Sun 2022-13-01 13:30:00** is incorrect because of the value **13** in the month position of the time. The highest possible value for this field is **12**, which represents December.

3. **C.** If you don't know the UUID of a device, you can use the **lsblk -f** command to display all the UUIDs for all devices. The rest of the answers are not valid commands.
4. **D.** The **User** setting indicates which user account should be used to start a service. The other answers have no meaning in a service unit file.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

This page intentionally left blank

To receive your 10% off Exam Voucher, register your product at:

www.pearsonitcertification.com/register

and follow the instructions.

Pearson

Where are the companion content files?

Register this digital version of CompTIA® Linux+ XK0-005 Exam Cram to access important downloads.

Register this eBook to unlock the companion files. Follow these steps:

1. Go to **pearsonITcertification.com/account** and log in or create a new account.
2. Enter the ISBN: **9780137898558** (NOTE: Please enter the print book ISBN provided to register the eBook you purchased.)
3. Answer the challenge question as proof of purchase.
4. Click on the "Access Bonus Content" link in the Registered Products section of your account page, to be taken to the page where your downloadable content is available.

This eBook version of the print title does not contain the practice test software that accompanies the print book.

You May Also Like—Premium Edition eBook and Practice Test. To learn about the Premium Edition eBook and Practice Test series, visit **pearsonITcertification.com/practicetest**

The Professional and Personal Technology Brands of Pearson

Addison Wesley · Cisco Press · informIT · PEARSON IT Certification · QUE · SAMS