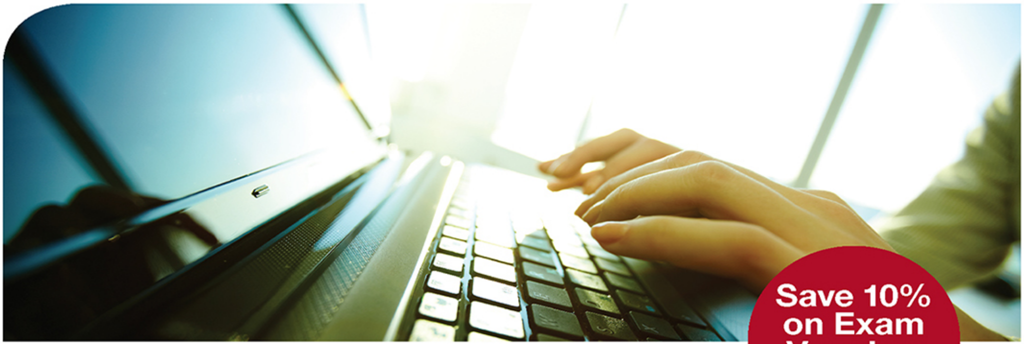

Save 10% on Exam Voucher

See Inside

CompTIA® Linux+™ Portable Command Guide

All the commands for the CompTIA XK0-004 exam in one compact, portable resource

PEARSON IT CERTIFICATION

SECOND EDITION

WILLIAM "BO" ROTHWELL

What Do You Want to Do Today?

(Continues on the inside back cover)

CompTIA® Linux+™ Portable Command Guide

All the commands for the CompTIA XK0-004 exam in one compact, portable resource

William "Bo" Rothwell

Pearson

CompTIA® Linux+™ Portable Command Guide

All the commands for the CompTIA XK0-004 exam in one compact, portable resource

William "Bo" Rothwell

Published by:
Pearson IT Certification
221 River Street
Hoboken, NJ 07030 USA

ScoutAutomatedPrintCode

Library of Congress Control Number: 2019908202

ISBN-13: 978-0-13-559184-0

ISBN-10: 0-13-559184-8

Warning and Disclaimer

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

The information is provided on an "as is" basis. The authors and Pearson shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the discs or programs that may accompany it.

The opinions expressed in this book belong to the author and are not necessarily those of Pearson.

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson 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.

Editor-in-Chief
Mark Taub

Product Line Manager
Brett Bartow

Executive Editor
Mary Beth Ray

Acquisitions Editor
Paul Carlstroem

Development Editor
Christopher Cleveland

Managing Editor
Sandra Schroeder

Project Editor
Mandie Frank

Copy Editor
Bart Reed

Technical Editor
Casey Boyles

Editorial Assistant
Cindy Teeters

Designer
Chuti Prasertsith

Composition
codeMantra

Indexer
Ken Johnson

Proofreader
Karen Davis

Credits

Figure 1-2	Screenshot of GNU GRUB © 2010-2018 Free Software Foundation, Inc.
Figure 1-3	Screenshot of GNU GRUB © 2010-2018 Free Software Foundation, Inc.
Figure 3-1	Screenshot of NMTUI interface © 2005–2014 The GNOME Project
Figure 3-2	Screenshot of NMTUI interface © 2005–2014 The GNOME Project
Figure 7-1	Screenshot of Aptitude Utility © The Distro Tracker Developers
Figure 9-1	Screenshot of GNU nano © Chris Allegretta
Figure 9-2	Screenshot of vi Editor © Bill Joy
Figure 21-1	Screenshot of Iftop © Paul Warren
Figure 21-2	Screenshot of Wireshark © Wireshark Foundation
Figure 21-3	Screenshot of mtr © BitWizard
Figure 22-1	Screenshot of Top © William LeFebvre
Cover	Pressmaster/Shutterstock

This page intentionally left blank

Contents at a Glance

Table of Contents

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 immediately dismantled it and held his first computer class, showing his friends what made this "computer thing" work.

Since this 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, 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.

About the Technical Reviewer

Casey Boyles started working in the IT field more than 27 years ago and quickly moved on to distributed application and database development. Casey later moved on to technical training and course development, where he specializes in full stack Internet application development, database architecture, and systems security. Casey typically spends his time smoking cigars while "reading stuff and writing stuff."

Dedications

As always, I owe more gratitude than I could ever provide to those in my life who support me the most: Sarah, Julia, Mom, and Dad.

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 Mary Beth Ray. I'm very sad that this will be our last book together. I have enjoyed working with you the last few years. Good fortune in your new journey in life. :)

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

Introduction

In 2019, COMPTIA released a new version of the Linux+ certification exam. This new exam isn't just an update from the last certification, but really a completely new exam.

The previous Linux+ certification was titled "CompTIA Linux+ powered by LPI." However, the new certification no longer involves LPI (the Linux Professional Institute). As a result, the format and topics are significantly different.

For example, the new certification only requires passing one exam, not two. Additionally, the exam objectives have undergone a major rewrite, and you will see several DevOps-based topics not included in the previous certification.

Additionally, you should be prepared for a handful of scenario questions that you will be asked to answer based on a situation described to you. However, most of the exam will be multiple choice, much like the previous exams.

Use this book as a reference to all the key exam-testable topics. This book makes for an excellent roadmap on your journey to learning Linux and passing the Linux+ certification exam.

Good luck!

—William "Bo" Rothwell
May 18, 2019

Who Should Read This Book

This book is for those people preparing for the CompTIA Linux+ certification exam, whether through self-study, on-the-job training and practice, or study via a training program. The book provides you with the depth of knowledge to help you pass the exam as well as introduces valuable features of the Linux operating system.

Command Syntax Conventions

The conventions used to present command syntax in this book are as follows:

- **Boldface** indicates commands and keywords that are entered literally as shown. In actual configuration examples and output (not general command syntax), boldface indicates commands that are manually input by the user (such as a **show** command).
- *Italic* indicates arguments for which you supply actual values.
- Vertical bars (|) separate alternative, mutually exclusive elements.
- Square brackets ([ ]) indicate an optional element.
- Braces ({ }) indicate a required choice.
- Braces within brackets ([{ }]) indicate a required choice within an optional element.

Organization of This Book

Because this book is designed to help you prepare for the CompTIA Linux+ certification exam, I have opted to match the organization of the book to align with the exam objective topics. There are 27 topics for the exam, and those topics match with each of the 27 chapters in this book. This organization should help aid you in your preparation for the exam.

Here is the layout for each section of this book:

Part I: Hardware and System Configuration

- **Chapter 1, "Explain Linux boot process concepts"**—You will learn the process of how the system behaves during the boot process and how the administrator can modify this behavior in this chapter.
- **Chapter 2, "Given a scenario, install, configure, and monitor kernel modules"**—In this chapter, you will explore kernel modules, which are small programs that add to the functionality of the kernel.
- **Chapter 3, "Given a scenario, configure and verify network connection parameters"**—You will learn how to set up network interfaces in this chapter.
- **Chapter 4, "Given a scenario, manage storage in a Linux environment"**—In this chapter, you learn how to create partitions and filesystems using utilities like fdisk and mkfs.
- **Chapter 5, "Compare and contrast cloud and virtualization concepts and technologies"**—The focus of this chapter is the core concepts of virtual OS instances, both on a local system and in the cloud.
- **Chapter 6, "Given a scenario, configure localization options"**—In this chapter, you will discover how to modify the system to behave differently in different locations throughout the world.

Part II: Systems Operation and Maintenance

- **Chapter 7, "Given a scenario, conduct software installations, configurations, updates, and removals"**—You will learn in this chapter how to manage software packages.
- **Chapter 8, "Given a scenario, manage users and groups"**—This chapter focuses on utilities that allow you to add, modify, and delete user and group accounts.
- **Chapter 9, "Given a scenario, create, modify, and redirect files"**—Learn how to create files using commonly used Linux editors as well as sending the output of commands into files.
- **Chapter 10, "Given a scenario, manage services"**—The focus of this chapter is services, which are programs that run on the local system and provide some sort of feature or function to either the local system or remote systems.
- **Chapter 11, "Summarize and explain server roles**—The understanding of servers and the role they play on Linux systems are explored in this chapter.

- **Chapter 12, "Given a scenario, automate and schedule jobs"**—Learn the process of scheduling programs and applications to run at future times.
- **Chapter 13, "Explain the use and operation of Linux devices"**—Learn how to manage devices, such as USB devices, memory, and physical storage devices.
- **Chapter 14, "Compare and contrast Linux graphical user interfaces"**—Explore the wide variety of graphical user interfaces (GUIs) in this chapter.

Part III: Security

- **Chapter 15, "Given a scenario, apply or acquire the appropriate user and/or group permissions and ownership"**—Building on the concepts you learned in Chapter 8, this chapter will focus on how to manage permissions that provide access to files and directories.
- **Chapter 16, "Given a scenario, configure and implement appropriate access and authentication methods"**—This chapter explores the different authentication methods, a feature that allows users to gain access to the system.
- **Chapter 17, "Summarize security best practices in a Linux environment"**—This chapter provides a high-level overview of important security best practices to implement on Linux systems.
- **Chapter 18, "Given a scenario, implement logging services"**—Learn how system logging is configured in this chapter.
- **Chapter 19, "Given a scenario, implement and configure Linux firewalls"**—The focus of this chapter is the concept and configuration of firewalls on Linux systems.
- **Chapter 20, "Given a scenario, backup, restore, and compress files"**—Protect against data loss by learning how to create backups in this chapter.

Part IV: Linux Troubleshooting and Diagnostics

- **Chapter 21, "Given a scenario, analyze system properties and remediate accordingly"**—This chapter explores the essentials of troubleshooting common problems on Linux systems.
- **Chapter 22, "Given a scenario, analyze system processes in order to optimize performance"**—The focus of this chapter is tools and concepts related to determining if a Linux system is performing at an optimal level.
- **Chapter 23, "Given a scenario, analyze and troubleshoot user issues"**—Learn how to troubleshoot common problems related to user accounts in this chapter.
- **Chapter 24, "Given a scenario, analyze and troubleshoot application and hardware issues"**—This chapter provides you with the concepts and tools needed to discover and resolve issues related to system hardware and applications.

Part V: Automation and Scripting

- **Chapter 25, "Given a scenario, deploy and execute basic BASH scripts"**—Shell scripts, a collection of shell commands, are covered in this chapter.
- **Chapter 26, "Given a scenario, carry out version control using Git"**—In this chapter, you will learn how to manage a Git repository.
- **Chapter 27, "Summarize orchestration processes and concepts"**—In this chapter, you will explore orchestration, which includes the process of quickly and effectively installing and configuring systems.

CHAPTER 15

Given a scenario, apply or acquire the appropriate user and/or group permissions and ownership

This chapter provides information and commands concerning the following topics:

File and directory permissions

- Read, write, execute
- User, group, other
- SUID
- Octal notation
- umask
- Sticky bit
- GUID
- Inheritance
- Utilities (chmod, chown, chgrp, getfacl, setfacl, ls, ulimit, chage)

Context-based permissions

- SELinux configurations (disabled, permissive, enforcing)
- SELinux policy (targeted)
- SELinux tools (setenforce, getenforce, sestatus, setsebool, getsebool, chcon, restorecon, ls -Z, ps -Z)
- AppArmor (aa-disable, aa-complain, aa-unconfined, /etc/apparmor.d/, /etc/apparmor.d/tunables)

Privilege escalation

- su
- sudo
- wheel
- visudo
- sudoedit

User types

- Root
- Standard
- Service

File and Directory Permissions

The user who owns a file or directory has the ability to allow or deny access to the file or directory using permissions. Additionally, the root user has the ability to change the permissions of any file or directory on the system. This section focuses on these permissions and how to apply them.

Read, Write, Execute

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 will protect his 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 ten characters of the output denote the file type (recall that a hyphen [–] as the character in the first position denotes a plain file, whereas a **d** denotes a directory) and the permissions for the file. 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 (symbolized by **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 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, they mean the following:

- **Read**: Can list files in the directory.
- **Write**: Can add and delete files in the directory (requires execute permission).
- **Execute**: Can "**cd**" into the directory or use it in a pathname.

User, Group, Other

See the "Read, Write, Execute" subsection in this chapter.

SUID

The following table describes the special permission sets of **suid**, **sgid**, and the sticky bit:

	suid	**sgid**	**Sticky Bit**
Description	When set on executable files, **suid** allows a program to access files using the permissions of the user owner of the executable file.	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 the directory inherit the group ownership of the directory.	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.
Set	**chmod u+s *file*** or **chmod 4*xxx* *file*** (*xxx* refers to regular read, write, and execute permissions.)	**chmod g+s *file*** or **chmod 2*xxx* *file*** (*xxx* refers to regular read, write, and execute permissions.)	**chmod o+t *file*** or **chmod 1*xxx* *file*** (*xxx* refers to regular read, write, and execute permissions). Note: Sticky bit permissions are almost always set to the octal value of 1777.
Remove	**chmod u-s *file*** or **chmod 0*xxx* *file***	**chmod g-s *file*** or **chmod 0*xxx* *file***	**chmod o-t *file*** or **chmod 0*xxx* *file***

See the "chmod" subsection in this chapter for details about the **chmod** command.

Octal Notation

See the "chmod" subsection in this chapter for details about octal notation.

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 the following table:

Type	Maximum Possible Permission for New Item
File	**rw-rw-rw-**
Directory	**rwxrwxrwx**

Figure 15.1 describes how a umask value of **027** would apply to new files versus how it would apply to new directories.

Description	File			Directories		
Maximum	rw-	r**w**-	**rw**-	rwx	r**w**x	**rw**x
Umask Applied	---	-**M**-	**MM**-	---	-**M**-	**MM**-
Result	rw-	r--	---	rwx	r-x	--x

Figure 15-1 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.

Sticky Bit

See the "SUID" subsection in this chapter for details about the sticky bit.

GUID

See the "SUID" subsection in this chapter for details about GUID.

Inheritance

Unlike in some operating systems, basic and advanced Linux permissions don't utilize inheritance. The idea of inheritance is when a new file or directory inherits the permissions from the directory that the item is created in.

There is a way to have permissions inherit from the parent directory: by using ACLs (access control lists). See the "setfacl" subsection for details.

Utilities

Several utilities or commands allow you to manage permissions. The utilities that are testable for the Linux+ exam are covered in this section.

chmod

The **chmod** command is used to change permissions on files. It can be used in two ways: symbolic method and octal method. With the octal method, the 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
```

The following table demonstrates some examples using the octal method:

Example	Description
chmod 755 file	Sets the permissions of **rwxr-xr-x**.
chmod 511 file	Sets the permissions of **r-x--x--x**.
chmod 600 file	Sets the permissions of **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, use the symbolic method by passing three values to the **chmod** command, as shown in the following table:

Who	What	Permission
u = user owner	+	**r**
g = group owner	-	**w**
o = other	=	**x**
a = all sets		

The following demonstrates adding 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
```

Here are some important options for the **chmod** command:

Option	Description
-R	Recursively apply changes to an entire directory structure.
-v	Verbose. Produce output demonstrating the changes that are made.

chown

The **chown** command is used to change the user owner or group owner of a file or directory. The following table demonstrates different ways to use this command:

Example	Description
chown tim abc.txt	Changes the user owner of the **abc.txt** file to tim user.
chown tim:staff abc.txt	Changes the user owner of the **abc.txt** file to 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 command must own the file and be a member of the group that the ownership is being changed to.

Here are some important options for the **chown** command:

Option	Description
-R	Recursively apply changes to an entire directory structure.
--reference=*file*	Change the user and group owner to the ownership of *file*.
-v	Verbose. Produce output demonstrating the changes that are made.

chgrp

The **chgrp** command is designed to change the group ownership of a file. The syntax of this command is **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 command must own the file and be a member of the group that the ownership is being changed to.

Here are some important options for the **chgrp** command:

Option	Description
-R	Recursively apply changes to an entire directory structure.
--reference=*file*	Change the user and group owner to the ownership of *file*.
-v	Verbose. Produce output demonstrating the changes that are made.

getfacl

See the "setfacl" subsection next.

setfacl

ACLs (access control lists) allow 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 will be explained later in this section.)

The value for *who* will be 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 will appear 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 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 will create 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 will inherit the ACLs that were created in 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--
```

Here are some important options for the **setfacl** command:

Option	Description
-b	Remove all ACLs.
-d	Set a default ACL on a directory; this will be inherited by any new file or directory created with this directory.
-R	Apply recursively.

ls

See the “Read, Write, Execute” subsection in this chapter to see how the **ls** command is important for displaying permissions.

ulimit

The **ulimit** command lists or sets a user’s account limits:

```
[root@OCS ~]# ulimit -a
core file size                 (blocks, -c)   0
data seg size                  (kbytes,   -d)   unlimited
scheduling priority            (-e)    0
file size                      (blocks, -f)   unlimited
```

```
pending signals                 (-i)     15439
max locked memory               (kbytes, -l)    64
max memory size                 (kbytes, -m)      unlimited
open files                      (-n)       1024
pipe size                       (512 bytes, -p)    8
POSIX message queues            (bytes, -q)    819200
real-time priority              (-r)    0
stack size                      (kbytes, -s)    8192
cpu time                        (seconds, -t)   unlimited
max user processes              (-u)   4096
virtual memory                  (kbytes, -v)   unlimited
file locks                      (-x)   unlimited
```

These limits are normally configured by the system administrator using a PAM (Pluggable Authentication Modules) configuration file:

```
[root@OCS ~]# tail -n 12 /etc/security/limits.conf
#<domain>          <type>            <item>              <value>
#

#*                 soft              core                0
#*                 hard              rss                 10000
#@student          hard              nproc               20
#@faculty          soft              nproc               20
#@faculty          hard              nproc               50
#ftp               hard              nproc               0
# End of file
```

For example, you may want to limit how many concurrent logins an account can have:

```
student             -             maxlogins            4
```

Users rarely use the **ulimit** command to limit their own account, so the options for this command are not as important as understanding what the output displays. Additionally, some of the limits are very rarely used. The commonly used limits are described in the following table:

Limit	Description
fsize	Maximum file size allowed in memory
cpu	Maximum CPU time allowed
nproc	Maximum number of concurrently running processes
maxlogins	Maximum number of concurrent logins

chage

See the “chage” section in Chapter 8, “Given a scenario, manage users and groups.”

Context-Based Permissions

Files and directories may be compromised by users who either do not understand permissions or accidently provide more access than intended. This is a reflection of an old system administration saying, "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 utilize Mandatory Access Control (MAC). However, when a context-based solution is enabled, DAC still applies (both MAC and DAC are enforced).

Context-based permissions can be configured to accommodate for this flaw by providing an additional level of security when processes (programs) are used to access files. This section covers two commonly used context-based methods: SELinux and AppArmor.

SELinux Configurations

An SELinux security policy can be applied that will require processes to be a part of an SELinux security context (think "security group") in order to be able to access files and directories. Regular permissions will still be used to further define access, but for accessing the file/directory, this 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 a security risk. Consider a situation where you have many active processes that provide a variety of services. For example, one of these processes may 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
```

Note that in the preceding output, 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 the preceding command, all users have the ability to view the contents of the **/etc/passwd** file. Ask yourself this: Do you want some random person (usually called 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 can only access a certain set of files. This is what most administrators use SELinux for: to secure processes that may be compromised by hackers making use of known (or, perhaps, unknown) exploits.

This subsection covers the essentials of managing an SELinux security policy.

disabled

When in disabled mode, SELinux is not functional at all. No checks are performed when users attempt to access files or directories. See the "setenforce" and "getenforce" subsections in this chapter for more details on viewing and changing the SELinux mode.

permissive

When in permissive mode, SELinux performs checks but will never block access to a file or directory. This mode is designed for troubleshooting problems as log messages are created when in this mode. See the "setenforce" and "getenforce" subsections in this chapter for more details on viewing and changing the SELinux mode.

enforcing

When in enforcing mode, SELinux performs checks and will block access to files or directories if necessary. See the "setenforce" and "getenforce" subsections in this chapter for more details on viewing and changing the SELinux mode.

SELinux Policy

An SELinux policy is a collection of rules that determine what restrictions are imposed by the policy. The policy itself is often very complex, and details are beyond the scope of the Linux+ exam. It is, however, important to know that the policy sets the 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 enabled.

A targeted policy contains rules designed to protect the system from services, rather than 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.

targeted

See the "SELinux Policy" subsection in this chapter.

SELinux Tools

A large number of tools are used to manage SELinux. This subsection covers the tools you should know for the Linux+ exam.

setenforce

You can disable the security policy (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 will not block any access to files and directories, but warnings will be 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 "disabled," "permissive," and "enforcing" subsections in this chapter for more details regarding SELinux modes.

sestatus

The **sestatus** command provides overall status information about SELinux:

```
root@OCS:~# sestatus
SELinux status:                  enabled
SELinuxfs mount:                 /sys/fs/selinux
SELinux root directory:          /etc/selinux
Loaded policy name:              targeted
Current mode:                    enforcing
Mode from config file:           enforcing
Policy MLS status:               enabled
Policy deny_unknown status:      allowed
Max kernel policy version:       28
```

setsebool

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 "getsebool" subsection next for information about Boolean values.

getsebool

Part of an SELinux security policy includes Booleans. A Boolean is a setting that can be assigned either a true or a false value. This value can affect the behavior of the SELinux policy.

```
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
```

In order 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 “setsebool” subsection in this chapter for information on how to set a Boolean value.

chcon

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
```

See the “ls -Z” subsection in this chapter for more details regarding security contexts.

restorecon

There are SELinux rules that 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 performs the changes recursively on a directory structure.

See the “ls -Z” subsection in this chapter for more details regarding security contexts.

ls -Z

Each process runs with a security context. To see this, use the **-Z** option to the **ps** command (the **head** command is used here simply to limit the output of the 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
```

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 should 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 (note that the SELinux context contains so much data that the filename cannot fit on the same line):

```
root@OCS:~# ls -Z /var/www/html/index.html
unconfined_u:object_r:httpd_sys_content_t:s0/var/www/html/index.html
```

ps -Z

See the “ls -Z” subsection in this chapter.

AppArmor

AppArmor is a MAC system that plays a similar role to SELinux in that it provides a context-based permission model. This subsection describes the key components of AppArmor that are exam testable.

aa-disable

An AppArmor profile is a rule set that describes how AppArmor should restrict a process. A profile can be disabled for a specific profile by using the **aa-disable** command. Here’s an example:

```
root@OCS:~# ln -s /etc/apparmor.d/usr.sbin.mysqld
  /etc/apparmor.d/disable
root@OCS:~#  apparmor_parser -R /etc/apparmor.d/usr.sbin.mysqld
```

NOTE: To view the status of a profile, use the **aa-status** command. To enable a profile again, use the following commands:

```
root@OCS:~# rm /etc/apparmor.d/disable/usr.sbin.mysqld
root@OCS:~#  apparmor_parser -r
  /etc/apparmor.d/usr.sbin.mysqld
```

aa-complain

If you need to troubleshoot an AppArmor profile, it is best to put it into complain mode. In this mode, there are no restrictions enforced, but any problems will be 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, use the following command:

```
root@OCS:~#  sudo aa-enforce /usr/sbin/mysqld
Setting /usr/sbin/mysqld to enforce mode
```

aa-unconfined

Use the **aa-unconfined** command to list processes that are not restricted by the AppArmor profiles.

/etc/apparmor.d/

The **/etc/apparmor.d** directory is the location of the definitions of the AppArmor profiles. Note that knowing how to create or read these files is beyond the scope of the Linux+ 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.

/etc/apparmor.d/tunables

The **/etc/apparmor.d/tunables** directory is the location of files that can be used to fine-tune the behavior of AppArmor. Note that knowing how to create or read these files is beyond the scope of the Linux+ exam.

Privilege Escalation

The concept behind privilege escalation is that a user may need 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. There are several techniques that can provide privilege access; this subsection covers the techniques that are exam testable.

su

The **su** command allows a user to shift user accounts:

```
[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 will be provided. When you're not using the **-** character, a non-login shell will be provided.

sudo

When 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 will be prompted for your own password and, if the settings in the **/etc/sudoers** file are correct, the command will execute correctly. If the settings are not correct, an error message will appear.

The following table describes common options for the **sudo** command:

Option	Description
-b	Run the command in the background.
-e	Run like the **sudoedit** command. See the "sudoedit" subsection in this chapter.
-l	List which commands are allowed for this user.
-u *user*	Run the command as *user* rather than as the root user.

Also see the "visudo" section in this chapter for details regarding the **/etc/sudoers** file.

wheel

A common method for providing non-root users with root access is to use the wheel group. If enabled in the **/etc/sudoers** file (normally this line is "commented out"), anyone in the wheel group will have the ability to run any command as the root user via the **sudo** command:

```
%wheel    ALL=(ALL)     ALL
```

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 should use the **visudo** command rather than edit the file directly.

The following table describes important definitions for the **/etc/sudoers** file:

Option	Description
User_Alias	A name that represents a group of users (for example, **User_Alias ADMINS = julia, sarah**)
Cmnd_Alias	A name that represents a group of commands (for example, **Cmnd_Alias SOFTWARE = /bin/rpm, /usr/bin/yum**).

The format of an entry for the **/etc/sudoers** file uses the following syntax:

```
user               machine=commands
```

To allow the 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 the ability 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
```

sudoedit

If you want to edit a file using sudo access, consider using the **sudeoedit** or **sudo -e** command. Using this feature requires having the ability to edit a file using a command designed to edit files (such as **nano**, **vi**, or **vim**).

Example:

```
sudoedit file1
```

Note that the editor that will be chosen depends on variables. The following variables are consulted:

- SUDO_EDITOR
- VISUAL
- EDITOR

If none of these variables is set, then the vi editor is typically the default.

User Types

This section breaks down the different user types you are likely to encounter on Linux-based systems.

Root

The root account is the system administrator account. It is important to note that what makes the root account special is the UID of 0. Any user with a UID of 0 is a full system administrator. As a security note, when you're performing audits, look for any user with a UID of 0, as this is a common hacking technique.

Standard

Any account with a UID of 1000 or higher is considered a standard or regular user account. People are normally assigned standard user accounts so they can log in to the system and perform tasks.

Service

A typical Linux system will have many service user accounts. These service user accounts typically have UID values under 1000, 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).

Additionally, if you add new software to the 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.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

T

U

V

W

X

Y

Z