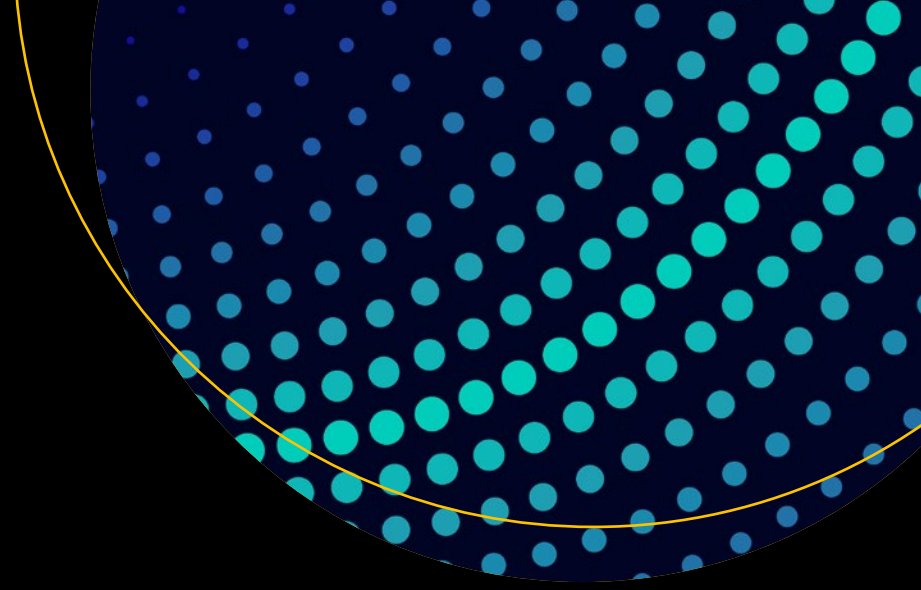

# Using and Administering Linux: Volume 2

Zero to SysAdmin: Advanced Topics

David Both

Apress®

# Using and Administering Linux: Volume 2

## Zero to SysAdmin: Advanced Topics

**David Both**

Apress®

*Using and Administering Linux: Volume 2*

David Both
Raleigh, NC, USA

ISBN-13 (pbk): 978-1-4842-5454-7 ISBN-13 (electronic): 978-1-4842-5455-4
https://doi.org/10.1007/978-1-4842-5455-4



Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484254547. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

*This book – this course – is dedicated to all Linux and open source course developers and trainers.*

:(){ :|:& };:

# Table of Contents

# About the Author

**David Both** is an open source software and GNU/Linux advocate, trainer, writer, and speaker. He has been working with Linux and open source software for more than 20 years and has been working with computers for over 45 years. He is a strong proponent of and evangelist for the "Linux Philosophy for System Administrators." David has been in the IT industry for over 40 years.

Mr. Both worked for IBM for 21 years and, while working as a Course Development Representative in Boca Raton, FL, in 1981, wrote the training course for the first IBM PC. He has taught RHCE classes for Red Hat and has worked at MCI WorldCom, Cisco, and the State of North Carolina. In most of the places he has worked since leaving IBM in 1995, he has taught classes on Linux ranging from Lunch'n'Learns to full 5-day courses. Helping others learn about Linux and open source software is one of his great pleasures.

David prefers to purchase the components and build his own computers from scratch to ensure that each new computer meets his exacting specifications. Building his own computers also means not having to pay the Microsoft tax. His latest build is an ASUS TUF X299 motherboard and an Intel i9 CPU with 16 cores (32 CPUs) and 64GB of RAM in a Thermaltake Core X9 case.

He has written articles for magazines including *Linux Magazine*, *Linux Journal*, and *OS/2* back when there was such a thing. His article "Complete Kickstart," co-authored with a colleague at Cisco, was ranked 9th in the *Linux Magazine* Top Ten Best System Administration Articles list for 2008. He currently writes prolifically and is a volunteer community moderator for Opensource.com. He particularly enjoys learning new things while researching his articles.

David currently lives in Raleigh, NC, with his very supportive wife and a strange rescue dog that is mostly Jack Russell. David also likes reading, travel, the beach, old M*A*S*H reruns, and spending time with his two children, their spouses, and four grandchildren.

David can be reached at LinuxGeek46@both.org or on Twitter @LinuxGeek46.

# About the Technical Reviewer

**Jason Baker** has been a Linux user since the early 2000s, ever since stuffing a Slackware box under his desk and trying to make the darn thing work. He is a writer and presenter on a variety of open source projects and technologies, many of which can be found on Opensource.com. A Red Hat Certified Systems Administrator, he is currently the managing editor of Enable SysAdmin, Red Hat's community publication for system administrators. When he's not at work, he enjoys tinkering with hardware and using open source tools to play with maps and other visualizations of cool data sets. He lives in Chapel Hill, NC, with his wife, Erin, and their rescue cat, Mary.

# Acknowledgments

Writing a book is not a solitary activity, and this massive three-volume Linux training course required a team effort so much more than most.

The most important person in this effort has been my awesome wife, Alice, who has been my head cheerleader and best friend throughout. I could not have done this without your support and love.

I am grateful for the support and guidance of Louise Corrigan, senior editor for open source at Apress, who believed in me and my vision for this book. This book would not have been possible without her.

To my coordinating editor, Nancy Chen, I owe many thanks for her hours of work, guidance, and being there to discuss many aspects of this book. As it grew and then continued to grow some more, our discussions were invaluable in helping to shape the final format of this work.

And to Jim Markham, my development editor, who quietly kept an eye and a guiding hand on the vast volume of material in these three volumes to ensure that the end result would meet the needs of you – my readers – and most importantly, you as the student.

Jason Baker, my intrepid technical reviewer, has done an outstanding job to ensure the technical accuracy of all three volumes of this course. Due to the major changes made in some parts of the course as its final form materialized, he retested some chapters in their entirety to help ensure that I had not screwed anything up. Jason also made important suggestions that have significantly enhanced the quality and scope of the entire three-volume work. These volumes are much better for his contributions. Jason's amazing work and important contributions to this book and the course of which it is part have helped to make it far better than it might have been.

Of course any remaining errors and omissions are my responsibility alone.

# Introduction

First, thank you for purchasing *Using and Administering Linux: Volume 2 – Zero to SysAdmin: Advanced Topics*. The Linux training course upon which you have embarked is significantly different from other training that you could purchase to learn about Linux.

## About this course

This Linux training course, *Using and Administering Linux – Zero to SysAdmin*, consists of three volumes. Each of these three volumes is closely connected and they build upon each other. For those new to Linux, it's best to start here with Volume 1, where you'll be guided through the creation of a virtual laboratory – a virtual network and a virtual machine – that will be used and modified by many of the experiments in all three volumes. More experienced Linux users can begin with later volumes and download the script that will set up the VM for the start of Volumes 2 and 3. Instructions provided with the script will provide specifications for configuration of the virtual network and the virtual machine.

Refer to the following volume overviews to select the volume of this course most appropriate for your current skill level.

This Linux training course differs from others because it is a complete self-study course. Newcomers should start at the beginning of Volume 1 and read the text, perform all of the experiments, and complete all of the chapter exercises through to the end of Volume 3. If you do this, even if you are starting from zero knowledge about Linux, you can learn the tasks necessary to becoming a Linux system administrator, a SysAdmin.

Another difference this course has over others is that all of the experiments are performed on one or more virtual machines (VMs) in a virtual network. Using the free software, VirtualBox, you will create this virtual environment on any reasonably sized host, whether Linux or Windows. In this virtual environment, you are free to experiment on your own, make mistakes that could damage the Linux installation of a hardware host, and still be able to recover completely by restoring the Linux VM host from any one of multiple snapshots. This flexibility to take risks and yet recover easily makes it possible to learn more than would otherwise be possible.

I have always found that I learn more from my mistakes than I ever have when things work as they are supposed to. For this reason I suggest that rather than immediately reverting to an earlier snapshot when you run into trouble, you try to figure out how the problem was created and how best to recover from it. If, after a reasonable period of time, you have not resolved the problem, that would be the point at which reverting to a snapshot would make sense.

Inside, each chapter has specific learning objectives, interactive experiments, and review exercises that include both hands-on experiments and some review questions. I learned this format when I worked as a course developer for IBM from 1978 through 1981. It is a tried and true format that works well for self-study.

These course materials can also be used as reference materials. I have used my previous course materials for reference for many years and they have been very useful in that role. I have kept this as one of my goals in this set of materials.

---

**Note** Not all of the review exercises in this course can be answered by simply reviewing the chapter content. For some questions you will need to design your own experiment in order to find a solution. In many cases there will very probably be multiple solutions, and all that produce the correct results will be the "correct" ones.

---

# Process

The process that goes with this format is just as important as the format of the course – really even more so. The first thing that a course developer must do is generate a list of requirements that define both the structure and the content of the course. Only then can the process of writing the course proceed. In fact, many times I find it helpful to write the review questions and exercises before I create the rest of the content. In many chapters of this course I have worked in this manner.

These courses present a complete, end-to-end Linux training course for students like you who know before you start that you want to learn to be a Linux system administrator – a SysAdmin. This Linux course will allow you to learn Linux right from the beginning with the objective of becoming a SysAdmin.

Many Linux training courses begin with the assumption that the first course a student should take is one designed to start them as users. Those courses may discuss the role of root in system administration, but ignore topics that are important to future SysAdmins. Other courses ignore system administration altogether. A typical second course will introduce the student to system administration, while a third may tackle advanced administration topics.

Frankly, this baby step approach did not work well for many of us who are now Linux SysAdmins. We became SysAdmins, in part at least, due to our intense desire – our deep need – to learn as much as possible as quickly as possible. It is also, I think in large part, due to our highly inquisitive natures. We learn a basic command and then start asking questions and experimenting with it to see what its limits are, what breaks it, and what using it can break. We explore the man(ual) pages and other documentation to learn the extreme usages to which it might be put. If things don't break by themselves, we break them intentionally to see how they work and to learn how to fix them. We relish our own failures because we learn more from fixing them than we do when things always work as they are supposed to.

In this course we will dive deep into Linux system administration almost from the very beginning. You will learn many of the Linux tools required to use and administer Linux workstations and servers – usually multiple tools that can be applied to each of these tasks. This course contains many experiments to provide you with the kind of hands-on experiences that SysAdmins appreciate. All of these experiments guide you one step at a time into the elegant and beautiful depths of the Linux experience. You will learn that Linux is simple and that simplicity is what makes it both elegant and knowable.

Based on my own years working with Unix and Linux, the course materials contained in these three volumes are designed to introduce you to the practical daily tasks you will perform as a Linux user and, at the same time, as a Linux system administrator – SysAdmin. But I do not know everything – that is just not possible – no SysAdmin does. Further, no two SysAdmins know exactly the same things because that too is impossible. We have each started with different knowledge and skills; we have different goals; we have different experiences because the systems on which we work have failed in different ways, had different hardware, were embedded in different networks, had different distributions installed, and many other differences. We use different tools and approaches to problem-solving because the many different mentors and teachers we had used different sets of tools from each other; we use different Linux distributions; we think differently; and we know different things about the hardware on which Linux runs. Our past is much of what makes us what we are and what defines us as SysAdmins.

So I will show you things in this course – things that I think are important for you to know – things that, in my opinion, will provide you with the skills to use your own curiosity and creativity to find solutions that I would never think of to problems I have never encountered.

## What this course is not

This course is not a certification study guide. It is not designed to help you pass a certification test of any type. This course is intended purely to help you become a good or perhaps even great SysAdmin, not to pass a test.

There are a few good certification tests. Red Hat and Cisco certifications are among the best because they are based on the test taker's ability to perform specific tasks. I am not familiar with any of the other certification tests because I have not taken them. But the courses you can take and books you can purchase to help you pass those tests are designed to help you pass the tests and not to administer a Linux host or network. That does not make them bad – just different from this course.

## Content overview

Because there are three volumes to this course and because I reference other chapters, some of which may be in other volumes, we need a method for specifying in which volume the referenced material exists. If the material is in another volume, I will always specify the volume number, that is, "Chapter 2 in Volume 3" or "Volume 2, Chapter 5." If the material is in the same volume as the reference to it, I may simply specify the chapter number; however, I may also reference the current volume number for clarity.

This quick overview of the contents of each volume should serve as a quick orientation guide if you need to locate specific information. If you are trying to decide whether to purchase this book and its companion volumes, it will give you a good overview of the entire course.

# Using and Administering Linux: Volume 1 Zero to SysAdmin: Getting Started

Volume 1 of this training course introduces operating systems in general and Linux in particular. It briefly explores *The Linux Philosophy for SysAdmins*[1] in preparation for the rest of the course.

Chapter 4 then guides you through the use of VirtualBox to create a virtual machine (VM) and a virtual network to use as a test laboratory for performing the many experiments that are used throughout the course. In Chapter 5, you will install the Xfce version of Fedora – a popular and powerful Linux distribution – on the VM. In Chapter 6 you will learn to use the Xfce desktop which will enable you to leverage your growing command-line interface (CLI) expertise as you proceed through the course.

Chapters 7 and 8 will get you started using the Linux command line and introduce you to some of the basic Linux commands and their capabilities. In Chapter 9 you will learn about data streams and the Linux tools used to manipulate them. And in Chapter 10 you will learn a bit about several text editors which are indispensable to advanced Linux users and system administrators.

Chapters 11 through 13 start your work as a SysAdmin and take you through some specific tasks such as installing software updates and new software. Chapters 14 and 15 discuss more terminal emulators and some advanced shell skills. In Chapter 16 you will learn about the sequence of events that take place as the computer boots and Linux starts up. Chapter 17 shows you how to configure your shell to personalize it in ways that can seriously enhance your command line efficiency.

Finally, Chapters 18 and 19 dive into all things file and filesystems:

1. Introduction
2. Introduction to operating systems
3. The Linux Philosophy for SysAdmins
4. Preparation
5. Installing Linux
6. Using the Xfce desktop
7. The Linux command line

[1]Both, David, *The Linux Philosophy for SysAdmins*, Apress, 2018

8. Core utilities
9. Data streams
10. Text editors
11. Working as root
12. Installing updates and new software
13. Tools for problem-solving
14. Terminal emulator mania
15. Advanced shell topics
16. Linux boot and startup
17. Shell configuration
18. Files, directories, and links
19. Filesystems

## Using and Administering Linux: Volume 2 Zero to SysAdmin: Advanced Topics

Volume 2 of *Using and Administering Linux* introduces you to some incredibly powerful and useful advanced topics that every SysAdmin must know.

In Chapters 1 and 2 you will experience an in-depth exploration of logical volume management – and what that even means – as well as the use of file managers to manipulate files and directories. Chapter 3 introduces the concept that, in Linux, everything is a file. You will also learn some fun and interesting uses of the fact that everything is a file.

In Chapter 4 you will learn to use several tools that enable the SysAdmin to manage and monitor running processes. Chapter 5 enables you to experience the power of the special filesystems, such as /proc, that enable us as SysAdmins to monitor and tune the kernel while it is running – without a reboot.

Chapter 6 will introduce you to regular expressions and the power that using them for pattern matching can bring to the command line, while Chapter 7 discusses managing printers and printing from the command line. In Chapter 8 you will use several tools to unlock the secrets of the hardware in which your Linux operating system is running.

Chapters 9 through 11 show you how to do some simple – and not so simple – command-line programming and how to automate various administrative tasks.

You will begin to learn the details of networking in Chapter 12, and Chapters 13 through 15 show you how to manage the many services that are required in a Linux system. You will also explore the underlying software that manages the hardware and can detect when hardware devices such as USB thumb drives are installed and how the system reacts to that.

Chapter 16 shows you how to use the logs and journals to look for clues to problems and confirmation that things are working correctly.

Chapters 17 and 18 show you how to enhance the security of your Linux systems, including how to perform easy local and remote backups:

1. Logical volume management
2. File managers
3. Everything is a file
4. Managing processes
5. Special filesystems
6. Regular expressions
7. Printing
8. Hardware detection
9. Command-line programming
10. Automation with BASH scripts
11. Time and automation
12. Networking
13. systemd
14. dbus and udev
15. Using logs and journals
16. Managing users
17. Security
18. Backups

## Using and Administering Linux: Volume 3 Zero to SysAdmin: Network Services

In Volume 3 of *Using and Administering Linux*, you will start by creating a new VM on the existing virtual network. This new VM will be used as a server for the rest of this course and it will replace some of the functions performed by the virtual router that is part of our virtual network.

Chapter 2 begins this transformation from simple workstation to server by adding a new network interface card (NIC) to the VM so that it can act as a firewall and router, then changing its network configuration from DHCP to static. This includes configuring both NICs so that one is connected to the existing virtual router so as to allow connections to the outside world and so that the other NIC connects to the new "inside" network that will contain the existing VM.

Chapters 3 and 4 guide you through setting up the necessary services, DHCP and DNS, that are required to support a managed, internal network, and Chapter 5 takes you through configuration of SSHD to provide secure remote access between Linux hosts. In Chapter 6 you will convert the new server into a router with a simple yet effective firewall.

You will learn to install and configure an enterprise class email server that can detect and block most spam and malware in Chapters 7 through 9. Chapter 10 takes you through setting up a web server, and in Chapter 11 you will set up WordPress, a flexible and powerful content management system.

In Chapter 12 you return to email by setting up a mailing list using Mailman. Then Chapter 13 guides you through sharing files to both Linux and Windows hosts. Sometimes accessing a desktop remotely is the only way to do some things, so in Chapter 14 you will do just that.

Chapter 15 shows you how to set up a time server on your network and how to determine its accuracy. Although we have incorporated security in all aspects of what has already been covered, Chapter 16 covers some additional security topics.

Chapter 17 discusses package management from the other direction by guiding you through the process of creating an RPM package for the distribution of your own scripts and configuration files.

Finally, Chapter 18 will get you started in the right direction because I know you are going to ask, "Where do I go from here?"

1. Preparation
2. Server configuration
3. DHCP
4. Name services – DNS
5. Remote access with SSH
6. Routing and firewalls
7. Introducing email
8. Email clients
9. Combating spam
10. Apache web server
11. WordPress
12. Mailing lists
13. File sharing with NFS and SAMBA
14. Using remote desktop access
15. Does anybody know what time it is?
16. Security
17. Advanced package management
18. Where do I go from here?

# Taking this course

Although designed primarily as a self-study guide, this course can be used effectively in a classroom environment. This course can also be used very effectively as a reference. Many of the original course materials I wrote for Linux training classes I used to teach as an independent trainer and consultant were valuable to me as references. The experiments became models for performing many tasks and later became the basis for

automating many of those same tasks. I have used many of those original experiments in parts of this course, because they are still relevant and provide an excellent reference for many of the tasks I still need to do.

You will see as you proceed through the course that it uses many software programs considered to be older and perhaps obsolete like Sendmail, Procmail, BIND, the Apache web server, and much more. Despite their age, or perhaps because of it, the software I have chosen to run my own systems and servers and to use in this course has been well-proven and is all still in widespread use. I believe that the software we will use in these experiments has properties that make it especially valuable in learning the in-depth details of how Linux and those services work. Once you have learned those details, moving to any other software that performs the same tasks will be relatively easy. In any event, none of that "older" software is anywhere near as difficult or obscure as some people seem to think that it is.

## Who should take this course

If you want to learn to be an advanced Linux user and SysAdmin, this course is for you. Most SysAdmins have an extremely high level of curiosity and a deep-seated need to learn Linux System Administration. We like to take things apart and put them back together again to learn how they work. We enjoy fixing things and are not hesitant about diving in to fix the computer problems that our friends and coworkers bring us.

We want to know what happens when some part of computer hardware fails so we might save defective components such as motherboards, RAM memory, and hard drives. This gives us defective components with which we can run tests. As I write this, I have a known defective hard drive inserted in a hard drive docking station connected to my primary workstation and have been using it to test failure scenarios that will appear later in this course.

Most importantly, we do all of this for fun and would continue to do so even if we had no compelling vocational reason for doing so. Our intense curiosity about computer hardware and Linux leads us to collect computers and software like others collect stamps or antiques. Computers are our avocation – our hobby. Some people like boats, sports, travel, coins, stamps, trains, or any of thousands of other things, and they pursue them relentlessly as a hobby. For us – the true SysAdmins – that is what our computers are. That does not mean we are not well rounded and do not do other things. I like to travel,

read, go to museums and concerts, and ride historical trains, and my stamp collection is still there, waiting for me when I decide to take it up again.

In fact, the best SysAdmins, at least the ones I know, are all multifaceted. We are involved in many different things, and I think that is due to our inexhaustible curiosity about pretty much everything. So if you have an insatiable curiosity about Linux and want to learn about it – regardless of your past experience or lack thereof – then this course is most definitely for you.

## Who should not take this course

If you do not have a strong desire to learn about or to administer Linux systems, this course is not for you. If all you want – or need – to do is use a couple apps on a Linux computer that someone has put on your desk, this course is not for you. If you have no curiosity about what superpowers lie underneath the GUI desktop, this course is not for you.

## Why this course

Someone asked me why I want to write this course. My answer is simple – I want to give back to the Linux community. I have had several amazing mentors over the span of my career and they taught me many things – things I find worth sharing with you along with much that I have learned for myself.

This course – all three volumes of it – started its existence as the slide presentations and lab projects for three Linux courses I created and taught. For a number of reasons, I do not teach those classes any more. However, I would still like to pass on my knowledge and as many of the tips and tricks I have learned for the administration of Linux as possible. I hope that with this course I can pass on at least some of the guidance and mentoring that I was fortunate enough to have in my own career.

CHAPTER 1

# Logical Volume Management

## Objectives

In this chapter you will learn

- The advantages of logical volume management (LVM)
- The structure of LVM
- To manage LVM systems
- To create new volume groups and logical volumes
- To add space to existing volume groups and logical volumes

## The need for logical volume management

Managing disk space has always been a significant task for SysAdmins. Running out of disk space used to be the start of a long and complex series of tasks to increase the space available to a disk partition. It also required taking the system offline. This usually involved installing a new hard drive, booting to recovery or single-user mode, creating a partition and a filesystem on the new hard drive, using temporary mount points to move the data from the too small filesystem to the new, larger one, changing the content of the /etc/fstab file to reflect the correct device name for the new partition, and rebooting to remount the new filesystem on the correct mount point.

I have to tell you that when LVM first made its appearance in Fedora, I resisted it rather strongly. My initial reaction was that I did not need this additional layer of abstraction between me and the hard drives. It turns out that I was wrong and that logical volume management is very useful.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_1

Logical volume management, LVM, allows for very flexible disk space management. It provides features like the ability to add (or remove) disk space to a filesystem, and it allows for the collection of multiple physical hard drives and partitions into a single volume group which can then be divided into logical volumes. LVM also allows us to create entirely new volume groups and logical volumes. It can do all of this without rebooting or unmounting the existing filesystems, so long as drive space is available or a device can be hot-plugged.

## Running out of disk space in VirtualBox

I always like to run new distributions in a VirtualBox virtual machine for a few days or weeks to ensure that I will not run into any devastating problems when I start installing it on my production machines.

The morning, after the Fedora 11 release, I started installing Fedora 11 in a new virtual machine on my primary workstation, thinking I had enough disk space allocated to the filesystem of the host computer in which it was being installed. I did not. About a third of the way through the installation I ran out of space on that host's filesystem. Fortunately VirtualBox is great software itself. It detected the out of space condition, paused the virtual machine, and even displayed an error message indicating the exact cause of the problem.

## Recovery

Since Fedora and most modern distributions use logical volume management and I had some free space available on the hard drive, I was able to assign additional disk space to the appropriate filesystem on the fly. This means that I did not have to reformat the entire hard drive and reinstall the operating system or even reboot. I simply assigned some of the available space to the appropriate logical volume and resized the filesystem – all while the filesystem was mounted and active, and the running program, VirtualBox was using the filesystem and waiting. I resumed running the virtual machine and the installation continued as if nothing had occurred.

Running out of disk space while a critical program is running has happened many times to almost all SysAdmins. And while many programs are not as well written and resilient as VirtualBox, Linux logical volume management made it possible to recover without losing any data and without having to restart the time-consuming VM installation.

# LVM structure

The structure of a logical volume manager (LVM) disk environment is illustrated in Figure 1-1. Logical volume management enables the combining of multiple individual hard drives and/or disk partitions into a single volume group. That volume group can then be subdivided into logical volumes or used as a single large volume. Regular filesystems, such as EXT3 or EXT4, can then be created on a logical volume.

Logical volume management allows combining partitions and entire hard drives into volume groups. In Figure 1-1, two complete physical hard drives and one partition from a third hard drive have been combined into a single volume group. Two logical volumes have been created from the space in the volume group, and a filesystem, such as an EXT4 filesystem, has been created on each of the two logical volumes.

To add space to a logical volume, we can extend the (LV) into existing space on the volume group (VG) if there is any available. If not, we might need to install a new hard drive and extend an existing VG to include at least part of the new drive. Then we can extend the LV within the VG.

Note that a logical volume cannot be larger than the volume group in which it resides. A volume group may contain multiple partitions and physical volumes that encompass parts or all of multiple hard drives. This enables overall more efficient use of the physical disk space available. Volume groups may also be extended to provide additional disk space for the contained logical volumes.

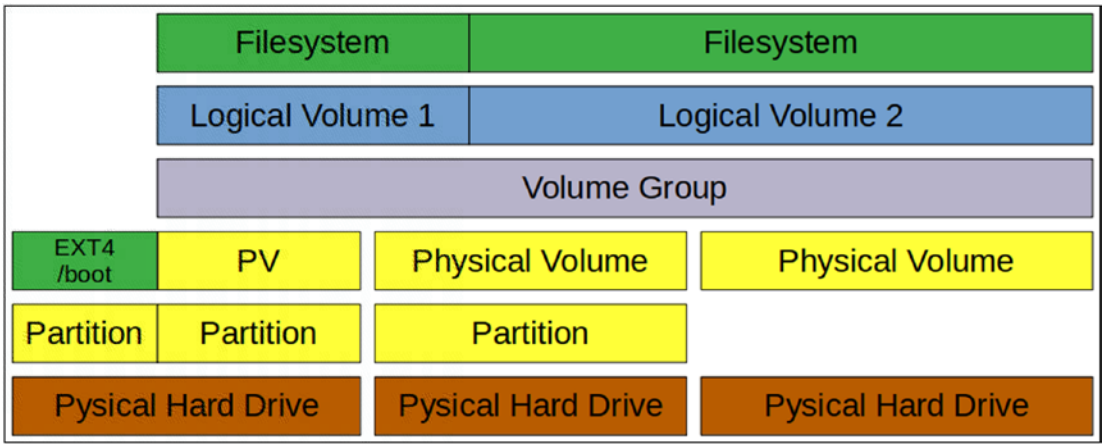


***Figure 1-1.*** *Logical volume management allows combining partitions and entire hard drives into volume groups*

# Extending a logical volume

The need to resize – especially to expand – an existing filesystem has been around since the beginnings of Unix and probably back to the very first filesystems and has not gone away with Linux. It has gotten easier, however, with logical volume management and the ability to expand an active, mounted filesystem. The steps required to do this are fairly simple but will differ depending upon specific circumstances.

Let's start with a simple logical volume extension where space is already available in the volume group. This section covers extending an existing logical volume in an LVM environment using the command-line interface (CLI). This is the simplest situation in which we might find ourselves and the easiest to accomplish.

This procedure can be used on a mounted, live filesystem only with the Linux 2.6 Kernel (and higher) and EXT3 and EXT4 filesystems. These requirements should not be hard to meet because the most recent kernel series is 5.x.x and most distributions use EXT3 or EXT4 filesystems by default.

I do not recommend that you resize a mounted and active volume on any critical system, but it can be done and I have done so many times, even on the root (/) filesystem. Use your judgment but consider how much pain might be experienced if there were a failure during the resizing vs. the pain related to taking an active system offline to perform the resizing.

---

**Warning** Not all filesystem types can be resized. The EXT3, EXT4, BTRFS, and XFS filesystems can be resized on an active, mounted filesystem. Other filesystems may not be able to be resized. Be sure to check the documentation for the filesystem you want to resize to ensure that it can be done.

---

Note that volume groups and logical volumes can be extended while the system is up and running; and the filesystem being expanded can be mounted and active during this process. Let's add 2GB of space to the /home filesystem.

All of the experiments in this chapter must be performed as root.

## EXPERIMENT 1-1

In a real-world environment, we would need to explore a little to determine whether the volume group on which our /home logical volume exists has enough space on which to do so. Let's start with that. The vgs command lists all volume groups and the lvs command lists all logical volumes.

```
[root@studentvm1 ~]# lsblk -i
NAME                        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                           8:0    0   60G  0 disk
|-sda1                        8:1    0    1G  0 part /boot
`-sda2                        8:2    0   59G  0 part
  |-fedora_studentvm1-root 253:0    0    2G  0 lvm  /
  |-fedora_studentvm1-swap 253:1    0    4G  0 lvm  [SWAP]
  |-fedora_studentvm1-usr  253:2    0   15G  0 lvm  /usr
  |-fedora_studentvm1-home 253:3    0    2G  0 lvm  /home
  |-fedora_studentvm1-var  253:4    0   10G  0 lvm  /var
  `-fedora_studentvm1-tmp  253:5    0    5G  0 lvm  /tmp
sdb                           8:16   0   20G  0 disk
|-sdb1                        8:17   0    2G  0 part
`-sdb2                        8:18   0    2G  0 part
sr0
[root@studentvm1 ~]# vgs
  VG                #PV #LV #SN Attr   VSize   VFree
  fedora_studentvm1   1   6   0 wz--n- <59.00g <21.00g
[root@studentvm1 ~]# lvs
  LV   VG                Attr       LSize  Pool Origin Data%  Meta%  Move Log
  Cpy%Sync Convert
  home fedora_studentvm1 -wi-ao----  2.00g
  root fedora_studentvm1 -wi-ao----  2.00g
  swap fedora_studentvm1 -wi-ao----  4.00g
  tmp  fedora_studentvm1 -wi-ao----  5.00g
  usr  fedora_studentvm1 -wi-ao---- 15.00g
  var  fedora_studentvm1 -wi-ao---- 10.00g
[root@studentvm1 ~]#
```

These commands show that /home filesystem is located on the fedora_studentvm1 volume group and that there is 21GB of space available on that VG. That makes this task very simple.

First expand the logical volume from existing free space within the volume group. The following command expands the LV by 2GB. The volume group name is fedora_studentvm1 and the logical volume name is home.

```
[root@studentvm1 ~]# lvextend -L +2G /dev/fedora_studentvm1/home
  Size of logical volume fedora_studentvm1/home changed from 2.00 GiB (512
  extents) to 4.00 GiB (1024 extents).
  Logical volume fedora_studentvm1/home successfully resized.
[root@studentvm1 ~]# lvs
  LV   VG                Attr       LSize  Pool Origin Data%  Meta%  Move Log
  Cpy%Sync Convert
  home fedora_studentvm1 -wi-ao----  4.00g
  root fedora_studentvm1 -wi-ao----  2.00g
  swap fedora_studentvm1 -wi-ao----  4.00g
  tmp  fedora_studentvm1 -wi-ao----  5.00g
  usr  fedora_studentvm1 -wi-ao---- 15.00g
  var  fedora_studentvm1 -wi-ao---- 10.00g
[root@studentvm1 ~]# df -h
Filesystem                          Size  Used Avail Use% Mounted on
devtmpfs                            2.0G     0  2.0G   0% /dev
tmpfs                               2.0G     0  2.0G   0% /dev/shm
tmpfs                               2.0G  1.1M  2.0G   1% /run
tmpfs                               2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/mapper/fedora_studentvm1-root  2.0G   33M  1.8G   2% /
/dev/mapper/fedora_studentvm1-usr    15G  4.1G  9.9G  30% /usr
/dev/mapper/fedora_studentvm1-home  2.0G  7.7M  1.8G   1% /home
/dev/mapper/fedora_studentvm1-var   9.8G  1.2G  8.2G  13% /var
/dev/mapper/fedora_studentvm1-tmp   4.9G   21M  4.6G   1% /tmp
/dev/sda1                           976M  179M  731M  20% /boot
tmpfs                               395M  8.0K  395M   1% /run/user/992
tmpfs                               395M     0  395M   0% /run/user/0
/dev/sdb1                           2.0G  6.0M  1.8G   1%
[root@studentvm1 ~]#
```

That extends the size of the logical volume but does not change the size of the EXT4 filesystem. Notice the logical volume has increased to 4GB but the filesystem is still 2GB in size. The following command expands the size of the filesystem to fill the space on the volume.

```
[root@studentvm1 ~]# resize2fs /dev/fedora_studentvm1/home ; df -h
resize2fs 1.44.3 (10-July-2018)
Filesystem at /dev/fedora_studentvm1/home is mounted on /home; on-line
resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/fedora_studentvm1/home is now 1048576 (4k) blocks
long.

Filesystem                          Size  Used Avail Use% Mounted on
<snip>
/dev/mapper/fedora_studentvm1-home  3.9G  9.7M  3.7G   1% /home
<snip>
[root@studentvm1 ~]#
```

We have added 2GB of space to the /home filesystem without rebooting or unmounting /home.

# Extending volume groups

The use of volume groups provides us a great deal of flexibility in managing disk space, especially when we need to add more space to one or more logical volumes.

In this section we explore multiple options for expanding disk space using volume groups. We will extend existing volume groups and create new ones to provide additional space for logical volumes. We will then create a new volume or extend an existing one.

## Create a new volume group – 1

There are times when it will be necessary to create a new volume group to contain one or more new logical volumes. Sometimes there is already space available on an existing hard drive which is on /dev/sdb. We have one unused partition of 2GB and approximately 16GB of as yet partitioned space.

In Experiment 1-2 we use an existing but unused partition to create a volume group.

## EXPERIMENT 1-2

Before beginning, verify the amount of space left on /dev/sdb. I know that we just did this in Experiment 1-1, but we should never make any assumptions about the current state of the system, so I make it a practice to check before doing anything.

```
 [root@studentvm1 ~]# lsblk
NAME                        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                           8:0    0   60G  0 disk
|-sda1                        8:1    0    1G  0 part /boot
`-sda2                        8:2    0   59G  0 part
  |-fedora_studentvm1-root 253:0    0    2G  0 lvm  /
  |-fedora_studentvm1-swap 253:1    0    4G  0 lvm  [SWAP]
  |-fedora_studentvm1-usr  253:2    0   15G  0 lvm  /usr
  |-fedora_studentvm1-home 253:3    0    4G  0 lvm  /home
  |-fedora_studentvm1-var  253:4    0   10G  0 lvm  /var
  `-fedora_studentvm1-tmp  253:5    0    5G  0 lvm  /tmp
sdb                           8:16   0   20G  0 disk
|-sdb1                        8:17   0    2G  0 part /TestFS
`-sdb2                        8:18   0    2G  0 part
```

The **lsblk** command tells us that the /dev/sdb drive has a total of 20GB of space and that the /TestFS partition, /dev/sdb1, that we created in Chapter 19 of Volume 1 uses 2GB as does the used partition, /dev/sdb2. The remaining unallocated space is about 16GB in size. Note that if you unmounted the /TestFS filesystem, the mount point would not show up, but it would be /dev/sdb1.

I want to allocate the rest of the space on the /dev/sdb drive to this new volume group. Normally I would just delete the /dev/sdb2 partition to allow me to create a new partition consisting of the rest of the space on the hard drive. That would be too easy and definitely much less fun and informative. We will create a third partition on the drive, and these two partitions will be combined into a single volume group on which we will create a logical volume.

Create a new primary partition on /dev/sdb that uses the rest of the space on the drive. You should be able to do this now without explicit instructions. Then verify that the partition was created.

Create two physical volumes (PV) on each /dev/sdb2 and /dev/sdb3. We can do this with a single command. The **pvcreate** command will warn that the btrfs partition you created in Volume 1, Chapter 19, is detected and be sure you want to delete it, if you didn't remove it then.

```
[root@studentvm1 ~]# pvcreate /dev/sdb2 /dev/sdb3
  Physical volume "/dev/sdb2" successfully created.
  Physical volume "/dev/sdb3" successfully created.
```

Now create a volume group encompassing both of the partitions /dev/sdb2 and /dev/sdb3.

```
[root@studentvm1 ~]# vgcreate NewVG-01 /dev/sdb2 /dev/sdb3
  Volume group "NewVG-01" successfully created
```

Verify the new volume group.

```
[root@studentvm1 ~]# vgs
  VG                #PV #LV #SN Attr   VSize   VFree
  NewVG-01            2   0   0 wz--n-  17.99g  17.99g
  fedora_studentvm1   1   6   0 wz--n- <59.00g <19.00g
```

Notice that the combined size of the volume group NewVG-01 is almost 18GB, all of which is available. Now create a new logical volume on this volume group. The -L 2G option defines the size of the new volume. Obviously, NewVG-01 is the name of the volume group on which the volume is to be created and --name TestVol1 specifies the name of the new logical volume.

```
[root@studentvm1 ~]# lvcreate -L 2G NewVG-01 --name TestVol1
  Logical volume "TestVol1" created.
[root@studentvm1 ~]# lvs
  LV       VG                Attr       LSize  Pool Origin Data%  Meta%  Move
  Log Cpy%Sync Convert
  TestVol1 NewVG-01          -wi-a-----  2.00g
  home     fedora_studentvm1 -wi-ao----  4.00g
  root     fedora_studentvm1 -wi-ao----  2.00g
  swap     fedora_studentvm1 -wi-ao----  4.00g
  tmp      fedora_studentvm1 -wi-ao----  5.00g
  usr      fedora_studentvm1 -wi-ao---- 15.00g
  var      fedora_studentvm1 -wi-ao---- 10.00g
```

That was easy. However, the man page for **lvcreate** is long and complex, and it is not clear, even from the examples, that this simple command can create a new volume.

Now create an EXT4 filesystem on the new volume, mount it temporarily on /mnt, and test it by creating a few files with some test data there. There is no need to create a permanent mount point for this filesystem or to add an entry in /etc/fstab because this filesystem will not be used for anything more than a bit of testing. Note that this is the exact type of use case for which the /mnt mount point is intended by the Linux FHS as we saw in Chapter 19 in Volume 1.

```
[root@studentvm1 ~]# mkfs -t ext4 /dev/mapper/NewVG--01-TestVol1
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 524288 4k blocks and 131072 inodes
Filesystem UUID: 7f14a6c7-b425-4307-8103-58e5b0bd593d
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

[root@studentvm1 ~]# mount /dev/mapper/NewVG--01-TestVol1 /mnt
[root@studentvm1 ~]# ll /mnt
total 16
drwx------. 2 root root 16384 Jan 22 12:05 lost+found
[root@studentvm1 ~]# lsblk
NAME                      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                         8:0    0   60G  0 disk
|-sda1                      8:1    0    1G  0 part /boot
`-sda2                      8:2    0   59G  0 part
  |-fedora_studentvm1-root 253:0    0    2G  0 lvm  /
  |-fedora_studentvm1-swap 253:1    0    4G  0 lvm  [SWAP]
  |-fedora_studentvm1-usr  253:2    0   15G  0 lvm  /usr
  |-fedora_studentvm1-home 253:3    0    4G  0 lvm  /home
  |-fedora_studentvm1-var  253:4    0   10G  0 lvm  /var
  `-fedora_studentvm1-tmp  253:5    0    5G  0 lvm  /tmp
sdb                         8:16   0   20G  0 disk
|-sdb1                      8:17   0    2G  0 part /TestFS
|-sdb2                      8:18   0    2G  0 part
```

```
`-sdb3                        8:19   0   16G  0 part
  `-NewVG--01-TestVol1      253:6    0    2G  0 lvm  /mnt
sr0                          11:0    1 1024M  0 rom
[root@studentvm1 ~]#
```

It is not necessary to spend a lot of time, but do a bit of testing on this new volume. Just create a few files with some data in them and verify that all is working. When you are finished testing, unmount the volume.

Although the two partitions we used to create the new volume group in Experiment 1-2 are both on the same hard drive, it does illustrate that multiple partitions and even entire hard drives can be combined into a single volume group.

## Create a new volume group – 2

In situations in which there is no existing space to be found, it will be necessary to add a new drive in order to create that space. Experiment 1-3 takes us through the process of adding a new hard drive and then creating a volume group and a logical volume in the new space.

### EXPERIMENT 1-3

In Experiment 19-8 we created a new virtual hard drive for the VM. As part of that process, you increased the total number of ports on the SATA controller to at least 5 so we could add more virtual hard drives to that controller. We added one hard drive in that experiment, and now we add another virtual hard drive to that controller. If necessary you can refer to the illustrations in Experiment 19-8.

In the VirtualBox manager, open the **Settings** dialog and go to the **Storage** tab. On the SATA controller line, click the rightmost icon to add a new hard drive. Hover over the icon and the hint says, "Adds hard disk." Click the **Adds hard disk** icon. Then click the **Create new disk** button. Leave VDI as the hard disk file type and click **Next** to continue. Leave the storage as **Dynamically allocated** and click the **Next** button.

On the **File location and size** dialog, we won't change the default location but we will change the file name to **StudentVM1-2** and set the size to 2GB. We won't need much space for this. Click the **Create** button to complete creation of the new virtual hard drive. Click the **OK** button to close the settings dialog.

Use the **lsblk** command to determine the drive identifier. On my StudentVM1 host, it is /dev/sdc and it should be on your virtual machine as well. You should be sure to use the correct device for your computer.

Now create a new physical volume (PV) on the new hard drive.

```
[root@studentvm1 ~]# pvcreate /dev/sdc
  Physical volume "/dev/sdc" successfully created.
```

Extend the NewVG-01 volume group to include the new physical volume.

```
[root@studentvm1 ~]# vgextend NewVG-01 /dev/sdc
  Volume group "NewVG-01" successfully extended
[root@studentvm1 ~]#
```

Extend the logical volume. This command extends the logical volume by adding all of the space on the new /dev/sdc physical volume.

```
[root@studentvm1 ~]# lvextend /dev/NewVG-01/TestVol1 /dev/sdc
  Size of logical volume NewVG-01/TestVol1 changed from 2.00 GiB (512
  extents) to <4.00 GiB (1023 extents).
  Logical volume NewVG-01/TestVol1 successfully resized.
[root@studentvm1 ~]#
```

Finally, resize the filesystem.

```
[root@studentvm1 ~]# resize2fs /dev/NewVG-01/TestVol1
resize2fs 1.44.3 (10-July-2018)
Filesystem at /dev/NewVG-01/TestVol1 is mounted on /mnt; on-line resizing
required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/NewVG-01/TestVol1 is now 1047552 (4k) blocks long.
```

If the **resize2fs** command fails, run **e2fsck -f /dev/NewVG-01/TestVol1** and then retry the resizing.

```
[root@studentvm1 ~]# lsblk -i
NAME                        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                           8:0    0   60G  0 disk
|-sda1                        8:1    0    1G  0 part /boot
`-sda2                        8:2    0   59G  0 part
  |-fedora_studentvm1-root 253:0    0    2G  0 lvm  /
  |-fedora_studentvm1-swap 253:1    0    4G  0 lvm  [SWAP]
  |-fedora_studentvm1-usr  253:2    0   15G  0 lvm  /usr
  |-fedora_studentvm1-home 253:3    0    4G  0 lvm  /home
  |-fedora_studentvm1-var  253:4    0   10G  0 lvm  /var
  `-fedora_studentvm1-tmp  253:5    0    5G  0 lvm  /tmp
sdb                           8:16   0   20G  0 disk
|-sdb1                        8:17   0    2G  0 part /TestFS
|-sdb2                        8:18   0    2G  0 part
`-sdb3                        8:19   0   16G  0 part
  `-NewVG--01-TestVol1      253:6    0    4G  0 lvm  /mnt
sdc                           8:32   0    2G  0 disk
`-NewVG--01-TestVol1        253:6    0    4G  0 lvm  /mnt
sr0                          11:0    1 1024M  0 rom
[root@studentvm1 ~]#
```

We have expanded the capacity of a logical volume by adding space to the volume group from a new drive.

Notice that none of the experiments in this chapter required rebooting the computer. All of the experiments were performed with the system fully functional.

# Tips

Using LVM is already quite easy, but I have found a few things that can make it even more so.

I use the extended filesystems unless there is clear reason to use another filesystem. Not all filesystems support resizing, but XFS, BTRFS, EXT3, and EXT4 do. The EXT filesystems are also very fast and efficient. The EXT filesystems can be tuned by a knowledgeable SysAdmin to meet the needs of most environments if the default tuning parameters do not.

I use meaningful volume and volume group names to help make identification of groups and volumes easy while working on them. I also use EXT filesystem labels for the same reason. Filesystem labels can make mounting filesystems easy by reducing the amount of typing required to mount them manually or to add the specification to the /etc/fstab file.

Be aware that volume groups that span multiple physical volumes will fail completely if one of the physical devices composing them fails. LVM is not inherently fault tolerant although a properly designed software RAID system using LVM can be. As always, make frequent backups of everything; see Chapter 18 in this volume of the course.

## Advanced capabilities

LVM has some additional very powerful and interesting advanced features that are beyond the scope of this course. It is possible to create hybrid volumes that consist of rotating hard drives along with SSDs in which the SSD acts as a data cached for the slower hard disk drive. LVM can be used to create RAID volumes, mirror volumes, and snapshot volumes.

## Chapter summary

Logical volume management (LVM) provides a high-level tool for the advanced management of disk space on modern Linux hosts. By abstracting the hardware into volume groups and logical volumes, it enables the SysAdmin to create volumes that are not limited by the physical space on individual hard drives. It provides the capability to manage logical volumes by adding space when and where it is needed without disturbing ongoing operations.

The LVM facility has many more functions than we have used in this chapter. They enable capabilities such as creating and restoring backups of volume groups, deleting, renaming, resizing, and more of the groups and volumes that make up the total of an LVM system.

I suggest reading the man pages for these commands to at least get a fair idea of the vast amount of control that can be exerted over an LVM system. You can use tab completion to locate other LVM-related commands: lv<tab><tab>.

# Exercises

Perform these exercises to complete this chapter:

1. What are some of the reasons for using logical volume management?
2. How does the information available with the **vgdisplay** command differ from that of the **vgs** command?
3. I sometimes use a drive docking station to test hard drives I suspect may have errors or that I want to attempt data recovery from. How would I access the data on a hard drive if the drive had been configured as the only member of a volume group?
4. Expand the /tmp filesystem by 5GB onto some of the remaining and yet unused space on /dev/sdc.

CHAPTER 2

# File Managers

## Objectives

In this chapter you will learn

- The functions of a file manager
- Basic usage of Midnight Commander (MC), a text-mode file manager
- Basic usage of Krusader, a graphical file manager based on Midnight Commander
- Basic usage of Thunar, the graphical default file manager for the Xfce desktop
- Short introductions to a few other file managers

## Introduction

One of the most common administrative tasks that end users and administrators alike need to perform is file management. Managing files can consume a major portion of your time. Locating files, determining which files and folders (directories) are taking the most disk space, deleting files, moving files, and simply opening files for use in an application are some of the most basic yet frequent tasks we do as computer users. File management programs are tools that are intended to streamline and simplify those necessary chores.

This chapter will be mostly about learning the Midnight Commander text-mode file manager. The reason for this is that, as SysAdmins, we work mostly on the command line and Midnight Commander will be available even if it might need to be installed. We will also look at the Thunar graphical file manager which is the default for the Xfce desktop, and we will take a very brief look at some other available file managers.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_2

As with every aspect of Linux, there are many options available for file managers. A few of those provided by my usual distribution, Fedora, are listed in the following. Some are text and others are graphical interfaces.

## Text-mode interface

- Midnight Commander
- Vifm
- nnn

## Graphical interface

- Thunar
- Krusader
- Dolphin
- XFE

I have used each of these at different times for various reasons, and they all have qualities to recommend them. Ranging from very simple to feature-packed, there is a file manager available that will meet your needs. This chapter looks briefly at each of the file managers listed.

# Default file manager

Like most Linux distributions, Fedora has a default graphical desktop file manager, currently Thunar for the Xfce desktop. The Linux desktop usually has an icon that looks like a little house; that is your "home" directory, that is, folder. Click the Home icon and the default file manager opens with your home directory as the PWD, or Present Working Directory. The Home icon is located on the desktop along with the Trash icon and some drive icons, as shown in Figure 2-1.

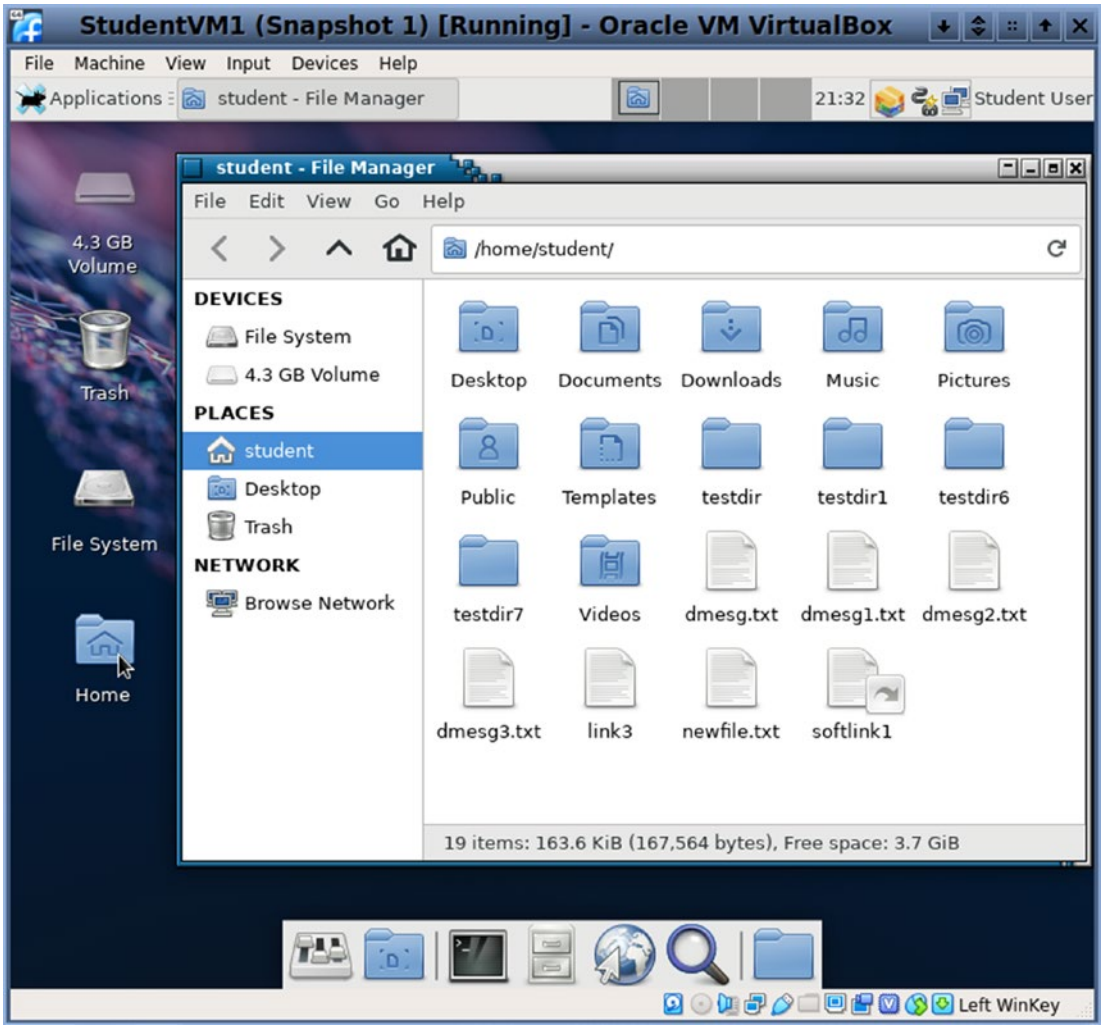


***Figure 2-1.*** *The Xfce desktop with the Home icon and the Thunar file manager open*

Each desktop such as Xfce, KDE Plasma, Cinnamon, LXDE, and so on has a default graphical file manager. In Xfce the default file manager can be changed using **Applications ➤ Settings ➤ Preferred Applications ➤ Utilities ➤ File Manager**. We will cover Thunar in more detail later in this chapter along with installing and using some of the other graphical file managers.

# Text-mode file managers

Because we SysAdmins spend so much time using the command-line interface, we will spend most of our time in this chapter on text-mode file managers. Text-mode file managers are particularly useful when a GUI is not available, but they can also be used as a primary file manager in a desktop terminal emulator session even when you are using a GUI.

There are several fine text-mode file managers to choose from. My personal favorite is Midnight Commander, but there are others that are also powerful, usable, and well respected.

## Midnight Commander

Midnight Commander[1] is a text-based program that I use frequently because I often have need to interact with local and remote Linux computers using the CLI. It can be used with almost any of the common shells and remote terminals through SSH.

Midnight Commander provides an interactive, visually based, text-mode program for navigating the filesystem and managing files. It can be used to copy, edit, move, or delete files and complete directory trees. It can also be used to expand archive files of various types and explore their contents.

You can start Midnight Commander from the CLI with the **mc** command. Figure 2-2 shows Midnight Commander in the Konsole terminal emulator program. The user interface for Midnight Commander is two text-mode panels, left and right, which each displays the contents of a directory. Along the very top of the Midnight Commander (a.k.a. MC) interface is a menu bar containing menu items for configuring Midnight Commander the current panel and a selection bar that highlights one line of the current directory is displayed in the current panel.

The top of each panel displays the name of the current directory for that panel. The directory entry for the current panel is highlighted.

Navigation is accomplished with the arrow and tab keys. The Enter key can be used to enter a highlighted directory. The bottom portion of the interface displays information about the file or directory highlighted in each panel, a hint feature and a line of function key labels; you can simply press the function key on your keyboard that corresponds

[1]Midnight Commander, https://midnight-commander.org/

to the function you want to perform. Between the hint line and the function keys is a command line. You can type any CLI command here you would at the standard Bash or other shell prompt; it is, after all, a Bash prompt.

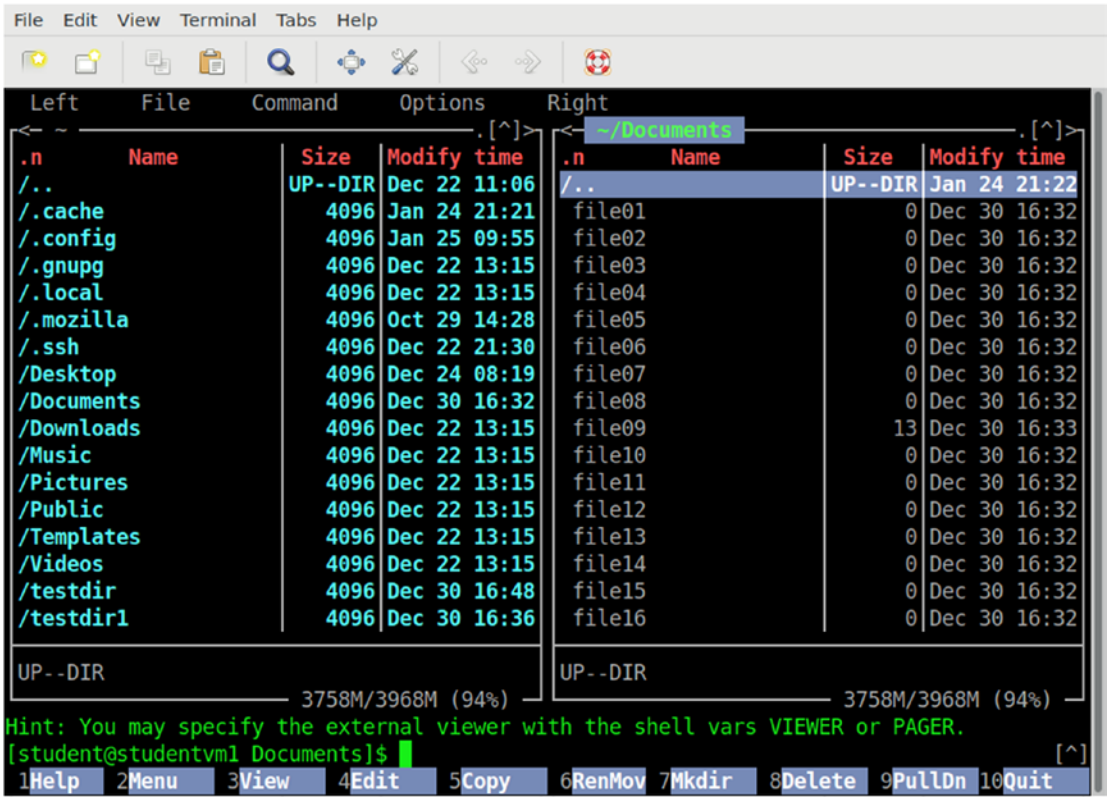


***Figure 2-2.*** *Midnight Commander can be used to move, copy, and delete files*

## EXPERIMENT 2-1

As the root user, install Midnight Commander.

```
[root@studentvm1 ~]# dnf -y install mc
```

Then as the user student, ensure that the PWD is your home directory. Start Midnight Commander with the **mc** command.

Midnight Commander starts with two open panels. Switch between the panels using the **Tab** key. Use the arrow keys to move the highlight bar (cursor) up and down the list of files and directories on the current panel. Highlight the Documents directory in the right panel and press the Enter key to change into that directory.

Moving up to the parent directory can be accomplished by highlighting the double-dot (..) entry at the top of the list in the panel and pressing the Enter key. In the left panel in Figure 2-3, this entry is shown at the top and the Size column shows "UP—DIR." But don't do that now.

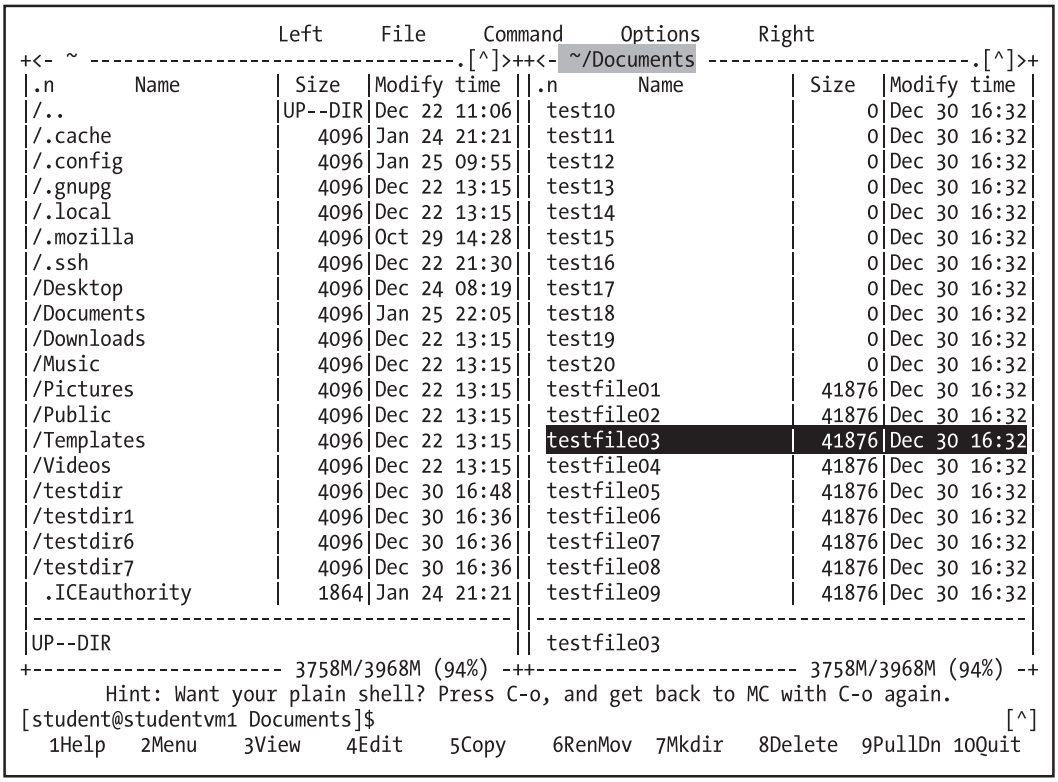


***Figure 2-3.*** *Midnight Commander with two panels open. The* ***Tab*** *key switches between panels*

---

**Note** I started Midnight Commander using the -a option which uses ASCII plain text characters to draw lines instead of advanced line drawing characters as shown in Figure 2-2. Those line drawing characters do not line up quite as well when copied from the terminal session into a document.

---

To enter a command at the Midnight Commander command prompt, just start typing. Let's look at the value of the $SHELL variable.

```
|----------------------------------||----------------------------------|
|UP--DIR                           ||UP--DIR                           |
+--------------- 3690M/3968M (92%) -++---------------- 3690M/3968M (92%) -+
Hint: Want your plain shell? Press C-o, and get back to MC with C-o again.
[student@studentvm1 ~]$ echo $SHELL                                      [^]
 1Help  2Menu  3View  4Edit  5Copy   6RenMov  7Mkdir  8Delete  9PullDn 10Quit
```

The result is shown in a sub-shell. Press Enter to exit the sub-shell and return to MC.
In the right panel, scroll down to highlight one of the files that has some content. The size of a file is shown in the Size column. View the file content by pressing the **F3** key as seen in Figure 2-4. This is a viewer only and you cannot edit the file from within this window. The F4 key would open the file in an editor.

```
/home/student/Documents/testfile03                          1902/41876          4%
[    0.000000] Linux version 4.19.10-300.fc29.x86_64 (mockbuild@bkernel03.phx2.fedoraproj
ect.org) (gcc version 8.2.1 20181105 (Red Hat 8.2.1-5) (GCC)) #1 SMP Mon Dec 17 15:34:44
UTC 2018
[    0.000000] Command line: BOOT_IMAGE=/vmlinuz-4.19.10-300.fc29.x86_64 root=/dev/mapper
/fedora_studentvm1-root ro resume=/dev/mapper/fedora_studentvm1-swap rd.lvm.lv=fedora_stu
dentvm1/root rd.lvm.lv=fedora_studentvm1/swap rd.lvm.lv=fedora_studentvm1/usr rhgb quiet
LANG=en_US.UTF-8
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes, using 'st
andard' format.
[    0.000000] BIOS-provided physical RAM map:
<snip>
[    0.000000] SMBIOS 2.5 present.
[    0.000000] DMI: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/01/2006
[    0.000000] Hypervisor detected: KVM
[    0.000000] kvm-clock: Using msrs 4b564d01 and 4b564d00
 1Help   2UnWrap  3Quit    4Hex     5Goto    6        7Search  8Raw     9Format 10Quit
```

***Figure 2-4.*** *Press the F3 key to view the content of a file*

The top line of the viewer shows the path and file name, the distance into the file being viewed compared to the total amount of data in the file (1902/41876), and the percentage of the distance into the file being viewed.

There are navigation, search, and viewing options that can be accessed using the Function keys as shown on the bottom line. Press **F3** again or **F10** to quit the viewer and return to the main MC window.

Notice the function key assignments at the bottom of the MC window. **F1** will display some help. There are also function keys for move, copy, delete, and quit, among others. Press the corresponding function key on the keyboard to perform that function.

Be sure that the F1 and F10 keys are not being captured by the terminal emulator. Use the menu bar of the Xfce4-terminal emulator session and select **Edit ➤ Preferences** and then select the **Advanced** tab. Add check marks to the "**Disable menu shortcut key (F10 by default)**" and "**Disable help window shortcut key (F1 by default)**" options to allow Midnight Commander to capture these keystrokes.

This should not be an issue for virtual consoles, and if you're using a GUI terminal emulator, you can also click the F1 or F10 using the mouse.

Highlight one of the files in the right panel and press **F5** to start the copy of the file. This opens the Copy dialog shown in Figure 2-5.

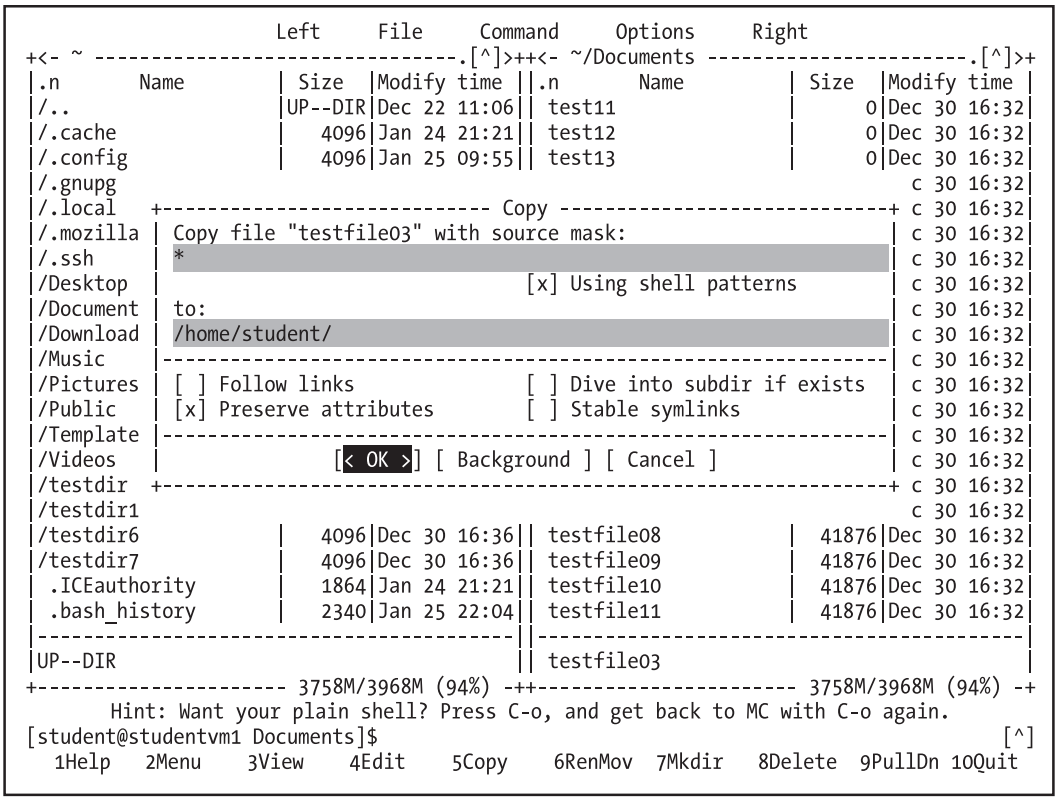


***Figure 2-5.*** *The Copy dialog provides some options for the copy command*

Multiple files can be selected on which to perform move, copy, or delete operations. Highlight each file and press the **Insert** key to select each desired file. In this case we only want to copy one file. Use the down arrow key to sequence through the options and highlight **OK**, then press **Enter** to complete the operation.

Switch to the left panel and scroll down until you can see the file you copied to verify that it is in the directory there. Highlight the copied file in the left panel and press **F8** to delete that file. A dialog opens to allow verification that you want to delete the selected file or files. Select **Yes** and press **Enter**.

Switch back to the right panel and select several non-adjacent files using the Insert key. Highlight each desired file and press the Insert key to "tag" it, as the MC documentation terms selecting a file or directory. Your results should be similar to those in Figure 2-6.

```
                   Left      File      Command      Options      Right
+<- ~ ------------------------------.[^]>++<- ~/Documents ----------------------.[^]>+
|.n       Name         | Size  |Modify time ||.n       Name          | Size  |Modify time |
| .bashrc              |    350|Dec 25 14:26|| test16               |      0|Dec 30 16:32|
| .esd_auth            |     16|Dec 22 13:15|| test17               |      0|Dec 30 16:32|
| .vboxclien~board.pid|      6|Jan 24 21:21|| test18               |      0|Dec 30 16:32|
| .vboxclien~splay.pid|      6|Jan 24 21:21|| test19               |      0|Dec 30 16:32|
| .vboxclien~ddrop.pid|      6|Jan 24 21:21|| test20               |      0|Dec 30 16:32|
| .vboxclien~mless.pid|      6|Jan 24 21:21|| testfile01           |  41876|Dec 30 16:32|
| .viminfo             |   3383|Jan 16 13:43|| testfile02           |  41876|Dec 30 16:32|
| .xscreensaver        |   8816|Dec 23 17:13|| testfile03           |  41876|Dec 30 16:32|
| .xsession-errors     |   2939|Jan 24 21:24|| testfile04           |  41876|Dec 30 16:32|
| .xsession-errors.old|   2405|Jan 17 21:27|| testfile05           |  41876|Dec 30 16:32|
| dmesg.txt            |  41876|Dec 30 16:37|| testfile06           |  41876|Dec 30 16:32|
| dmesg1.txt           |  41936|Jan 16 14:08|| testfile07           |  41876|Dec 30 16:32|
| dmesg2.txt           |  41876|Dec 30 16:37|| testfile08           |  41876|Dec 30 16:32|
| dmesg3.txt           |  41876|Dec 30 16:37|| testfile09           |  41876|Dec 30 16:32|
| link3                |      0|Dec 30 16:40|| testfile10           |  41876|Dec 30 16:32|
| newfile.txt          |      0|Dec 30 16:37|| testfile11           |  41876|Dec 30 16:32|
|!softlink1            |      5|Dec 30 16:48|| testfile12           |  41876|Jan 27 08:52|
|--------------------------------------||---------- 209,380 B in 7 files -----------|
|-> link1                              || testfile09                                  |
+---------------------- 3758M/3968M (94%) -++----------------------- 3758M/3968M (94%) -+
          Hint: Want to do complex searches? Use the External Panelize command.
[student@studentvm1 Documents]$                                                    [^]
  1Help   2Menu    3View    4Edit    5Copy    6RenMov  7Mkdir   8Delete  9PullDn 10Quit
```

***Figure 2-6.*** *Select – or "tag" – several files in the right panel*

After tagging some files in the right directory panel, press the **F8** key and then verify that you want to delete the files. All of the tagged files will be deleted. If the selection bar is highlighting an untagged file, that file will not be deleted. We could have copied or moved the files instead of deleting them. You could also simply run the commands from the MC command line.

New directories can be created in a couple ways. First just type the following command at the Midnight Commander command line as shown in Figure 2-7. Just start typing as the selection bar is only used within the directory panels. Press **Enter**.

```
+---------------------- 3758M/3968M (94%) -++----------------------- 3758M/3968M (94%) -+
      Hint: Do you want Lynx-style navigation? Set it in the Configuration dialog.
[student@studentvm1 Documents]$ mkdir Directory01                                  [^]
  1Help   2Menu    3View    4Edit    5Copy    6RenMov  7Mkdir   8Delete  9PullDn 10Quit
```

***Figure 2-7.*** *Commands can be entered on the MC command line*

Another way to create a directory is to use the F7 key and use the dialog box in Figure 2-8. The name of the directory or file that is under the selection bar will be displayed in the dialog. Delete that, type in the directory name, and select **OK** and press **Enter** to complete creating the directory.

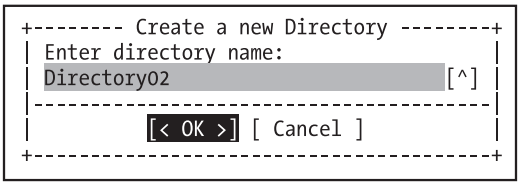


***Figure 2-8.*** *Type a new directory name in this dialog*

Let's take a few moments to look at the menu items across the top of the MC interface. The Left and Right menus allow you to personalize the display of data in the left and right panels, respectively. The File menu allows file operations such as creating links, changing the file mode and ownership, displaying only files that match a filter, making new directories, deleting and copying files, and more. Some of these functions are duplicated by the function keys.

There are a couple changes I like to make to the MC interface. I do like to view the file mode (permissions) and size when the selection bar is on them. This data is shown in the mini status line. It is necessary to make this configuration change to each panel separately. In Figure 2-9 I have already made this change to the left panel, and you can see mode and size of the file dmesg1.txt at the bottom of the left panel. Although the selection bar is in the right panel at the moment, the last selection in the left panel is the one shown in the mini status line.

To access the top menu bar, press the **F9** key. Use the right and left arrow keys to move the highlight between the main menu items; select the **Right** menu and then use the down arrow key to open the drop-down menu under the Right menu as shown in Figure 2-9. Continue to use the down arrow key to select the **Listing format** menu item and press **Enter**.

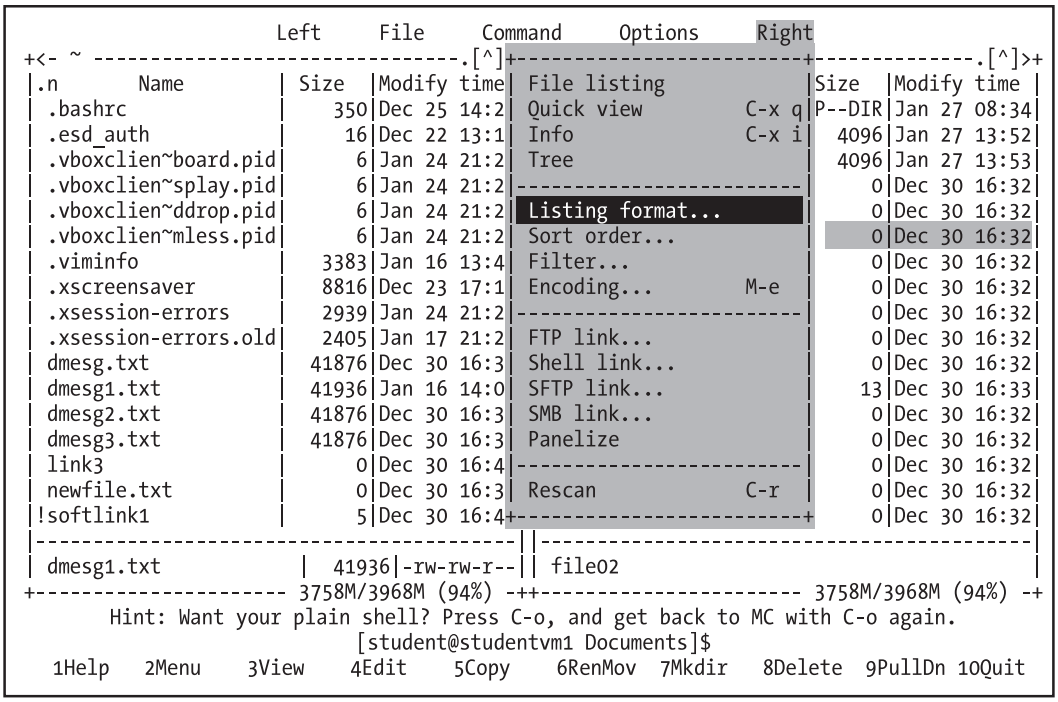


***Figure 2-9.*** *To view the file mode and size, press F9 to access the top menu and select* ***Listing format*** *for the left and right panel settings*

The Listing format dialog is shown in Figure 2-10. Use the arrow keys to highlight **User mini status**. Press the **space bar** to place an X in the check box. Select **OK** and press Enter. Now do this for the left panel as well.

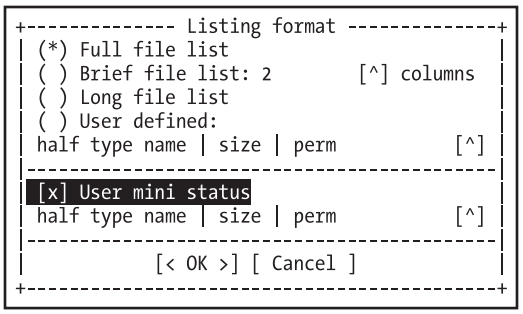


***Figure 2-10.*** *Place an X in the User mini status check box to view file size and mode*

I also like to use the Vim editor rather than the default MC internal editor. That internal editor is perfectly fine, but my fingers prefer the Vim key combinations because they remember them after 20 years working with Vi and Vim. This can be changed too.

From the top menu bar, select **Options ➤ Configuration** and you will see the **Configure options** dialog in Figure 2-11. Use the arrow keys to select **Use internal edit** and press the space bar to remove the X from the check box. Select **OK** and press **Enter** to complete this change.

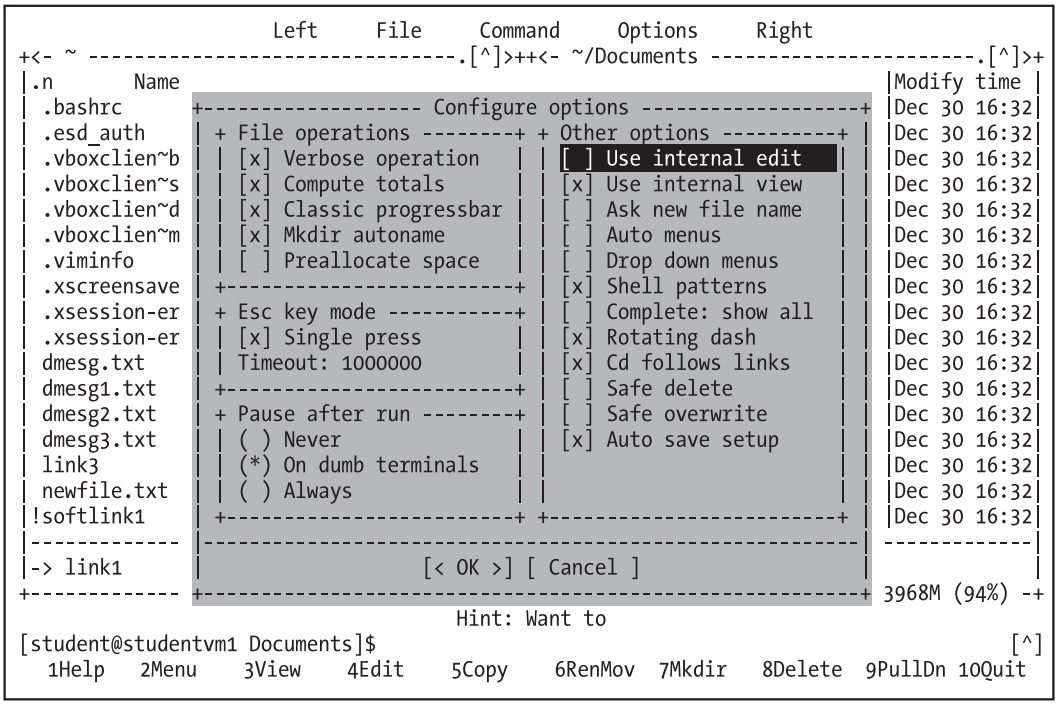


***Figure 2-11.*** *Remove the X from "Use internal edit" to use an external editor instead*

The F4 key is used to edit an existing, selected file. If you have an F14 key on your keyboard, it can be used to start the editor with a new, empty file, or you can use Shift+F4 to simulate the F14 key. These keys invoke the Vim editor now that we have changed the editor option to external. A different external editor may be specified in the $EDITOR environment variable.

There are other options I sometimes change, such as the colors. **Select Options ➤ Appearance** as shown in Figure 2-12. The resulting dialog has the current skin (color combination) highlighted. To choose a new skin, press **Enter** to view the list. Scroll to the one you want and press **Enter**. The change is immediate, and if it looks like it will be what you want, select **OK** and press **Enter**.

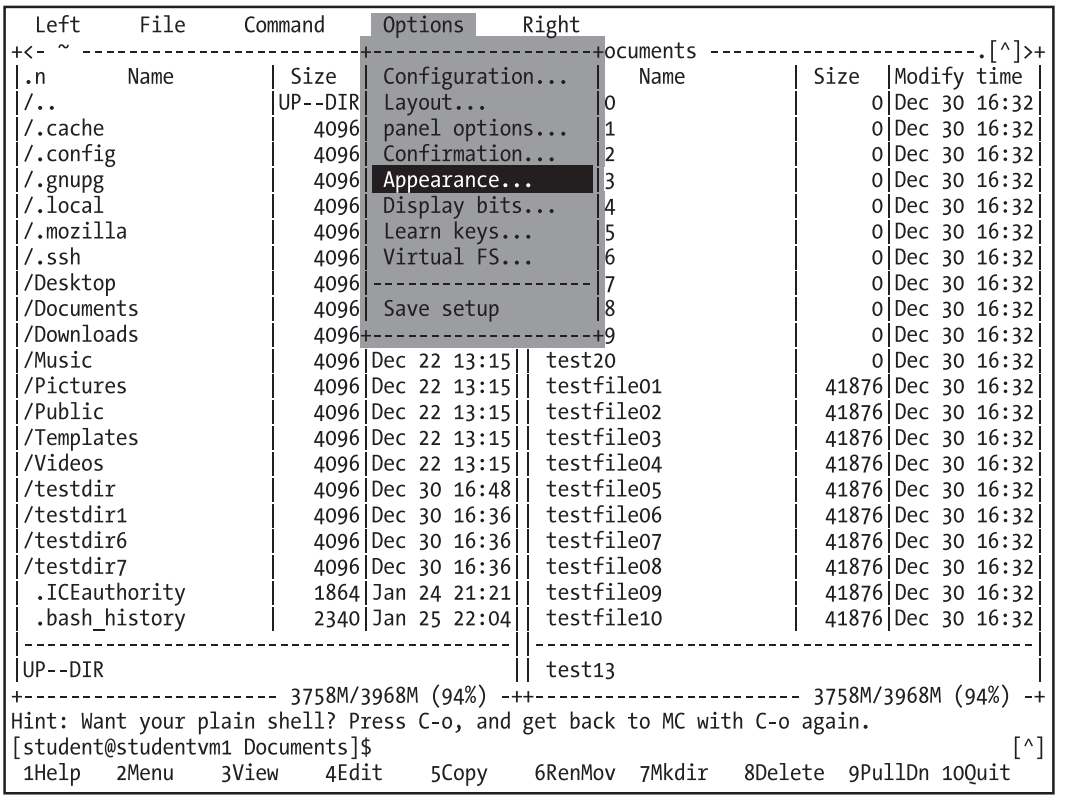


***Figure 2-12.*** *The* ***Options ➤ Appearance*** *dialog allows selection of "skins" for various colors*

Any changes you make to viewing modes or options must be saved because they are only temporary otherwise and will no longer be in effect the next time Midnight Commander is started. Save the changes you have made by accessing the top menu bar then **Options ➤ Save setup**.

MC is very powerful with many features that make it one of the most useful tools I have in my toolbox. Midnight Commander has a man page (**`man mc`**) that is over 2600 lines long. Be sure to read it to discover all of the capabilities and options available.

Like the rest of Linux, there is much more to MC than we have time for in this experiment. You should spend more time experimenting on your own with MC to learn more about it.

Exit from Midnight Commander by pressing the **F10** key. You may need to then press **Enter** to answer "Yes" to the question.

Midnight Commander has a virtual filesystem that enables connecting a local instance with remote hosts using FTP, SMB (SAMBA), and SSH protocols. This allows files to be copied from one host to another. Files on a remote host can be managed locally with MC.

# Other text-mode file managers

There are a number of other text-mode file managers available. Other than while researching this chapter, I have never used any of them because I find MC meets all of my text-mode file manager needs. We will only look at a couple that have Fedora packages available which makes them easy to install and which are, in my opinion at least, reasonably easy to use.

## Vifm

Vifm is a dual-pane file manager that provides a Vim-like environment. If you like Vim and its commands are embedded in your muscle memory, this is the file manager for you. Figure 2-13 illustrates the very minimalistic interface. The panes can be split horizontally or vertically.

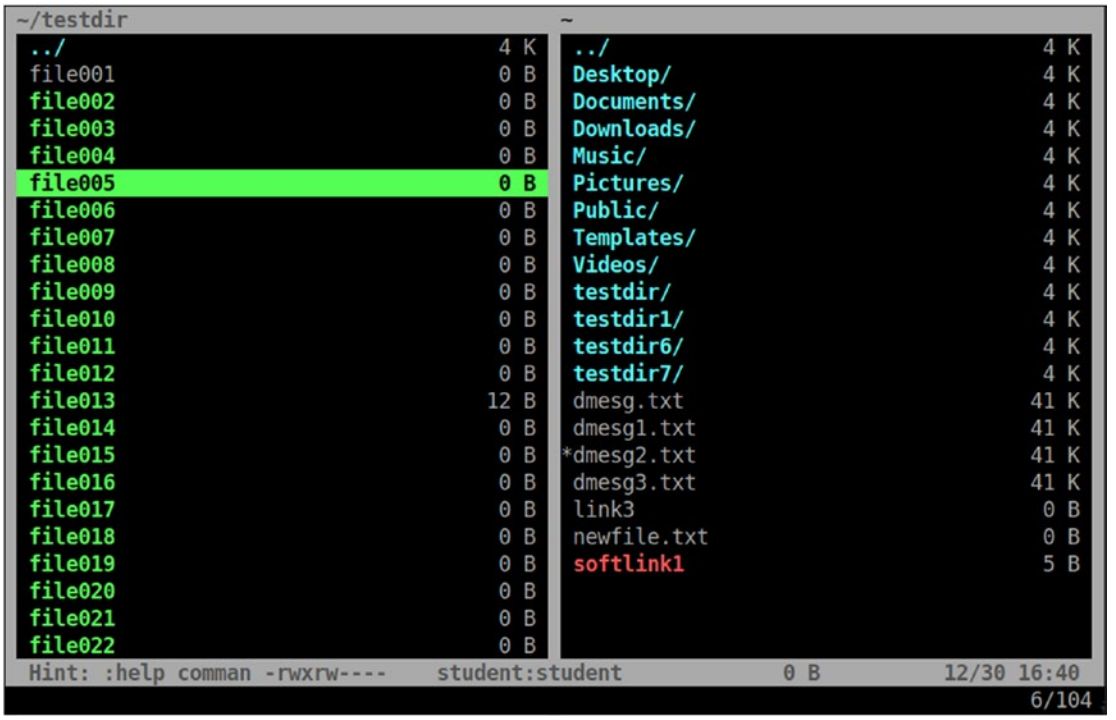


***Figure 2-13.*** *Vifm is a dual-pane file manager that uses Vim-like key combinations*

You can highlight a file and use a command like **dd** to delete a file. Vifm pops up a verification dialog, and you can respond Yes or No to complete the action. The **yy** command yanks (copies) a file in one pane and the **p** command pastes it in the other pane. The **Tab** key is used to switch active panes. Select a file and press the **Enter** key to open the file for editing with Vim. Exit from Vim in the normal manner. You can also exit from Vifm using the same keystrokes as used to exit from Vim itself. Vifm supports multi-file operations like delete, move, and copy.

The Fedora package name is Vifm if you want to install it.

## nnn

The nnn file manager, as shown in Figure 2-14, is a very simple, single pane tool that offers no frills. Only one directory and its content is displayed at a time. Use the arrow keys to select a directory and press the **Enter** key to make that directory the PWD. Select a file and press **e** to edit the file in Vim.

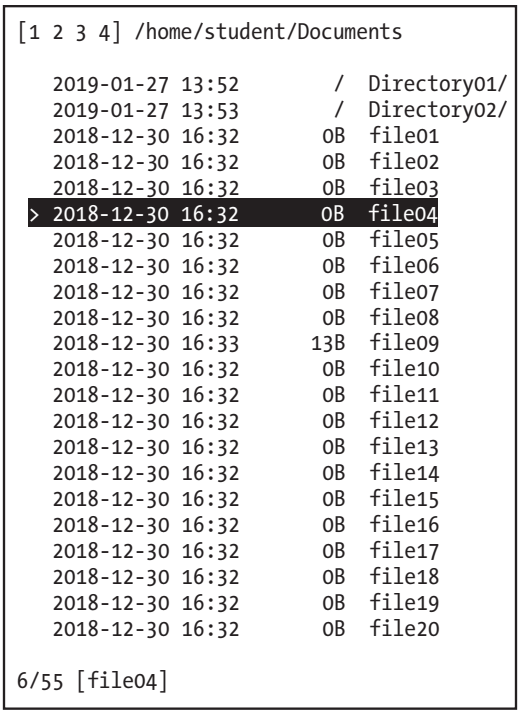


***Figure 2-14.*** *nnn is a single pane file manager with a very simple interface*

A press of the **n** key opens a dialog to create a new file or directory. The package name is nnn to install this file manager.

# Graphical file managers

Like all Linux tools, there are plenty of choices when it comes to graphical file managers. We will discuss a few of these here, but these are not all of the ones from which you might choose.

## Krusader

Krusader is an exceptional file manager that is modeled after Midnight Commander. It uses a similar two-panel layout but in a graphical interface as shown in Figure 2-15 instead of a text-mode interface. Krusader allows you to use the same keyboard navigation and command structure as Midnight Commander and also allows you to use the mouse or trackball to navigate and perform all of the standard drag-and-drop operations you would expect on files.

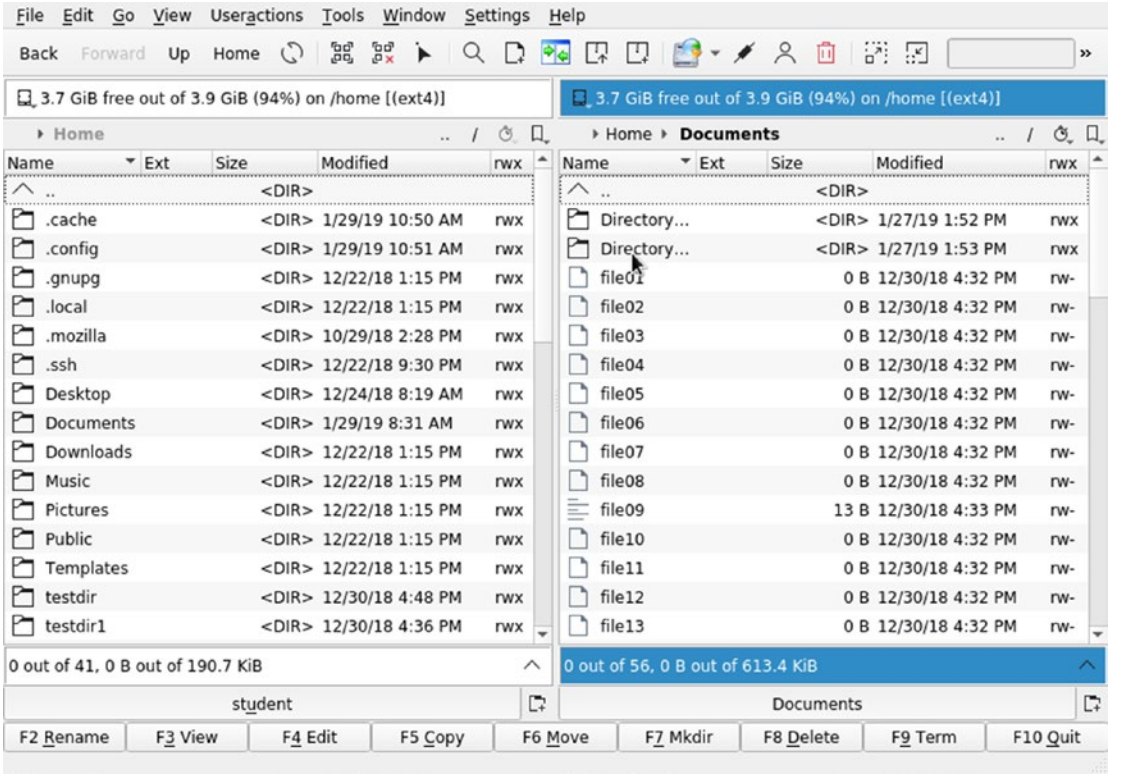


***Figure 2-15.*** *Krusader is much like Midnight Commander, but uses a GUI and provides significantly more flexibility*

The primary user interface for Krusader, much like that of Midnight Commander, is two text-mode panels, left and right, which each displays the contents of a directory. The top of each panel contains the name of the current directory for that panel. In addition, tabs can be opened for each panel and a different directory can be open in each tab. Navigation is accomplished with the arrow and tab keys or with the mouse. The Enter

key can be used to enter a highlighted directory or you can double-click the desired directory. Each tab and panel can be configured to show files in one of two different modes. In Figure 2-15, files are displayed in the detailed view which, in addition to the file name and an icon or preview, shows the file size, the date it was last modified, the owner, and the file permissions.

Along the very top of the Krusader graphical user interface are a menu bar and tool bar containing menu items for configuring Krusader and managing files. The bottom portion of the interface displays a line of function key labels; you can simply press the function key on your keyboard that corresponds to the function you want to perform.

By default Krusader saves the current tab and directory locations as well as other configuration items when you exit so that you will always return to the last configuration and set of directories when restarting the application. This configuration can be changed so that your home or some other directories are always the ones opened at startup.

## EXPERIMENT 2-2

As root, install Krusader.

```
[root@studentvm1 ~]# dnf install -y krusader
```

This command installed over 75 packages on my StudentVM1 virtual machine although that number may be significantly different for you. Krusader was written to integrate with the KDE desktop. Many of the packages are needed to support the KDE base functions of Krusader.

Start Krusader on the Xfce desktop: **Applications ➤ Accessories ➤ Krusader**.

Since this is the first time you will have used Krusader, it will show you a Welcome dialog and then take you through a dialog to help you perform a starting configuration. It is not necessary to make any changes at this time, so click the **OK** buttons and then the **Close** button to proceed directly to the Krusader window.

Krusader, like Midnight Commander, starts with two open panels. Switch between the panels using the **Tab** key or clicking with the mouse.

Highlight the Documents directory in the right panel and press the **Enter** key or just double-click it.

Change directories by double-clicking the desired directory. Moving up to the parent directory can be accomplished by double-clicking the double-dot (..) entry. You can also move to the parent of the directory in the highlighted panel by clicking the arrow icon on the tool bar.

Notice the function key assignments at the bottom of the Krusader window. These function keys provide functions similar to those of Midnight Commander. F1 will display some help. There are also function keys for move, copy, delete, and quit, among others. Simply press the corresponding function key on the keyboard or click the button to perform that function. You can also pop up a context menu by right-clicking a desired file and then taking one of the actions from the menu.

Figure 2-16 shows how to configure Krusader to display the embedded Konsole terminal session and the command line. Go ahead and configure those both now.

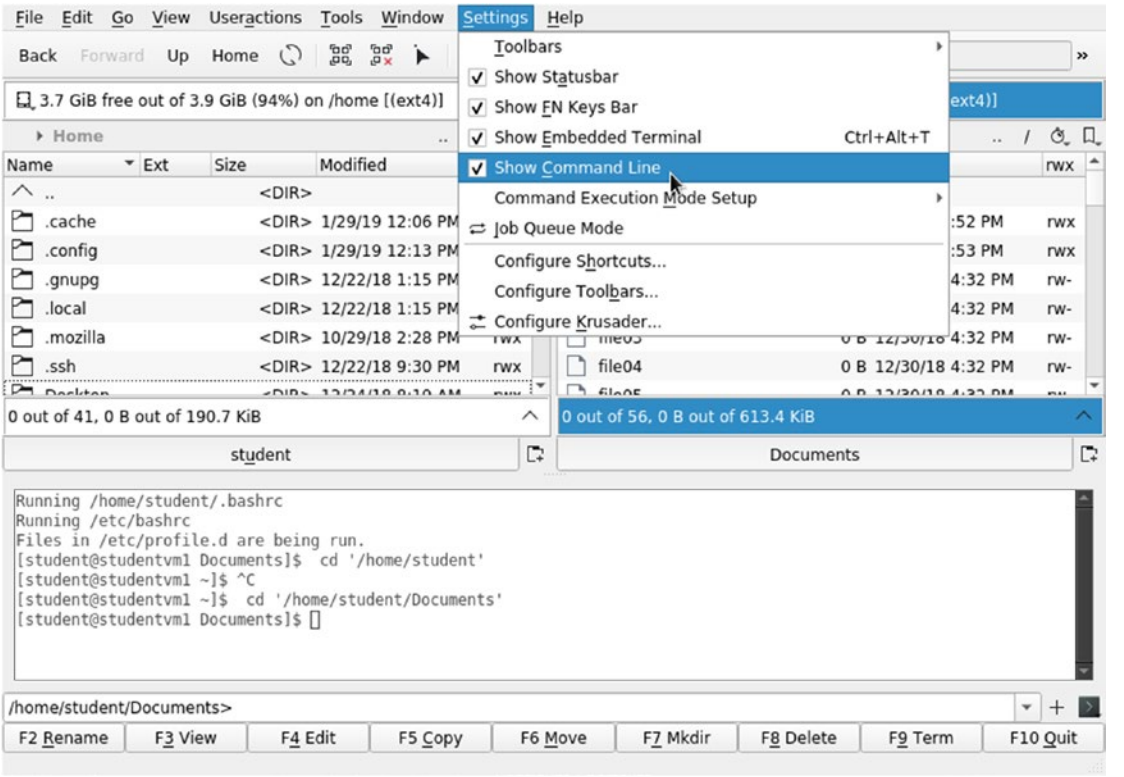


***Figure 2-16.*** *Configuring Krusader to show both the built-in command-line interface and the Konsole terminal window panel*

After having configured the terminal session and the simple command line, you can issue CLI commands simply by typing them; the CLI entry text box is at the bottom, just above the function key assignment line. The cursor there is always active while you are in navigation mode. To change the PWD of the current panel to the /tmp directory, type cd /tmp and press the **Enter** key, just as you would from the shell prompt.

Using the GUI, navigate to your ~/Documents directory. Highlight the file dmesg1.txt and press **F3**. This shows the contents of that file. Scroll up and down the file using the Page Up and Page Down keys or using the scrollbar. Click the **X** in the View tab to close the view of the file and return to the main Krusader window. In order to edit a file, the Kate GUI text editor would need to be installed.

Locate the dmesg2.txt or similar file and press **F8** to delete it. Click the Delete button to complete the deletion.

Take some time to explore Krusader on your own.

Press the F10 key to exit from Krusader.

---

I use Krusader as one of my main GUI file managers. It is powerful and easy to use, especially to someone familiar with Midnight Commander. The biggest drawback to Krusader is the large number of other KDE programs required to support it.

# Thunar

Thunar is a lightweight file manager that is the default for the Xfce desktop. Thunar, shown in Figure 2-17, has a single directory pane with which to work. It also has a sidebar for navigation. Thunar is a simple, decent file manager that is good for many beginners due to its simplicity.

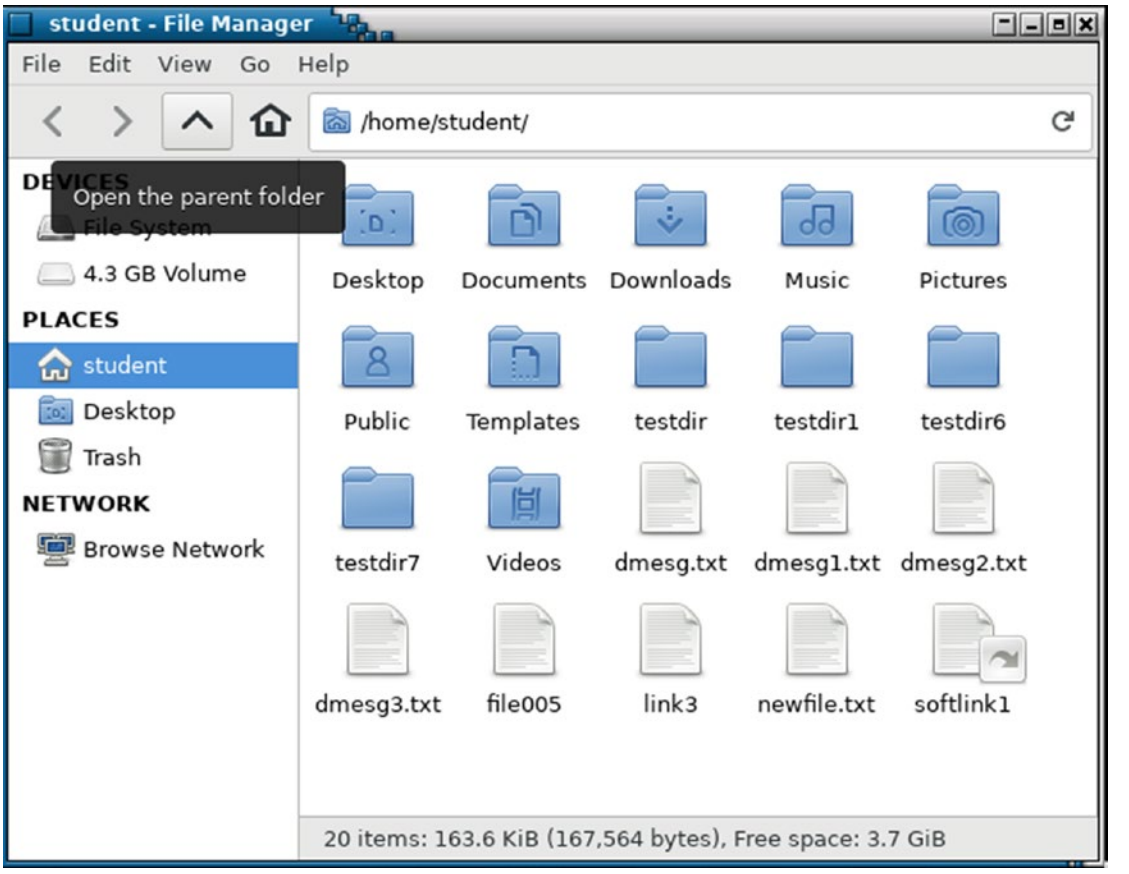


***Figure 2-17.*** *Thunar is the default file manager for the Xfce desktop*

The primary user interface for Thunar is fairly simple with a navigation sidebar and a single directory window in which to work. It does support multiple tabs but not splitting the panel into two. Navigation is accomplished with the arrow and tab keys or the mouse. The Enter key can be used to enter a highlighted directory.

# Dolphin

Dolphin is somewhat like Krusader. It can be configured for two directory navigation panels and it adds a sidebar that allows for easy filesystem navigation. It also supports tabs; however, when restarted, it always reverts to the default of one pair of directory panels that display your home directory.

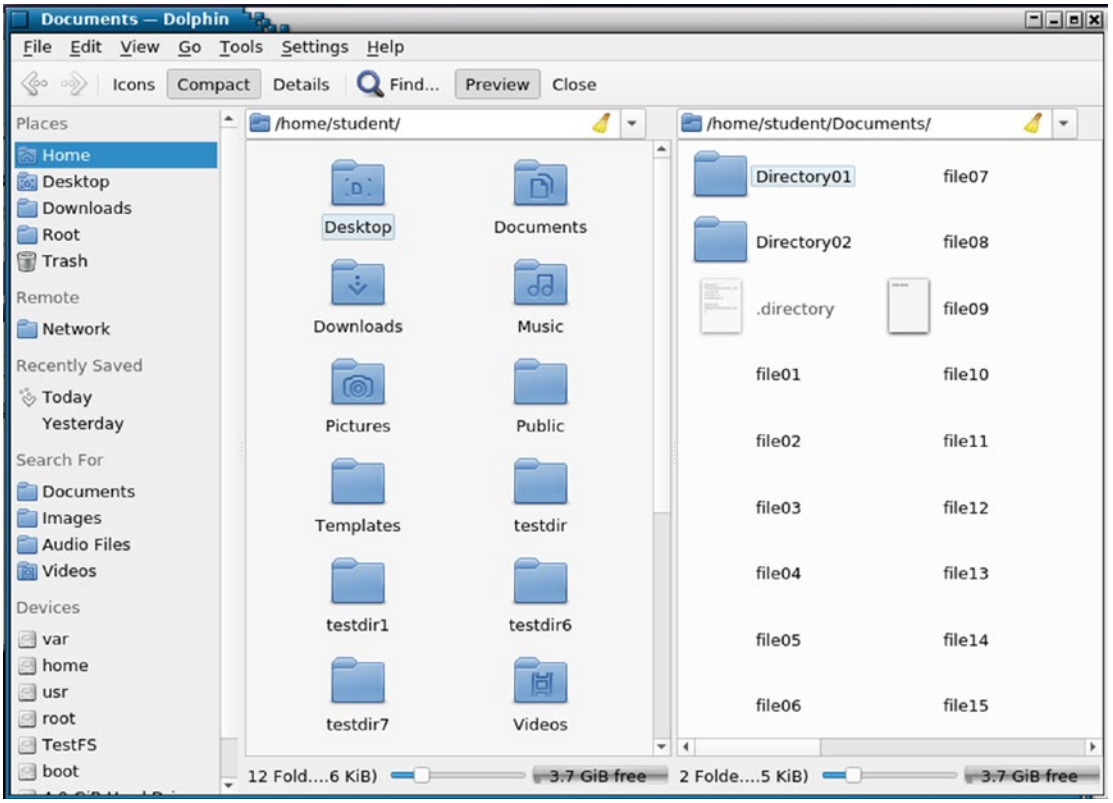


***Figure 2-18.*** *After some configuration, the Dolphin file manager uses two directory navigation panels and a navigation sidebar*

The primary user interface for Dolphin can be configured to be very similar to Krusader. Navigation is accomplished with the arrow and tab keys or the mouse. The Enter key can be used to enter a highlighted directory. Dolphin also supports expanding the directory trees (folders) in both the sidebar navigation panel and the directory panels.

Dolphin is not installed by default from the Xfce live image so it needs to be installed manually. Dolphin requires about 35 additional dependencies which are also installed. The package name is “dolphin” in case you want to install and test it. I did find that Dolphin has problems with display of some icons; you can see this in Figure 2-18. I did not try to fix this problem.

# XFE

XFE, seen in Figure 2-19, is one of the more interesting of the file managers as it has an interface all its own and is a bit more flexible than some of the other file managers.

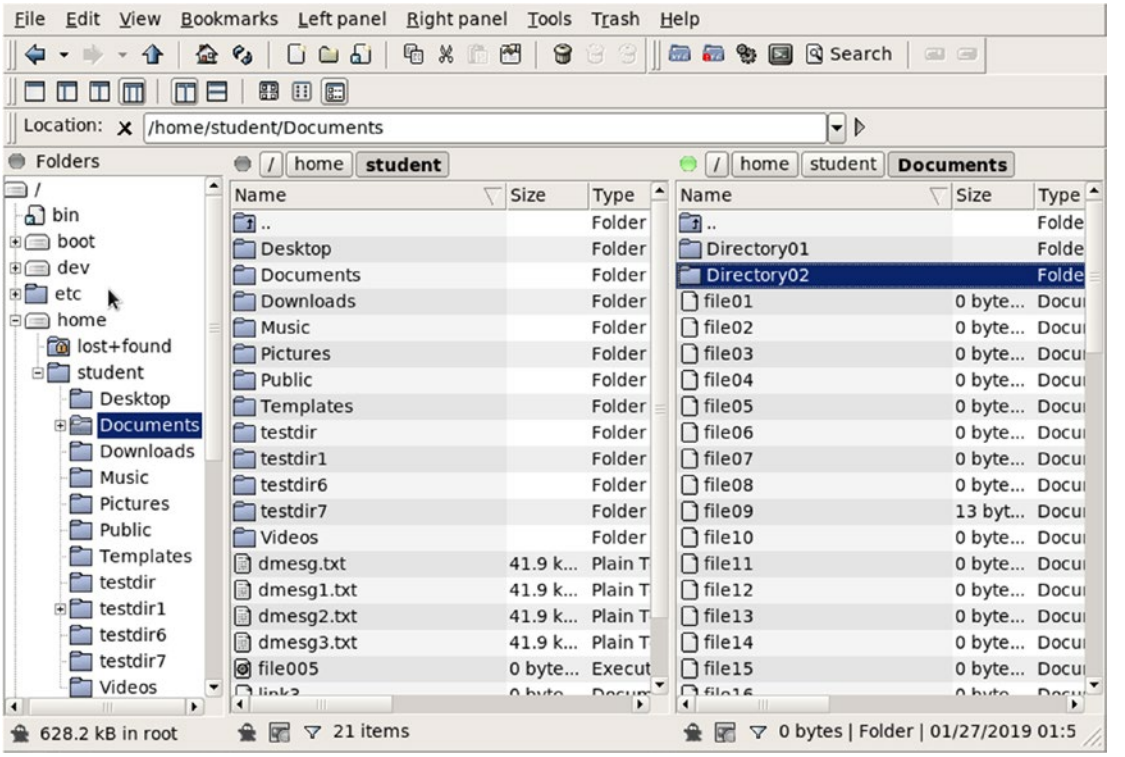


***Figure 2-19.*** *The XFE file manager*

XFE may be configured to display one or two directory panels and the navigation bar which is optional. It performs all the expected drag-and-drop functions, but it requires some manual configuration to link some desired applications like LibreOffice with specific file types. It has a reasonable set of configuration options, but nowhere near those of Krusader.

XFE is also quite restrictive about retaining its own set of "themes" and has no option to use the desktop color scheme, icons, decorations, or widgets.

Install the “xfe” package to experiment with this file manager.

## Chapter summary

There are many file managers, and that one which I have not covered may already be your favorite. Your choice of file manager should be the one that works best for you. GNU/Linux provides several viable choices and one will most likely meet most of your needs. If your favorite does not meet your needs for a particular task, you can always use the one that does.

All of these file managers are free of charge and distributed under some form of open source license. All are available from common, trusted repositories for Fedora and CentOS.

## Exercises

Perform these exercises to complete this chapter:

1. For single-panel GUI file managers like Thunar, is it possible to use a second instance to drag and drop actions such as copy or move?
2. Configure the right pane of Midnight Commander to view the Info of the file highlighted by the selection bar in the left panel. What information does this Info panel display about the filesystem?
3. Is this configuration, you have just done in Exercise 2 persistent between restarts of MC?
4. Continuing with Midnight Commander, switch the two panels so that the Info panel is on the right.
5. Convert the MC Info panel to show a directory tree. Are the subdirectories displayed when the directory tree is first entered?

CHAPTER 3

# Everything Is a File

## Objectives

In this chapter you will learn

- The definition of a file
- To understand what “everything is a file” really means and why it is important
- The implications of “everything is a file”
- How to use common Linux file management tools to access hardware as files

This is one of the most important concepts that make Linux especially flexible and powerful: everything is a file. That is, everything can be the source of a data stream, the target of a data stream, or in many cases both. In this chapter you will explore what “everything is a file” really means and learn to use that to advantage as a SysAdmin.

> *The whole point with "everything is a file" is ... the fact that you can use common tools to operate on different things.*
>
> —Linus Torvalds in an email.

## What is a file?

Here is a trick question for you. Which of the following are files?

- Directories
- Shell scripts
- Running terminal emulators




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_3

- LibreOffice documents
- Serial ports
- Kernel data structures
- Kernel tuning parameters
- Hard drives – /dev/sda
- /dev/null
- Partitions – /dev/sda1
- Logical volumes (LVM) – /dev/mapper/volume1-tmp
- Printers
- Sockets

To Unix and Linux, they are all files and that is one of the most amazing concepts in the history of computing. It makes possible some very simple yet powerful methods for performing many administrative tasks that might otherwise be extremely difficult or impossible.

Linux handles almost everything as a file. This has some interesting and amazing implications. This concept makes it possible to copy an entire hard drive, boot record included, because the entire hard drive is a file, just as are the individual partitions.

"Everything is a file" is possible because all devices are implemented by Linux as these things called device files. Device files are not device drivers, rather they are gateways to devices that are exposed to the user.

# Device files

Device files are technically known as device special files.[1] Device files are employed to provide the operating system and the users an interface to the devices that they represent. All Linux device files are located in the /dev directory, which is an integral part of the root (/) filesystem because they must be available to the operating system during early stages of the boot process – before other filesystems are mounted.

[1]Wikipedia, Device File, https://en.wikipedia.org/wiki/Device_file

## Device file creation

Over the years, chaos overtook the /dev directory with huge numbers of mostly unneeded devices. The udev daemon was created to simplify this problem. Understanding how udev works is key to dealing with devices, especially hot-plug devices and how they can be managed.

The /dev/directory has always been the location for the device files in all Unix and Linux operating systems. In the past, device files were created at the time the operating system was created. This meant that all possible devices that might ever be used on a system needed to be created in advance. In fact, tens of thousands of device files needed to be created to handle all of the possibilities. It became very difficult to determine which device file actually related to a specific physical device or if one were missing.

## udev simplification

udev is designed to simplify this problem by creating entries in /dev only for those devices that actually currently exist at boot time or which have a high probability of actually existing on the host. This significantly reduces the total number of device files required.

In addition, udev assigns names to devices when they are plugged into the system, such as USB storage and printers, and other non-USB types of devices as well. In fact, udev treats all devices as plug and play (PnP), even at boot time. This makes dealing with devices consistent at all times, whether at boot time or when they are hot-plugged later. It is not necessary for us as SysAdmins to do anything else for the device files to be created. The Linux kernel takes care of everything. It is only possible to mount the partition in order to access its contents after the device file such as /dev/sdb1 has been created.

Kernel developer and maintainer Greg Kroah-Hartman, one of the creators of udev, has written a paper[2] that provides some insight into the details of udev and how it is supposed to work. Note that udev has matured since the article was written and some things have changed, such as the udev rule locations and structure. Regardless, this paper provides some deep and important insight into udev and current device naming strategies which I'll attempt to summarize in this chapter.

[2]Greg Kroah-Hartman, *Linux Journal*, Kernel Korner – udev – Persistent Naming in User Space, www.linuxjournal.com/article/7316

## Naming rules

In modern versions of Fedora and CentOS, udev stores its default naming rules in files in the /usr/lib/udev/rules.d directory and its local rules and configuration files in the /etc/udev/rules.d directory. Each file contains a set of rules for a specific device type. CentOS 6 and earlier stored the global rules in /lib/udev/rules.d/. The location of the udev rules files may be different on your distribution.

In earlier versions of udev, there were many local rule sets created, including a set for network interface card (NIC) naming. As each NIC was discovered by the kernel and renamed by udev for the very first time, a rule was added to the rule set for the network device type. This was initially done to ensure consistency before names had changed from "ethX" to more consistent ones.

**RULE CHANGE BLUES**

One of the main consequences of using udev for persistent plug and play naming is that it makes things much easier for the average non-technical user. This is a good thing in the long run; however, there have been migration problems, and many SysAdmins were – and still are – not happy with these changes.

The rules changed over time and there were at least three significantly different naming conventions for network interface cards. That naming disparity caused a great deal of confusion, and many configuration files and scripts had to be rewritten multiple times during the period of these changes.

For example, the name of a NIC that was originally eth0 would have changed from that to em1 or p1p2 and finally to eno1. I wrote an article[3] on my web site that goes into some detail about these naming schemes and the reasons behind them.

Now that udev has multiple consistent default rules for determining device names, especially for NICs, storing the specific rules for each device in local configuration files is no longer required to maintain that consistency.

[3]David Both, Network Interface Card (NIC) name assignments, www.linux-databook.info/?page_id=4243

# Device data flow

Let's look at the data flow of a typical command to visualize how device special files work. Figure 3-1 illustrates a simplified data flow for a simple command. Issuing the **`# cat /etc/resolv.conf`** command from a GUI terminal emulator such as Konsole or xterm causes the resolv.conf file to be read from the disk with the disk device driver handling the device-specific functions such as locating the file on the hard drive and reading it. The data is passed through the device file and then from the command to the device file and device driver for pseudo-terminal 6 where it is displayed in the terminal session.

Of course the output of the **`cat`** command could have been redirected to a file in the following manner, **`cat /etc/resolv.conf > /etc/resolv.bak`**, in order to create a backup of the file. In that case the data flow on the left side of Figure 3-1 would remain the same, while the data flow on the right would be through the /dev/sda2 device file, the hard drive device driver, and then back onto the hard drive in the /etc directory as the new file, resolv.bak.

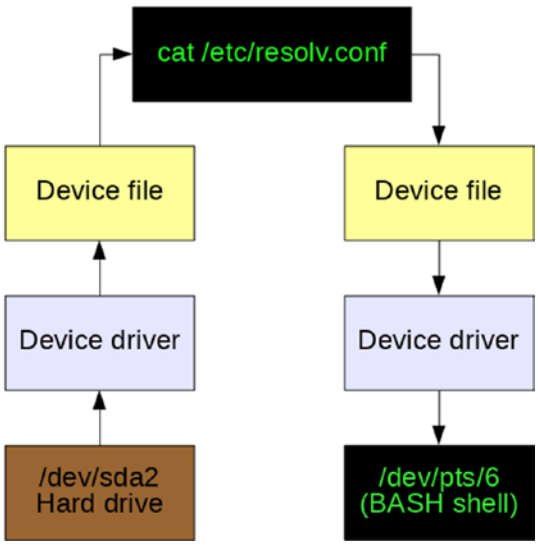


***Figure 3-1.*** *Simplified data flow with device special files*

These device special files make it very easy to use standard streams (STDIO) and redirection to access any and every device on a Linux or Unix computer. They provide a consistent and easy to access interface to every device. Simply directing a data stream to a device file sends the data to that device.

One of the most important things to remember about these device special files is that they are not device drivers. They are most accurately described as portals or gateways to the device drivers. Data is passed from an application or the operating system to the device file which then passes it to the device driver which then sends it to the physical device.

By using these device files which are separate from the device drivers, it is possible for users and programs to have a consistent interface to every device on the host computer. This is how common tools can be used to operate on different things as Linus says.

The device drivers are still responsible for dealing with the unique requirements of each physical device. That is, however, outside the scope of this book.

# Device file classification

Device files can be classified in at least two ways. The first and most commonly used classification is that of the type of data stream commonly associated with the device. For example, tty and serial devices are considered to be character based because the data stream is transferred and handled one character or byte at a time. Block-type devices such as hard drives transfer data in blocks, typically a multiple of 256 bytes.

Let's take a look at the /dev/directory and some of the devices in it.

**EXPERIMENT 3-1**

This experiment should be performed as the user student.

Open a terminal session and display a long listing of the /dev/directory.

```
[student@studentvm1 ~]$ ls -l /dev | less
<snip>
brw-rw----. 1 root    disk      8,   0 Jan 30 06:53 sda
brw-rw----. 1 root    disk      8,   1 Jan 30 06:53 sda1
brw-rw----. 1 root    disk      8,   2 Jan 30 06:53 sda2
brw-rw----. 1 root    disk      8,  16 Jan 30 06:53 sdb
brw-rw----. 1 root    disk      8,  17 Jan 30 06:53 sdb1
brw-rw----. 1 root    disk      8,  18 Jan 30 06:53 sdb2
brw-rw----. 1 root    disk      8,  19 Jan 30 06:53 sdb3
brw-rw----. 1 root    disk      8,  32 Jan 30 06:53 sdc
```

```
<snip>
crw-rw-rw-. 1 root    tty       5,   0 Jan 30 06:53 tty
crw--w----. 1 root    tty       4,   0 Jan 30 06:53 tty0
crw--w----. 1 root    tty       4,   1 Jan 30 11:53 tty1
crw--w----. 1 root    tty       4,  10 Jan 30 06:53 tty10
crw--w----. 1 root    tty       4,  11 Jan 30 06:53 tty11
crw--w----. 1 root    tty       4,  12 Jan 30 06:53 tty12
<snip>
```

The results from this command are too long to show here in full, but you will see a list of device files with their file permissions and their major and minor identification numbers.

The voluminous output of the **`ls -l`** command is piped through the **`less`** utility to allow you to page through the results; use the Page Up, Page Down, and up and down arrow keys to move around. Type **`q`** to quit and get out of the `less` display.

The pruned listing of device files shown in Experiment 3-1 are just a few of the ones in the /dev/directory on my StudentVM1 virtual machine. The ones on your VM should be very similar if not identical. They represent disk and tty type devices among many others. Notice the leftmost character of each line in the output. The ones that have a "b" are block-type devices and the ones that begin with "c" are character devices.

The more detailed and explicit way to identify device files is using the device major and minor numbers. The disk devices have a major number of 8 which designates them as SCSI block devices. Note that all parallel ATA (PATA)[4] and serial ATA (SATA)[5] hard drives and SSDs have been managed by the SCSI subsystem because the old ATA subsystem was deemed many years ago as not maintainable due to the poor quality of its code. As a result, hard drives that would previously be designated as "hd[a-z]" are now referred to as "sd[a-z]".

You can probably infer the pattern of disk drive minor numbers in the small sample shown earlier. Minor numbers 0, 16, 32, and so on up through 240 are the whole disk numbers. So major/minor 8/16 represents the whole disk /dev/sdb and 8/17 is the device file for the first partition, /dev/sdb1. Numbers 8/34 would be /dev/sdc2.

[4]Wikipedia, Parallel ATA, https://en.wikipedia.org/wiki/Parallel_ATA

[5]Wikipedia, Serial ATA, https://en.wikipedia.org/wiki/Serial_ATA

The tty device files in the preceding list are numbered a bit more simply from tty0 through tty63. I find the number of tty devices a little incongruous because the whole point of the new udev system is to create device files for only those devices that actually exist; I am not sure why it is being done this way. The device files on your host should have a timestamp that is the same as the last boot time.

The Linux Allocated Devices[6] file at Kernel.org is the official registry of device types and major and minor number allocations. It can help you understand the major/minor numbers for all currently defined devices.

# Fun with device files

Let's take a few minutes now and have some fun with some of these device files. We will perform a couple fun experiments that illustrate the power and flexibility of the Linux device files.

Most Linux distributions have multiple virtual consoles, 1 through 7, that can be used to log in to a local console session with a shell interface. These can be accessed using the key combinations HostKey-F1 for console 1, HostKey-F2 for console 2, and so on. Virtual consoles were introduced in Volume 1, Chapter 7, of this course. The default HostKey is the right Control key, but I have reconfigured mine to be the left Win key, a.k.a. the super key, because I find it easier. You can change the default HostKey with the VirtualBox manager.

**EXPERIMENT 3-2**

In this experiment we will show that simple commands can be used to send data between devices, in this case, different console and terminal devices. Perform this experiment as the student user.

On the StudentVM1 desktop window, press **HostKey-F2** to switch to console 2. On some distributions like Fedora, the login information includes the tty (Teletype) device associated with this console, but some do not. It should be tty2 because you are in console 2.

[6]Kernel.org, Linux Allocated Devices, www.kernel.org/doc/html/v4.11/admin-guide/devices.html

Log in to console 2 as the student user. Then use the **who am i** command – yes, just like that, with spaces – to determine which tty device is connected to this console.

```
[student@studentvm1 ~]$ who am i
student  tty2         2019-01-30 15:32
```

This command also shows the date and time that the user on the console logged in.

Before we proceed any further with this experiment, let's look at a listing of the tty2 and tty3 devices in /dev. We do that by using a set [23] so that only those two devices are listed.

```
[student@studentvm1 ~]$ ls -l /dev/tty[23]
crw--w----. 1 student tty 4, 2 Jan 30 15:39 /dev/tty2
crw--w----. 1 root    tty 4, 3 Jan 30 06:53 /dev/tty3
```

There are a large number of tty devices defined at boot time, but we do not care about most of them for this experiment, just the tty2 and tty3 devices. As device files there is nothing special about them; they are simply character type devices; note the “c” in the first column of the results. We will use these two TTY devices for this experiment. The tty2 device is attached to virtual console 2, and the tty3 device is attached to virtual console 3.

Press **HostKey-F3** to switch to console 3 and log in again as the student user. Use the `who am i` command again to verify that you really are on console 3 and then enter the echo command.

```
[student@studentvm1 ~]$ who am i
student  tty3         2019-01-30 15:38
[student@studentvm1 ~]$ echo "Hello world" > /dev/tty2
```

Press **HostKey-F2** to return to console 2. The string "Hello world" (without quotes) should be displayed on console 2.

This experiment can also be performed with terminal emulators on the GUI desktop. Terminal sessions on the desktop use pseudo-terminal devices in the /dev tree, such as /dev/pts/1, where pts stands for “pseudo-terminal session.”

Return to your graphical desktop using HostKey-F1. Open at least two terminal sessions on the GUI desktop using Konsole, Tilix, Xterm, or your other favorite graphical terminal emulator. You may open several if you wish. Determine which pseudo-terminal device files they are connected to with the **who am i** command and then choose one pair of terminal emulators to work with for this experiment. Use one to send a message to the another with the echo command.

```
[student@studentvm1 ~]$ who am i
student  pts/9         2017-10-19 13:21 (192.168.0.1)
```

However, it is possible that you will get no result from the **who am i** command. This occurs because **who am i** only seems to work on login terminal sessions and not on a non-login session such as one started from the desktop. So a virtual console session or a remote SSH login session would work with this. But there are at least two ways to circumvent this – as is usual in Linux.

We will use the **w** command. The **w** command lists the tasks being run on each terminal session so the terminal session that shows **w** in the WHAT column is the one you are looking for. In my case it is pts/6, as shown in the following.

```
[student@studentvm1 ~]$ w
 08:47:38 up 1 day, 20:12,  6 users,  load average: 0.11, 0.06, 0.01
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
student  tty1      Wed12    2days 14.16s  0.06s /bin/sh /etc/xdg/xfce4/
xinitrc -- vt
root     pts/1     Thu17   10:50m  0.03s  0.03s /bin/bash
root     pts/2     Thu17   15:18m  0.02s  0.02s /bin/bash
root     pts/3     Thu17   10:48m  0.02s  0.02s /bin/bash
student  pts/5     08:45   47.00s  0.02s  0.00s less
student  pts/6     08:45    0.00s  0.03s  0.00s w
[student@studentvm1 ~]$
[student@studentvm1 ~]$ echo "Hello world" > /dev/pts/5
```

On my test host, I sent the text “Hello world” from /dev/pts/6 to /dev/pts/5. Your terminal devices will be different from the ones I have used on my test VM. Be sure to use the correct devices for your environment for this experiment.

Another interesting experiment is to print a file directly to the printer using the cat command. If you do not have a printer attached to your physical host that is available to the VM, you can skip Experiment 3-3.

## EXPERIMENT 3-3

This experiment should be performed as the student user on StudentVM1.

If you have a USB printer available, plug it into the physical host. Then use the StudentVM1 window’s menu bar to open the **Devices ➤ USB** menu and then add a check to the printer in the listed USB devices in order to make it available to your VM.

You may need to determine which device is your printer. If your printer is a USB printer which almost all are these days, look in the /dev/usb directory for lp0 which is usually the default printer. You may find other printer device files in that directory as well.

I used LibreOffice Writer to create a short document which I then exported as a PDF file, test.pdf. Any Linux word processor will do so long as it can export to the PDF format.

We will assume that your printer device is /dev/usb/lp0, and that your printer can print PDF files directly, as most can. Be sure to use a PDF file and change the name test.pdf in the command to the name of your own file.

```
[student@studentvm1 ~]$ cat test.pdf > /dev/usb/lp0
```

This command should print the PDF file test.pdf on your printer.

The /dev directory contains some very interesting device files that are portals to hardware that one does not normally think of as a device like a hard drive or display. For one example, system memory – RAM – is not something that is normally considered as a "device," yet /dev/mem is the device special file through which direct access to memory can be achieved.

## EXPERIMENT 3-4

This experiment must be run as the root user. Because you are only reading the contents of memory, this experiment poses little danger.

If a root terminal session is not already available, open a terminal emulator session and log in as root. The next command will dump the first 200K of RAM to STDOUT.

```
[root@studentvm1 ~]# dd if=/dev/mem bs=2048 count=100
```

It may not look like that much and what you do see will be unintelligible. To make it a bit more intelligible – to at least display the data in a decent format that might be interpreted by an expert – pipe the output of the previous command through the **od** utility.

```
[root@studentvm1 ~]# dd if=/dev/mem bs=2048 count=100 | od -c
```

Root has more access to read memory than a non-root user, but most memory is protected from being written by any user, including root.

The **dd** command provides significantly more control than simply using the **cat** command to dump all of memory, which I have also tried. The **dd** command provides the ability to specify how much data is read from /dev/mem and would also allow me to specify the point at which to start reading data from memory. Although some memory was read using the **cat** command, the kernel eventually responded with the error in Figure 3-2.

You can also log in as a non-root user, student, and try this command. You will get an error message because the memory you are trying to access does not belong to your user. This is a memory protection feature of Linux that keeps other users from reading or writing memory that does not belong to them.

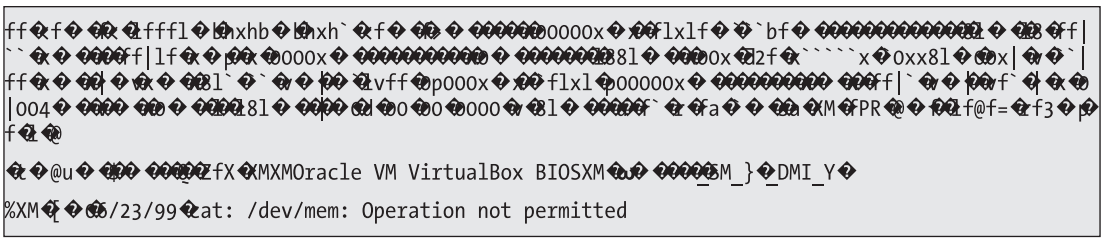


***Figure 3-2.*** *The error on the last line was displayed when the* ***cat*** *command attempted to dump protected memory to STDOUT*

These memory errors mean that the kernel is doing its job by protecting memory that belongs to other processes which is exactly how it should work. So, although you can use /dev/mem to display data stored in RAM memory, access to most memory space is protected and will result in errors. Only that virtual memory which is assigned by the kernel memory manager to the bash shell running the **dd** command should be accessible without causing an error. Sorry, but you cannot snoop in memory that does not belong to you unless you find a vulnerability to exploit.

Many types of malware depend upon privilege escalation to allow them to read the contents of memory that they would not normally be able to access. This allows the malware to find and steal personal data such as account numbers, user ID, and stored passwords. Fortunately, Linux protects against memory access by non-root users. It

also protects against privilege escalation. But even Linux security is not perfect. It is important to install security patches to protect against vulnerabilities that allow privilege escalation. You should also be aware of human factors such as the tendency people have to write down their passwords, but that is all another book.[7]

You can now see that memory is also considered to be a file and can be treated as such using the memory device file.

## Randomness, zero, and more

There are some other very interesting device files in /dev. The device special files null, zero, random, and urandom are not associated with any physical devices. These device files provide sources of zeros, nulls, and random numbers.

The null device /dev/null can be used as a target for the redirection of output from shell commands or programs so that they are not displayed on the terminal.

**EXPERIMENT 3-5**

I frequently use /dev/null in my bash scripts to prevent users from being presented with output that is irrelevant or that might be confusing to them. Enter the following command to redirect the output to the null device. Nothing will be displayed on the terminal. The data is just dumped into the big bit bucket in the sky.

```
[student@studentvm1 ~]$ echo "Hello world" > /dev/null
```

There is really no visible output from the /dev/null because the null device simply returns an end of file (EOF) character. Note that the byte count is zero. The null device is much more useful as a place to redirect unwanted output so that it is removed from the data stream.

[7]Apress has a number of good books on security at www.apress.com/us/security.

The /dev/random and /dev/urandom devices are both useful as data stream sources. As their names imply, they both produce essentially random output – not just numbers but any and all byte combinations. The /dev/urandom device produces a deterministic[8] stream of random output and is very fast while /dev/random produces a non-deterministic[9] stream but is slower.

## EXPERIMENT 3-6

Use this command to view typical output from /dev/urandom. You can use Ctrl-c to break out.

```
[student@studentvm1 ~]$ cat /dev/urandom
,3��  VwM
N�g�/�l�○�:�'␣'��!�ې|R��[塚t��Z��F.:H�7�,��
��z/��|�7q�Sp�"�(l_c��π��-
��������ś�Y���D^5�i8��"%���&ŋ|C9!y���f�5bPp;��C
��x��1���U��3~��ↄ
<snip>
```

I have shown only a part of the data stream from the command, but it should give you a sense for what you should see on your system.

You could also pipe the output of Experiment 3-6 through the **od** (Octal Display) command to make it a little more human readable just for this experiment. That makes little sense for most real-world applications because it is, after all, random data.

The man page for **od** shows that it can be used to obtain data directly from a file as well as specify the amount of data to be read.

[8]Deterministic means the output is determined by a known algorithm and uses a seed string as a starting point. Each unit of output is dependent upon the previous output and the algorithm, so if you know both the seed and the algorithm, the entire data stream can be reproduced. As a result, it is possible, although difficult, for a hacker to reproduce the output if the original seed is known.

[9]Non-deterministic results are not dependent upon the previous data in the random data stream. Thus, they are more truly random than if they were deterministic.

## EXPERIMENT 3-7

In this case I have used -N 128 to limit the output to 128 bytes.

```
[student@studentvm1 ~]$ od /dev/urandom -N 128
0000000 043514 022412 112660 052071 161447 057027 114243 061412
0000020 154627 105675 154470 110352 135013 127206 103057 136555
0000040 033417 011054 014334 040457 157056 165542 027255 121710
0000060 125334 065600 165447 165245 020756 101514 042377 132156
0000100 116024 027770 000537 014743 170561 011122 173454 102163
0000120 074301 104771 123476 054643 105211 151753 166617 154313
0000140 103720 147660 012644 037363 077661 076453 104161 033220
0000160 056501 001771 113557 075046 102700 043405 132046 045263
0000200
```

The **dd** command could also be used to specify a limit to the amount of data taken from the [u]random devices, but it cannot directly format the data.

The /dev/random device file produces non-deterministic random output, but it produces output more slowly. This output is not determined by an algorithm that is dependent only upon the previous number that was generated, but it is generated in response to keystrokes and mouse movements. This method makes it far more difficult to duplicate a specific series of random numbers. Use the cat command to view some of the output from the /dev/random device file. Try moving the mouse to see how it affects the output.

The random data generated from /dev/random and /dev/urandom, regardless of how it is read from those devices, is usually redirected to a file on some storage media or to STDIN of another program. Random data seldom needs to be viewed by the SysAdmin, developer, or the end user. But it does make a good demonstration for this experiment.

As its name implies, the /dev/zero device file produces an unending string of zeros as output. Note that these are octal zeros and not the ASCII character zero (0).

**EXPERIMENT 3-8**

Use the dd command to view some output from the /dev/zero device file. Note that the byte count for this command is non-zero.

```
[student@studentvm1 ~]$ dd if=/dev/zero  bs=512 count=500 | od -c
0000000  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
500+0 records in
500+0 records out
256000 bytes (256 kB, 250 KiB) copied, 0.00126996 s, 202 MB/s
0764000
```

# Back up the master boot record

Consider, for example, the simple task of making a backup of the master boot record (MBR) of a hard drive. I have had, on occasion, needed to restore or recreate my MBR, particularly the partition table. Recreating it from scratch is very difficult. Restoring it from a saved file is easy. So let's back up the boot record of the hard drive.

Note that all of the experiments in this section must be performed as root.

**EXPERIMENT 3-9**

Perform this experiment as the root user. We are going to create a backup of the master boot record (MBR), but we will not attempt to restore it.

The **dd** command must be run as root because for security reasons non-root users do not have access to the hard drive device files in the /dev directory. The bs value is not what you might think; it stands for block size. Count is the number of blocks to read from the source file.

```
[root@studentvm1 ~]# dd if=/dev/sda of=/tmp/myMBR.bak bs=512 count=1
```

This command creates a file, myMBR.bak, in the /tmp directory. The file is 512 bytes in size and contains the contents of the MBR including the bootstrap code and partition table. Look at the contents of the file you just created.

```
[root@studentvm1 ~]# cat /tmp/myMBR.bak
```

The output from dmesg looks like this.

```
[68258.690964] ata4: SATA link up 3.0 Gbps (SStatus 123 SControl 300)
[68258.691148] ata4.00: ATA-6: VBOX HARDDISK, 1.0, max UDMA/133
[68258.691173] ata4.00: 4194304 sectors, multi 128: LBA48 NCQ (depth 32)
[68258.691300] ata4.00: configured for UDMA/133
[68258.691499] scsi 3:0:0:0: Direct-Access     ATA      VBOX
HARDDISK    1.0  PQ: 0 ANSI: 5
[68258.692828] sd 3:0:0:0: [sdd] 4194304 512-byte logical blocks: (2.15
GB/2.00 GiB)
[68258.692876] sd 3:0:0:0: [sdd] Write Protect is off
[68258.692897] sd 3:0:0:0: [sdd] Mode Sense: 00 3a 00 00
[68258.692909] sd 3:0:0:0: [sdd] Write cache: enabled, read cache: enabled,
doesn't support DPO or FUA
[68258.693888] sd 3:0:0:0: Attached scsi generic sg4 type 0
[68258.700966] sd 3:0:0:0: [sdd] Attached SCSI disk
```

The new drive will also show up with the **lsblk** command as /dev/sdd.

Look at the boot record to provide a basis for later comparison. Note that there is no data in the boot record of this new drive.

```
[root@studentvm1 ~]# dd if=/dev/sdd bs=512 count=1
1+0 records in
1+0 records out
512 bytes copied, 0.000348973 s, 1.5 MB/s
```

Create a partition that fills the entire 2GB of the virtual drive and save the new partition table.

```
[root@studentvm1 ~]# fdisk /dev/sdd

Welcome to fdisk (util-linux 2.32.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x39dfcf64.

Command (m for help): p
Disk /dev/sdd: 2 GiB, 2147483648 bytes, 4194304 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x39dfcf64

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): <press Enter>

Using default response p.
Partition number (1-4, default 1): <press Enter>
First sector (2048-4194303, default 2048): <press Enter>
Last sector, +sectors or +size{K,M,G,T,P} (2048-4194303, default 4194303):
<press Enter>

Created a new partition 1 of type 'Linux' and of size 2 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

Check the boot record again, which is no longer completely empty.

```
[root@studentvm1 ~]# dd if=/dev/sdd bs=512 count=1 | od -c
1+0 records in
1+0 records out
512 bytes copied, 0.0131589 s, 38.9 kB/s
0000000  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0000660  \0  \0  \0  \0  \0  \0  \0  \0   f 222 371 261  \0  \0  \0
0000700   !  \0 203 025   P 005  \0  \b  \0  \0  \0 370   ?  \0  \0  \0
0000720  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0000760  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0   U 252
0001000
```

Verify that the partition is /dev/sdd1 and then create an EXT4 filesystem on the partition.

```
[root@studentvm1 ~]# mkfs -t ext4 /dev/sdd1
mke2fs 1.44.3 (10-July-2018)
Creating filesystem with 524032 4k blocks and 131072 inodes
Filesystem UUID: 3e031fbf-99b9-42e9-a920-0407a8b34513
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
```

Mount the new filesystem on the /mnt mount point and store some data on the device. This will verify that the new filesystem is working as it should.

```
[root@studentvm1 ~]# mount /dev/sdd1 /mnt ; ll /mnt
total 16
drwx------. 2 root root 16384 Jan 31 08:08 lost+found
[root@studentvm1 ~]# dmesg > /mnt/testfile001 ; ll /mnt
total 60
drwx------. 2 root root 16384 Jan 31 08:08 lost+found
-rw-r--r--. 1 root root 44662 Jan 31 08:12 testfile001
[root@studentvm1 ~]#
```

Copy the MBR.

```
[root@studentvm1 ~]# dd if=/dev/sdd of=/tmp/sddMBR.bak bs=512 count=1
1+0 records in
1+0 records out
512 bytes copied, 0.0171773 s, 29.8 kB/s
[root@studentvm1 ~]# cat /tmp/sddMBR.bak
f��  !��?U[root@studentvm1 ~]#
```

Now is the fun part. We unmount the partition, overwrite the MBR of the device with one 512-byte block of random data, then view the new content of the MBR to verify the change.

```
[root@studentvm1 ~]# umount /mnt
[root@studentvm1 ~]# dd if=/dev/urandom of=/dev/sdd bs=512 count=1
512+0 records in
512+0 records out
```

```
262144 bytes (262 kB, 256 KiB) copied, 0.10446 s, 2.5 MB/s
[root@studentvm1 ~]# dd if=/dev/sdd bs=512 count=1
_Cv3X??qQ??????????????4p|?
kn??rX?-?
        ?
         ?N???Y??9??]?i???\?TXSqy4?AK?_?o{j?l????p?\
A?u?-w??3?#99?]?k???K'?(?Q?,1O?H?jp
?????a!?)?O?^o?]?y??S?B?IAu3S??QU???}??????
                                        ?9I?A??IBQ??ZZ?3
?H
  ??x?0?_PX?>?.?m???\?
???z?QfYU??c?f?s?hW?yvR????/???m?m?????T)?_?????>?J?Z?
?Xv2?Oqu[,????t?????m)?a?5p????j??*??K??Z{??8?????#(U
Oh8*V??
?<N????7??'#4???G:????+?I??T???z?9t??~??i!????."=??
?c?,7:?v??l<?(??U?P*
w?4?+\???Q??????V???1+0 records in
1+0 records out
512 bytes copied, 0.000603043 s, 849 kB/s
[root@studentvm1 ~]#
```

Let's try a couple more things to test out this state of affairs before we restore this MBR. First we use `fdisk` to verify that the USB drive no longer has a partition table, which means that the MBR has been overwritten.

```
[root@studentvm1 ~]# fdisk -l /dev/sdd
Disk /dev/sdd: 2 GiB, 2147483648 bytes, 4194304 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
[root@studentvm1 ~]#
```

An attempt to mount the original partition will fail. The error message indicates that the special device does not exist. This shows that most of the special device files are created and removed as necessary, on demand.

```
[root@studentvm1 ~]# mount /dev/sdd1 /mnt
mount: /mnt: special device /dev/sdd1 does not exist.
```

It is time to restore the boot record you backed up earlier. Because you used the **dd** command to carefully overwrite with random data only the MBR which contains the partition table for the drive, all of the other data remains intact. Restoring the MBR will make it available again. Restore the MBR, view the MBR on the device, then mount the partition and list the contents.

```
[root@studentvm1 ~]# dd if=/tmp/sddMBR.bak of=/dev/sdd
1+0 records in
1+0 records out
512 bytes copied, 0.0261115 s, 19.6 kB/s

[root@studentvm1 ~]# fdisk -l /dev/sdd
Disk /dev/sdd: 2 GiB, 2147483648 bytes, 4194304 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb1f99266

Device     Boot Start     End Sectors Size Id Type
/dev/sdd1        2048 4194303 4192256   2G 83 Linux
[root@studentvm1 ~]#
```

The fdisk command shows that the partition table has been restored. Now mount the partition and verify that the data we stored there still exists.

```
[root@studentvm1 ~]# mount /dev/sdd1 /mnt ; ll /mnt
total 60
drwx------. 2 root root 16384 Jan 31 08:08 lost+found
-rw-r--r--. 1 root root 44662 Jan 31 08:12 testfile001
[root@studentvm1 ~]#
```

Wow – how cool is that! This series of experiments is designed to illustrate that you can use the fact that all devices can be treated like files and therefore use some very common but powerful CLI tools in some very interesting ways.

It is not necessary to specify the amount of data to be copied with the sb= and count= parameters because the dd command only copies the amount of data available, in this case a single 512-byte sector.

Unmount the /dev/sdd1 device because we are finished with it.

## Implications of everything is a file

The implications of "everything is a file" are far-reaching and much greater than can be listed here. You have already seen some examples in the preceding experiments. But here is a short list that encompasses those and more:

- Clone hard drives.
- Back up partitions.
- Back up the master boot record (MBR).
- Install ISO images onto USB thumb drives.
- Communicate with users on other terminals.
- Print files to a printer.
- Change the contents of certain files in the /proc pseudo-filesystem to modify configuration parameters of the running kernel.
- Overwrite files, partitions, or entire hard drives with random data or zeros.
- Redirect unwanted output from commands to a null device where it disappears forever.
- etc., etc., etc.

There are so many possibilities here that any list can really only scratch the surface. I am sure that you have – or will – figure out many ways to use this tenet of the Linux Philosophy far more creatively than I have discussed here.

## Chapter summary

It is all part of a filesystem. Everything on a Linux computer is accessible as a file in the filesystem space. The whole point of this is to be able to use common tools to operate on different things – common tools such as the standard GNU/Linux utilities and commands that work on files will also work on devices – because, in Linux, they are files.

# Exercises

Perform the following exercises to complete this chapter:

1. Why does even root have restricted access to RAM memory?
2. What is the difference between the device special files /dev/sdc and /dev/sdc1?
3. Can the cat command be used to dump the data from a partition?
4. Can the cat command be used to dump the data from a hard drive?
5. Is it possible to use standard Linux commands to clone complete hard drives in the manner of tools like Ghost, CloneZilla, or Paragon Drive Copy 15 Professional? Don't worry – skip this question if you are not familiar with any of these tools.
6. Create a backup file of the entire partition /dev/sdd1 (or the equivalent device on your VM) and store the data file on /tmp which should have more than enough space to contain this backup file.
7. What would happen if the backup file from #6 were restored to a new partition that was created to be larger than the original partition?

CHAPTER 4

# Managing Processes

## Objectives

In this chapter you will learn

- What a process is
- How processes are represented to the kernel
- Kernel process scheduling
- How processes use system resources
- To use common Linux tools for exploring and managing processes

## Processes

The function of an operating system like Linux is to run programs that perform tasks for the users. Behind the scenes, the operating system runs its own programs that are used to manage the computer hardware, the devices attached to it, and the running programs themselves.

Each program consists of one or more processes. A process is a running program and consumes resources such as memory and CPU time. In this chapter we will look at how the kernel schedules processes to receive CPU time.

## Process scheduling in the kernel

The Linux kernel provides scheduling services that determine which processes get CPU time, how much, and when. Most Linux distributions of the last decade use the Completely Fair Scheduler (CFS) which was introduced into the kernel in October of 2007.[1]

[1]Wikipedia, *Completely Fair Scheduler*, https://en.wikipedia.org/wiki/Completely_Fair_Scheduler




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_4

The overall objective of CPU scheduling in a modern operating system is to ensure that critical processes such as memory allocation or emptying a communications buffer get CPU time immediately when they need it while ensuring that system administration and user-level processes get CPU time and are responsive to the users including the root user. The scheduling of processes for access to CPU time is managed by a complex algorithm that considers many factors.

Each process has its existence in a kernel data structure as an abstraction consisting of data about the process including its process ID (PID) number, memory locations assigned to it, its priority and nice number, how much CPU time it has recently used, how long ago it was actually on CPU, files opened by the process, as well as other necessary data. Like the processes they represent, the kernel data structures that form the abstraction are ephemeral and the RAM memory assigned to them is reassigned to the pool of free memory.

Opensource.com has published an excellent article[2] that provides a good overview of the CFS scheduler and compares it to the older and more simple preemptive scheduler. For a more detailed description of the Linux CFS, Nikita Ishkov, a graduate student at the University of Tampere School of Information Sciences in Finland has written his master's thesis, "A complete guide to Linux process scheduling,"[3] about this.

# Tools

We SysAdmins have access to tools that allow us to view and manage the running processes. For me, these are **`top`**, **`atop`**, **`htop`**, and **`glances`**. All of these tools monitor CPU and memory usage, and most of them list information about running processes at the very least. Some monitor other aspects of a Linux system as well. All provide near real-time views of system activity. Although these tools can generally be run by any non-root user, the root user has more control over all processes, while non-root users can only manage their own processes and have some limitations on that.

[2]Kalin, Marty, *CFS: Completely fair process scheduling in Linux*, `https://opensource.com/article/19/2/fair-scheduling-linux`

[3]Ishkov, Nikita, University of Tampere School of Information Sciences, *A complete guide to Linux process scheduling*, `https://tampub.uta.fi/bitstream/handle/10024/96864/GRADU-1428493916.pdf`, 2015

Processes sometimes misbehave and need to be brought under control. Sometimes we just want to satisfy our curiosity in our efforts to more fully understand the workings of a properly functioning Linux computer so that we will be able to identify when it is malfunctioning. These tools can help us do both.

Let's start by looking at **top**, arguably the oldest of these tools and the one which is most likely to be always available on a modern Linux host. I will use top in the chapter to introduce a number of concepts pertaining to process management and to familiarize you with its use, then move on to other tools that you will find useful in monitoring and managing processes.

## top

One of the first tools I use when performing problem determination is **top**. I like it because it has been around since forever – well, 1984, anyway – and is always available, while the other tools may not be installed. The **top** program is a very powerful utility that provides a great deal of information about your running system. This includes data about memory usage, CPU loads, and a list of running processes including the amount of CPU time and memory being utilized by each process. Top displays system information in near real time, updating (by default) every 3 seconds. Fractional seconds are allowed by **top**, although very small values can place a significant load on the system. It is also interactive and the data columns to be displayed and the sort column can be modified.

The output from **top**, shown in Figure 4-1, is divided into two sections – the "summary" section, which is the upper section of the output, and the "process" section which is the lower portion of the output; I will use this terminology for **top**, **atop**, **htop**, and **glances** in the interest of consistency.

```
top - 21:49:11 up 1 day,  9:55,  6 users,  load average: 0.00, 0.00, 0.00
Tasks: 201 total,   1 running, 200 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.3 hi,  0.7 si,  0.0 st
MiB Mem :   3942.5 total,    234.4 free,    468.9 used,   3239.2 buff/cache
MiB Swap:   4096.0 total,   4094.7 free,      1.3 used.   3230.5 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1738 student   20   0  293564   3156   2708 S   0.3   0.1   2:03.22 VBoxClient
 2845 root      20   0   40128   6512   4236 S   0.3   0.2   0:01.47 sshd
11630 root      20   0       0      0      0 I   0.3   0.0   0:00.17 kworker/0:2-events_powe
11901 root      20   0  228752   4532   3912 R   0.3   0.1   0:00.05 top
    1 root      20   0  171396  14216   9120 S   0.0   0.4   0:13.70 systemd
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.21 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H
    8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 S   0.0   0.0   0:05.03 ksoftirqd/0
   10 root      20   0       0      0      0 I   0.0   0.0   0:04.95 rcu_sched
   11 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_bh
   12 root      rt   0       0      0      0 S   0.0   0.0   0:00.04 migration/0
   14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
   15 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
```

***Figure 4-1.*** *Typical output from the top utility*

## EXPERIMENT 4-1

In order to more fully understand the descriptions of **top** and the other process management tools that will be covered in this chapter, start **top** as root in a terminal session on the desktop.

```
[root@studentvm1 ~]# top
```

Refer to this instance of **top** as we proceed through the following sections that describe its output.

Much of the description to the display generated by **top** is similar to that of the other tools covered in this chapter. We will spend the most time with **top** for this reason and because it is always available while the other tools may need to be installed. We will then cover some of the differences between **top** and the other tools.

## Summary section

The summary section of the output from top is an overview of the system status. The first line shows the system uptime and the 1-, 5-, and 15-minute load averages. We will discuss load averages in more detail later in this chapter.

The second line shows the number of process currently active and the status of each. The lines containing CPU statistics are shown next. There can be a single line which combines the statistics for all CPUs present in the system or, as in the following example, one line for each CPU; in the case of the VM used for the example, this is a single dual-core CPU. Press the **1** key to toggle between the consolidated display of CPU usage and the display of the individual CPUs. The data in these lines is displayed as percentages of the total CPU time available.

The other fields for these CPU data have changed over time and I had a difficult time locating information about the last three as they are relatively new. So here is a description of all of these fields:

- **us: userspace:** Applications and other programs running in user space, that is, not in the kernel.
- **sy: system calls:** Kernel level functions. This does not include CPU time taken by the kernel itself, just the kernel system calls.
- **ni: nice:** Processes that are running at a positive nice level.
- **id: idle:** Idle time, that is, time not used by any running process.
- **wa: wait:** CPU cycles that are spent waiting for I/O to occur. This is wasted CPU time.
- **hi: hardware interrupts:** CPU cycles that are spent dealing with hardware interrupts.
- **si: software interrupts:** CPU cycles spent dealing with software-created interrupts such as system calls.
- **st: steal time:** The percentage of CPU cycles that a virtual CPU waits for a real CPU while the hypervisor is servicing another virtual processor.

The last two lines in the summary section are memory usage. They show the physical memory usage including both RAM and swap space.

## Process section

The process section of the output from top is a listing of the running processes in the system — at least for the number of processes for which there is room on the terminal display. The default columns displayed by top are described in the following. Several other columns are available and each can usually be added with a single keystroke; refer to the top man page for details.

- **PID:** The process ID.
- **USER:** The username of the process owner.
- **PR:** The priority of the process.
- **NI:** The nice number of the process.
- **VIRT:** The total amount of virtual memory allocated to the process.
- **RES:** Resident size (in KB unless otherwise noted) of non-swapped physical memory consumed by a process.
- **SHR:** The amount of shared memory in KB used by the process.
- **S:** The status of the process. This can be R for running, S for sleeping, and Z for zombie. Less frequently seen statuses can be T for traced or stopped and D for uninterruptable sleep.
- **%CPU:** The percentage of CPU cycles or time used by this process during the last measured time period.
- **%MEM:** The percentage of physical system memory used by the process.
- **TIME+:** Total CPU time to 100ths of a second consumed by the process since the process was started.
- **COMMAND:** This is the command that was used to launch the process.

Now that we know a little about the data displayed by top, let's do an experiment to illustrate some of its basic capabilities.

## EXPERIMENT 4-2

The **top** program should already be running in a root terminal session. If not, make it so. Start by observing the summary section.

The top program has a number of useful interactive commands you can use to manage the display of data and to manipulate individual processes. Use the **h** key to view a brief help page for the various interactive commands. Be sure to press **h** twice to see both pages of the help. Use the **q** command to quit **top**.

Use the **1** (one) key to display CPU statistics as a single, global number as shown in Figure 4-1 or by individual CPU as seen in Figure 4-2. The **l** (el) key turns load averages on and off. Use the **t** and **m** keys to rotate the process/CPU and memory lines of the summary section, respectively, through off, text only, and a couple types of bar graph formats.

The **d** or **s** keys are interchangeable and can be used to set the delay interval between updates. The default is 3 seconds, but I prefer a 1-second interval. Interval granularity can be as low as one-tenth (0.1) of a second, but this will consume more of the CPU cycles you are trying to measure. Try setting the delay interval to 5, 1, .5, and other intervals you think might be interesting. When you have finished, set the delay interval to 1 second.

Use the **Page Up** and **Page Down** keys to scroll through the list of running processes.

By default the running processes are sorted by CPU usage. You can use the **<** and **>** keystrokes to sequence the sort column to the left or right. By default there is no highlight or other mark to indicate by which column the results are being sorted. You can add highlighting – press the **x** key which will show the current sort column in bold – the entire column.

```
top - 10:23:10 up 1 day, 22:29,  6 users,  load average: 0.00, 0.00, 0.00
Tasks: 202 total,   1 running, 201 sleeping,   0 stopped,   0 zombie
%Cpu0  :   0.7/2.6    3[||                                                  ]
%Cpu1  :   0.3/1.3    2[|                                                   ]
MiB Mem :   18.4/3942.5[|||||||||||||||                                     ]
MiB Swap:   0.0/4096.0 [                                                    ]

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0  171396  14224   9120 S   0.7   0.4   0:17.96 systemd
  799 root      20   0  231324   2860   2044 S   0.3   0.1   0:09.38 screen
  900 dbus      20   0   42300   6052   4428 S   0.3   0.1   0:21.26 dbus-daemon
  962 root      20   0  588844  14960  13264 S   0.3   0.4   0:00.27 abrt-dump-journ
 1006 root      20   0  273668  35524  34192 S   0.3   0.9   0:11.58 sssd_nss
 1054 root      20   0  547304  19192  16336 S   0.3   0.5   0:10.18 NetworkManager
 2054 root      20   0  253960  11900  10172 S   0.3   0.3   0:01.11 abrt-dbus
11940 root      20   0  228752   4960   4092 R   0.3   0.1   1:30.09 top
27130 root      20   0       0      0      0 I   0.3   0.0   0:00.01 kworker/u4:0-events_unbound
    2 root      20   0       0      0      0 S   0.0   0.0   0:00.31 kthreadd
    3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H
    8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 S   0.0   0.0   0:07.30 ksoftirqd/0
   10 root      20   0       0      0      0 I   0.0   0.0   0:06.84 rcu_sched
   11 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_bh
   12 root      rt   0       0      0      0 S   0.0   0.0   0:00.05 migration/0
   14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
   15 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
   16 root      rt   0       0      0      0 S   0.0   0.0   0:00.22 migration/1
```

***Figure 4-2.*** *Use the t and m keys to change the CPU and memory summaries to bar graphs*

Use the **>** key to move the column all the way to the right and then the **<** key to move it back to the desired column. Move the sort index to various columns. When finished, set the CPU usage as the sort column again.

If you alter the **top** display configuration, you can use the **W** (in uppercase) key to write the changes to the configuration file, ~/.toprc, in your home directory. Leave **top** running.

## More about load averages…

Before we continue, it is important to discuss load averages in more detail. Load averages arean important criteria for measuring CPU usage. But what does this really mean when I say that the 1- (or 5- or 10-) minute load average is 4.04, for example? Load average can be considered a measure of demand for the CPU; it is a number that represents the average number of instructions waiting for CPU time. So this is a true measure of CPU performance, unlike the standard "CPU percentage" which includes I/O wait times during which the CPU is not really working.

For example, a fully utilized single processor system CPU would have a load average of 1. This means that the CPU is keeping up exactly with the demand; in other words, it has perfect utilization. A load average of less than 1 means that the CPU is underutilized and a load average of greater than 1 means that the CPU is overutilized and that there is pent-up, unsatisfied demand. For example, a load average of 1.5 in a single CPU system indicates that one-third of the CPU instructions are forced to wait to be executed until the one preceding it has completed.

This is also true for multiple processors. If a 4-CPU system has a load average of 4, then it has perfect utilization. If it has a load average of 3.24, for example, then three of its processors are fully utilized and one is utilized at about 24%. In the preceding example, a 4-CPU system has a 1-minute load average of 4.04, meaning that there is no remaining capacity among the 4 CPUs and a few instructions are forced to wait. A perfectly utilized 4-CPU system would show a load average of 4.00 so that the system in the example is fully loaded but not overloaded.

The optimum condition for load average is for it to equal the total number of CPUs in a system. That would mean that every CPU is fully utilized and yet no instruction must be forced to wait. But reality is messy and optimum conditions are seldom met. If a host were running at 100% utilization, this would not allow for spikes in CPU load requirements.

The longer-term load averages provide indication of the overall utilization trend.

*Linux Journal* has an excellent article[4] describing load averages, the theory, the math behind them, and how to interpret them in the December 1, 2006, issue.

[4] *Linux Journal, Examining Load Average,* www.linuxjournal.com/article/9001?page=0,0

## …and signals

The **top** utility and all of the other monitors discussed here allow you to send signals[5] to running processes. Each of these signals has a specific function though some of them can be defined by the receiving program using signal handlers.

The separate **kill** command can also be used to send signals to processes outside of the tools like **top**. The **kill -l** can be used to list all possible signals that can be sent. Three of these signals can be used to kill a process:

- **SIGTERM (15)** Signal 15, SIGTERM, is the default signal sent by top and the other monitors when the **k** key is pressed. It may also be the least effective because the program must have a signal handler built into it. The program's signal handler must intercept incoming signals and act accordingly. So for scripts, most of which do not have signal handlers, SIGTERM is ignored. The idea behind SIGTERM is that by simply telling the program that you want it to terminate itself, it will take advantage of that and clean up things like open files and then terminate itself in a controlled and nice manner.
- **SIGKILL (9)** Signal 9, SIGKILL, provides a means of killing even the most recalcitrant programs, including scripts and other programs that have no signal handlers. For scripts and other programs with no signal handler, however, it not only kills the running script but it also kills the shell session in which the script is running; this may not be the behavior that you want. If you want to kill a process and you don't care about being nice, this is the signal you want. This signal cannot be intercepted by a signal handler in the program code.
- **SIGINT (2)** Signal 2, SIGINT, can be used when SIGTERM does not work and you want the program to die a little more nicely, for example, without killing the shell session in which it is running. SIGINT sends an interrupt to the session in which the program is running. This is equivalent to terminating a running program, particularly a script, with the **Ctrl-C** key combination.

[5]Wikipedia, *Unix Signals*, https://en.wikipedia.org/wiki/Unix_signal

# CPU hogs

Now that we know a bit more, let's experiment. We will create a program that hogs CPU cycles and then run multiple instances of it so that we can use our tools on it, starting with **top**.

**EXPERIMENT 4-3**

Start two terminal sessions on the desktop as the student user. In one terminal session, run top and position this window so you can see it as you perform the following tasks in the second terminal session. Observe the load averages displayed in top as you progress through this experiment.

Create a Bash shell program file in your home directory named cpuHog and make it executable with the permissions `rwxr_xr_x`.

```
[student@studentvm1 ~]$ touch ./cpuHog
[student@studentvm1 ~]$ chmod 755 cpuHog
```

Edit the file with Vim and add the content shown in Figure 4-3 to it. Using **`while [ 1 ]`** forces this program to loop forever. Also the Bash syntax is very picky; be sure to leave spaces around the "1" in this expression; [ 1 ] will work but [1] will not work. Save the file and exit from Vim.

```
#!/bin/bash
# This little program is a cpu hog
X=0;while [ 1 ];do echo $X;X=$((X+1));done
```

***Figure 4-3.*** *The cpuHog program will enable us to explore the tools used to manage processes*

This program simply counts up by one and prints the current value of X to STDOUT. And it sucks up CPU cycles. The terminal session in which cpuHog is running should show a very high CPU usage in top. Observe the effect this has on system performance in top. CPU usage should immediately go way up and the load averages should also start to increase over time. If you want, you can open additional terminal sessions and start the cpuHog program in them so that you have multiple instances running.

Now start the cpuHog program.

```
[student@studentvm1 ~]$ ./cpuHog
```

This program will run until we stop it. Use the top program to show CPU usage. As you can see in Figure 4-4, this should show that one cpuHog instance is taking up a very large amount of CPU time. Record the PID – 5036 in Figure 4-4, but it will be different on your VM – for this instance of the cpuHog. You should also notice that the nice number of this instance is 0.

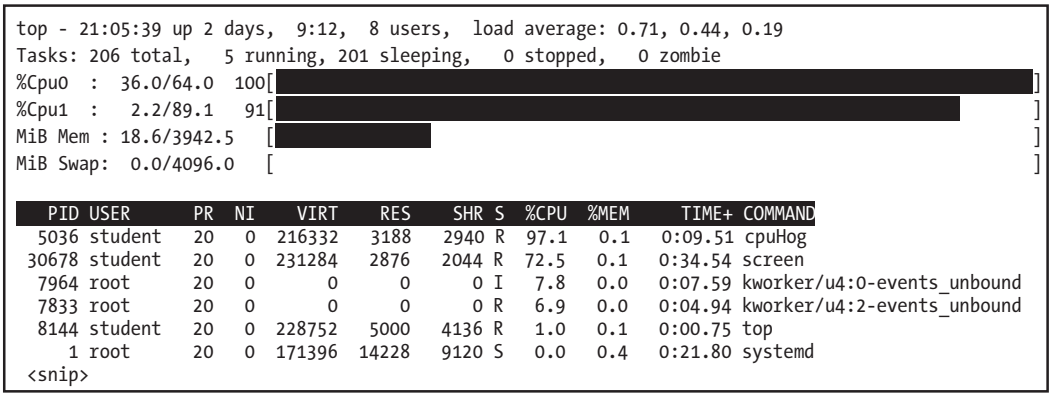


***Figure 4-4.*** *One instance of cpuHog is running and takes over 97% of CPU cycles on one CPU*

Open four new screens or terminal sessions and start an instance of cpuHog in each. Both CPUs should be running at 100%. You will also notice that other programs such as screen, if you are using it, also take CPU time.

Watch the load averages as they climb over time.

---

**Note** If you need to go do other things while working in this chapter, it would be fine - but not necessary - to terminate any running cpuHog instances and restart them later when you come back.

---

# Process scheduling

Linux schedules each task for time on the CPU using an algorithm that considers some basic factors, including its nice number. These factors are combined into a priority by the algorithm. The factors considered by the Linux kernel scheduler include the following for each process:

- Length of time waiting for CPU time
- Amount of CPU time recently consumed
- Nice number
- The priority of the process in question
- The priorities of the other processes waiting for CPU time

The algorithm, which is a part of the kernel scheduler, determines the priority of each process running in the system. Programs or processes with higher priorities are more likely to be allocated CPU time. Priorities are very dynamic and can change rapidly based on the factors listed previously.

Linux process priorities run from 0 through 39 with 39 being the lowest priority and 0 the highest. This seems to be reversed from common logic, but you should consider that higher numbers mean a "nicer" priority.

There is also an RT, or RealTime, priority which is used by some processes that need to get CPU time immediately when some event occurs. This might be a process that handles hardware interrupts for the kernel. In order to ensure that data is not lost as it arrives from a disk drive or network interface, for example, a high priority process is used to empty the data buffer when it becomes full and store the data in some specific memory location where it can be accessed as needed. Meanwhile, the empty input buffer can be used to store more incoming data from the device.

# Nice numbers

Nice numbers are the mechanism used by administrators to affect the priority of a process. It is not possible to change the priority of a process directly, but changing the nice number can modify the results of the kernel scheduler's priority setting algorithm. Nice numbers run from -20 to +19 where higher numbers are nicer.

The default nice number is 0 and the default priority is 20. Setting the nice number higher than zero increases the priority number somewhat, thus making the process nicer and therefore less greedy of CPU cycles. Setting the nice number to a more negative number results in a lower priority number making the process less nice. Nice numbers can be changed using the **renice** command or from within **top**, **atop**, and **htop**.

Now that we know a bit more about priorities and nice numbers, we can explore them in this next experiment.

## EXPERIMENT 4-4

Start this experiment in the terminal session that is already running `top` as the student user. Be sure to use the PID numbers that pertain to your VM rather than the ones I used on my VM.

Set the sort column to TIME+ so that you can more easily observe the steadily increasing total amount of CPU time accumulated by the cpuHogs.

Renice the oldest process, in my case the one with PID 5036, from within `top`. Simply type **r** and `top` asks you which process to renice. Enter the PID of the process and hit the **Enter** key. In my case the PID is 5036; the PID will definitely be different for you. Then it asks "Renice PID 5036 to value:". Now type the number **10** and hit the **Enter** key.

Verify that the nice number is now 10 and look at the priority number. On my VM the priority is now 30 which is lower than the default priority of 20. Switch to a different terminal session, one that is not running `top` or the cpuHog, and change the nice number from the command line.

```
[student@studentvm1 ~]$ renice 15 5036
5036 (process ID) old priority 10, new priority 15
[student@studentvm1 ~]$
```

Verify that the new nice number is 15. What is the priority number for this process now? On my VM the priority is now 35.

Now use the `renice` command and set the nice number of PID 5036 to -20.

```
[student@studentvm1 ~]$ renice -20 5036
renice: failed to set priority for 5036 (process ID): Permission denied
[student@studentvm1 ~]$
```

You will receive an error indicating that you don't have permission to do this. A non-root user cannot renice their **own** processes to a lower (less nice) number. Why do you think that this might be the case?

Start top in a root terminal session. Now, as root, reset the nice number of the cpuHog on your VM to -20. This will work this time because root can do anything. Observe the nice number of this process. Again the system is no more or less responsive, but in a real environment, a program that has a -20 nice number might cause the rest of the system to become sluggish.

Open another terminal session – as a new tab, a new window, or in a screen session – it does not matter which. Start another instance of the cpuHog program, but let's do it a bit differently.

```
[student@studentvm1 ~]$ nice -n +20 ./cpuHog
```

When we know we want to start a program with a different nice number than the default of 0 (zero), we can use the **nice** command to start it and specify the nice number. Verify the nice number and the priority number in the **top** display.

Note that any nice number higher than 19 is interpreted to be 19 regardless of which tool is used to set it. Although the system is no more or less responsive because this is a shell script and there is plenty of CPU available, this is one way to make a process behave more nicely.

Open more terminal sessions as the student user and start at least five more instances of the cpuHog without changing the nice number. Allow all of these processes to run. As you watch these cpuHog processes you should observe that our artificial environment, all of the cpuHog processes receive from about 15% to 24% of the overall CPU time.

---

I found in my own experiments that little changed in terms of the amount of CPU time the accrued to the cpuHog with the highest nice number vs. the cpuHog with the lowest. This experiment does show how to set nice numbers and the resultant changes in the priority of the processes, but there are other factors that the kernel uses in its allocation of CPU resources. Those factors make it impossible to state with certainty that changing the nice number of a particular process will have a specific effect. This is especially true in an artificially created situation such as an experiment like this one.

With that said, however, in a production environment, I have found that increasing the nice number of a particularly greedy hog process can improve the responsiveness of other processes.

# Killing processes

Sometimes a process must be killed because it cannot otherwise be controlled. There are a number of ways to do this.

**EXPERIMENT 4-5**

Be sure that you can see the top utility running in a separate terminal session. Switch to one terminal session that is running an instance of cpuHog as the student user.

As the student user, choose one current instance of the cpuHog and determine its PID. Now let's kill this newest cpuHog process from within top.

In the screen session in which **top** is running as the student user, type **k**. Now **top** asks "PID to kill:". Type in the PID of the process and press the **Enter** key. On my VM I chose a cpuHog with PID 5257, but you must use the PID of one of your cpuHog instances. The **top** program now displays "Kill PID 5257 with signal [15]:". At this point you could choose another signal or just press Enter. For now, just press **Enter** and the program disappears from the **top** process list.

Signal 15 is used to terminate a program nicely and give it a chance to clean up after itself and close open files if there are any. This is the nicest way of killing a process if it has no option to terminate itself within its own interface.

For processes that are a bit more recalcitrant and that don't respond to Signal 15, we can use Signal 9 which tells the kernel to just kill **the** program without being nice. Go to an unused screen session and enter the command and locate the PID of one of the cpuHogs. On my VM this was 7533, but be sure to use the correct PID for one of the cpuHogs on your system. This time we will use the **kill** command to send Signal 9.

```
[student@testvm1 ~]$ kill -9 7533
```

Choose another running cpuHog instance and let's use Signal 2 with the **kill** command.

```
[student@testvm1 ~]$ kill -2 12503
```

This sends the SIGINT (2) signal to process 12503. Switch to the screen session in which the program was running and verify that it has been killed. Note the message that says "Killed."

While still in that same terminal session, restart the program. After it has run for a couple seconds, press **Ctrl-c**. Note that the running program is killed.

```
6602
6603
6604
6605
6606
6607
6608
6609
6610
6611
^C
[student@studentvm1 ~]$
```

Using kill -2 is the same as pressing Ctrl-c from within the session running the program.

You should still have some cpuHogs running on your VM. Let them run for now as we can use them while exploring other tools for process management.

# Other interactive tools

As we have seen with other tools, Linux provides plenty of options for managing processes. The following tools are also interactive tools that provide different approaches and data displays. All are good and these only represent some of the many tools available.

## atop

The **atop** tool, shown in Figure 4-5, is an excellent monitor to use when you need more details about I/O activity. The default refresh interval is 10 seconds, but this can be changed using the interval (**i**) command to whatever is appropriate for what you are trying to do. **atop** cannot refresh at sub-second intervals like top can.

Use the **h** key to display help. Be sure to notice that there are multiple pages of help and you can use the space bar to scroll down to see the rest.

One nice feature of **atop** is that it can save raw performance data to a file and then play it back later for close inspection. This is handy for tracking down intermittent problems, especially ones that occur during times when you cannot directly monitor the system. The **atopsar** program is used to play back the data in the saved file.

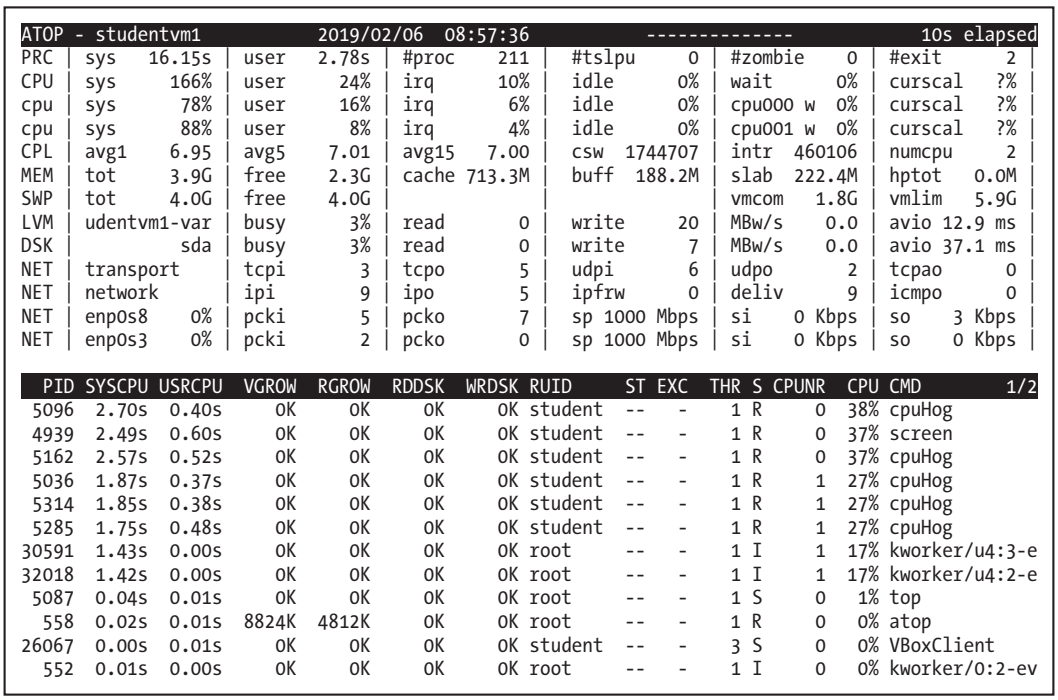


***Figure 4-5.*** *The atop system monitor provides information about disk and network activity in addition to CPU and process data. Click the image for a full size version*

## Summary section

**atop** contains much of the same information as top but also displays information about network, raw disk, and logical volume activity. Figure 4-5 shows these additional data in the columns at the top of the display. Note that if you have the horizontal screen real

estate to support a wider display, additional columns will be displayed. Conversely, if you have less horizontal width, fewer columns are displayed. I also like that atop displays the current CPU frequency and scaling factor – something I have not seen on any other of these monitors – on the second line in the rightmost two columns in Figure 4-5.

## Process section

The **atop** process display includes some of the same columns as that for top, but it also includes disk I/O information and thread count for each process as well as virtual and real memory growth statistics for each process. As with the summary section, additional columns will display if there is sufficient horizontal screen real estate. For example, in Figure 4-5, the RUID (Real User ID) of the process owner is displayed. Expanding the display will also show the EUID (Effective User ID) which might be important when programs run SUID (Set User ID).

**atop** can also provide detailed information about disk, memory, network, and scheduling information for each process. Just press the **d**, **m**, **n**, or **s** keys respectively to view that data. The **g** key returns the display to the generic process display.

Sorting can be accomplished easily by using **C** to sort by CPU usage, **M** for memory usage, **D** for disk usage, **N** for network usage, and **A** for automatic sorting. Automatic sorting usually sorts processes by the most busy resource. The network usage can only be sorted if the netatop kernel module is installed and loaded.

You can use the **k** key to kill a process, but there is no option to renice a process.

By default, network and disk devices for which no activity occurs during a given time interval are not displayed. This can lead to erroneous assumptions about the hardware configuration of the host. The **f** command can be used to force atop to display the idle resources.

**EXPERIMENT 4-6**

Install **atop** and then start it.

```
[root@studentvm1 ~]# dnf -y install atop ; atop
```

Observe the **atop** display for a few minutes. Start a couple cpuHogs and kill them after observing their effects on the system. Do not delete the cpuHogs that have been running for a long time.

Be sure to look at the csw (Context Switches) and intr (Interrupts) data. Context switches are the number of times per interval that the CPU switches from one program to another. Interrupts displays the number of times per interval that software or hardware interrupts occur and cause the CPU to handle those interrupts.

Read the man page for **atop** to learn more about any data items and interactive commands that you find interesting.

Do not kill the remaining cpuHog instances. Exit from **atop**.

## Configuration

The atop man page refers to global- and user-level configuration files, but none can be found in my own Fedora or CentOS installations. There is also no command to save a modified configuration and a save does not take place automatically when the program is terminated. So there appears to be no way to make configuration changes permanent.

# htop

The htop program is much like top on steroids, as you can see in Figure 4-6. It does look a lot like top, but it also provides some capabilities that top does not. Unlike atop, however, it does not provide any disk, network, or I/O information of any type.

```
1  [||||||||||||||||||||||||||||||100.0%]   Tasks: 122, 126 thr; 2 running
2  [||||||||||||||||||||||||||||||100.0%]   Load average: 6.93 6.97 6.99
Mem[|||||||||||||||||              523M/3.85G]   Uptime: 3 days, 18:18:35
Swp[                                 0K/4.00G]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 5036 student    0 -20  211M  1132   988 R 35.9  0.0 19h37:26 /bin/bash ./cpuHog
 4939 student   20   0  227M  4236  2048 S 35.2  0.1 21h36:44 SCREEN
 5285 student   20   0  211M  1072   924 R 32.8  0.0 19h35:32 /bin/bash ./cpuHog
 5314 student   20   0  211M  1136   988 R 31.2  0.0 19h29:25 /bin/bash ./cpuHog
 5162 student   20   0  211M  1132   988 R 27.3  0.0 19h33:59 /bin/bash ./cpuHog
 5096 student   32  12  211M  3152  2912 R 27.3  0.1 19h41:37 /bin/bash ./cpuHog
  866 root       0 -20 17044 16984 12580 S  2.3  0.4  3:02.08 atop
 4962 root      20   0  220M  4392  3712 R  1.6  0.1  0:00.07 htop
    1 root      20   0  103M 14660  9196 S  0.8  0.4  0:32.39 /usr/lib/systemd/systemd --switch
  653 root      20   0  197M  125M  112M S  0.8  3.2  0:20.94 /usr/lib/systemd/systemd-journald
  901 root      20   0  665M 78208 76220 S  0.8  1.9  0:20.35 /usr/sbin/rsyslogd -n
  930 dbus      20   0 42272  6632  4856 S  0.8  0.2  0:40.63 /usr/bin/dbus-daemon --system --a
 4967 root      20   0  159M  8656  7692 S  0.8  0.2  0:00.01 /usr/libexec/nm-dispatcher
 4797 root      20   0 39180  5040  3804 S  0.0  0.1  2:47.35 sshd: root@pts/0
 5068 root      20   0  225M  2908  1900 S  0.0  0.1  1:52.78 SCREEN
  678 root      20   0 33300 12232  8008 S  0.0  0.3  0:02.01 /usr/lib/systemd/systemd-udevd
  879 root      16  -4  104M  1824  1220 S  0.0  0.0  0:00.01 /sbin/auditd
  881 root      16  -4  104M  1824  1220 S  0.0  0.0  0:00.21 /sbin/auditd
  878 root      16  -4  104M  1824  1220 S  0.0  0.0  0:01.22 /sbin/auditd
  880 root      16  -4  7068  3412  3016 S  0.0  0.1  0:00.47 /usr/sbin/sedispatch
  926 root      20   0  308M  9796  8368 S  0.0  0.2  0:00.00 /usr/sbin/ModemManager
```

***Figure 4-6.*** *htop has nice bar charts to indicate resource usage and it can show the process tree*

If you get into a menu or dialog that you want to get out of without making any changes, you can always press the Escape (**Esc**) key once to back up.

## Summary section

The summary section of **htop** is displayed in two columns. It is very flexible and can be configured with several different types of information in pretty much any order you like. **htop** has a number of different options for the CPU display, including a single combined bar, a bar for each CPU, and various combinations in which specific CPUs can be grouped together into a single bar.

I think this is a cleaner summary display than some of the other system monitors and it is easier to read. The drawback to this summary section is that some information is not available in **htop** that is available in the other monitors, such as CPU percentages by user, idle, and system time.

The **F2** (Setup) key is used to configure the summary section of **htop**. A list of available data displays is shown, and you can use various keys to add them to the left or right column and to move them up and down within the selected column.

Be sure to look at the csw (Context Switches) and intr (Interrupts) data. Context switches are the number of times per interval that the CPU switches from one program to another. Interrupts displays the number of times per interval that software or hardware interrupts occur and cause the CPU to handle those interrupts.

## Process section

The process section of **htop** is very similar to that of top. As with the other monitors, processes can be sorted any of several factors, including CPU or memory usage, user, or PID. Note that sorting is not possible when the tree view is selected.

The **F6** key allows you to select the sort column; it displays a list of the columns available for sorting, and you select the column you want and press the **Enter** key.

You can use the **Up** and **Down** arrow keys to select a process. The cursor bar moves up and down to highlight individual processes. The **Space bar** can be used to tag multiple processes on which commands may be performed.

To kill a process, use the arrow keys to select the target process and press the **F9** or the **k** key. A list of signals to send the process is displayed down the left side of the terminal session with 15, SIGTERM, selected. You can specify the signal to use, if different from SIGTERM using the **Up** and **Down** arrow keys. You could also use the **F7** and **F8** keys to renice the selected process.

One option I especially like is **F5** which displays the list of running processes in a tree format making it easy to determine the parent/child relationships of running processes.

### EXPERIMENT 4-7

Perform this experiment as root. Install **htop** and then start it.

```
[root@studentvm1 ~]# dnf -y install htop ; htop
```

Observe the **htop** display for a few minutes. Press the **T** (uppercase) key to sort the processes on their total accumulated CPU time. Start at least four new cpuHogs. If the new cpuHog instances are not immediately visible, it may take a few minutes but the new cpuHogs will climb up the list of running processes so that they are. Their positions in the list will be more stable this way and make it easier to select one or more of them.

But we do not need to wait in order to locate these cpuHog instances. We could just use the **Page Down** and **Page Up** keys to do an eyeball search, but there is a better way.

Press the **F3** key and type in the search term "cpuH" (without the quotes) and the cursor will move to the first hpuHog instance and will be highlighted. Press **F3** to search through the list of running processes to locate the other cpuHog instances. Pressing any other key will exit from search mode and leave the cursor on the selected process.

Now use the **Up** and **Down** arrow keys to move the cursor to another of the new cpuHog instances. On my VM this is the cpuHog with a PID of 2325. Notice that the cursor remains on this process even if it changes position in the display. The PID of your cpuHog instance will be different.

We are going to kill this process so press **F9** to list the signals that can be sent to the highlighted cpuHog process. Move the cursor up to **SIGKILL** (Signal 9) as shown in Figure 4-7, and press **Enter** to kill this process.

Move the cursor to one of the remaining new cpuHogs. Press the **Space bar** to tag this process. Tag two more of the new cpuHogs. Do not tag any of the cpuHog instances that have been running for a longer time.

Kill the tagged cpuHogs. Press the **k** key. This time leave the cursor on SIGTERM (Signal 15) and press **Enter** to kill all of the tagged processes. Look through the terminal sessions in which you started these cpuHogs to verify that they have been killed.

It is possible to set the delay interval but only from the command line. There is not interactive command that provides this capability. The -d option sets the delay time in 10ths of a second – that is 10ths, not full seconds.

Press **q** to quit from **htop**. Then enter the following command to start htop with a 1-second interval.

```
[root@studentvm1 ~]# htop -d 10
```

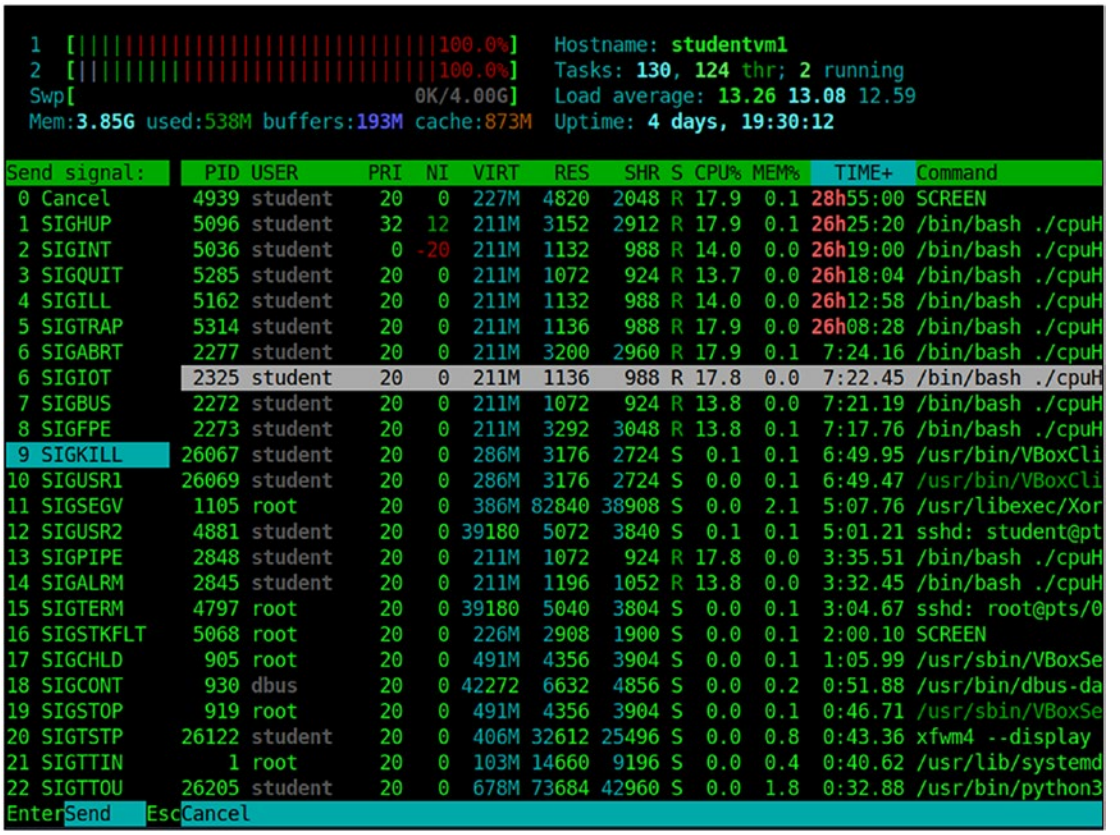


***Figure 4-7.*** *Select a cpuHog instance and press F9 to list the signals that can be sent*

Read the man page for **htop** to learn more about any data items and interactive commands that you find interesting.

Do not kill the remaining cpuHog instances and leave **htop** running.

## Configuration

Each user has their own configuration file, ~/.config/htop/htoprc, and changes to the htop configuration are stored there automatically. There is no global configuration file for htop.

The **htop** utility has a very flexible configuration. The CPU and memory meters in the header section can be configured in the **F2** Setup dialog page as seen in Figure 4-8.

Let's take some time to explore this setup capability in Experiment 4-8.

**EXPERIMENT 4-8**

Perform this experiment as the root user, but everything will work the same way with a non-root user except that non-root users will be unable to change nice numbers in a negative direction.

Press the **F2** key to open the setup dialog which is shown in Figure 4-8. The Meters options in the Setup column allows us to add, delete, and configure the various available meters.

Use the up and down arrow keys to select one of the four setup dialogs. After you check that out, return the cursor to the Meters dialog and look at the meters in each of the left and right columns.

On your display the CPU's meter will be at the top. Ensure that it is highlighted and press **Enter**. The CPU's meter is highlighted with a different color and a double Up/Down arrow symbol (↕). First, press the **Space bar** to rotate through the available meter types. Choose one you like. Use the **Up** and **Down** keys on the keyboard to move this meter to a location you prefer in the current column. Use the **Right arrow** and **Left Arrow keys** to change the column in which the meter is displayed. Press the **Enter** again to lock the meter into position.

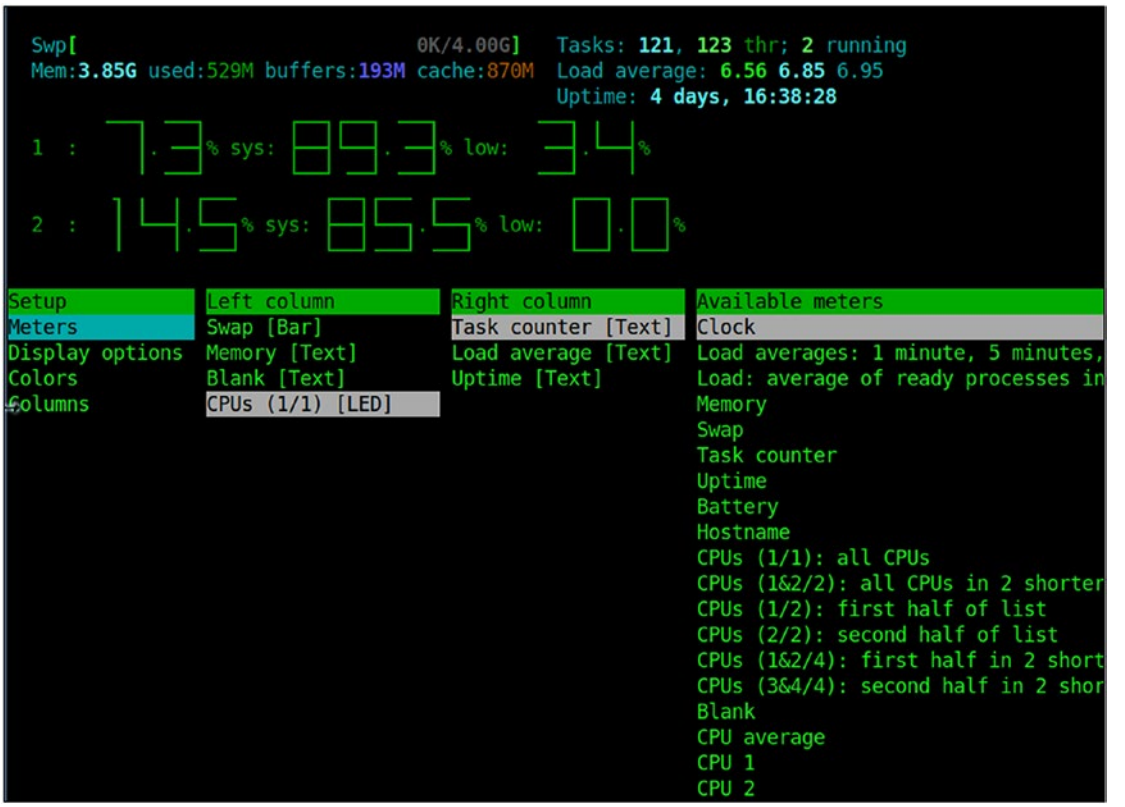


***Figure 4-8.*** *The htop setup dialog showing some different meter styles in the summary section*

The rightmost column in the Meters dialog displays all of the available meters. Move the highlight bar to the Hostname meter and press **Enter**. Use the arrow keys to move the Hostname meter to the top of the Right column and press Enter to lock it into position.

Move the cursor back to the Setup column and place it on **Display options**. Move the cursor to the Display options column and remove the x from **Hide kernel threads**. This enables **htop** to display kernel-related process threads. You could also press **K** (uppercase) when in the main **htop** screen to toggle the showing of kernel threads. The kthr item now appears in the Tasks line of the summary section.

I also like to enable **Leave a margin around header**, **Detailed CPU time**, and **Count CPUs from 0 instead of 1**. Try these and some of the other options in this dialog.

Try out some of the different color schemes in the **Colors** dialog. The Columns dialog allows you to configure the specific columns to be displayed in the process list and in what sequence they appear.

Be sure to spend plenty of time experimenting with htop because it is very useful. Exit from htop when you are finished.

# Glances

The Glances utility can display more information about your computer than any of the other text-mode monitors I am currently familiar with. This includes filesystem I/O, network I/O, and sensor readouts that can display CPU and other hardware temperatures as well as fan speeds and disk usage by hardware device and logical volume.

The drawback to having all of this information is that Glances uses a significant amount of CPU resources itself. On my systems I find that it can use from about 10% to 18% of CPU cycles. That is a lot so you should consider that impact when you choose your monitor. Glances can also explore your network using snmp protocols to discover and query other network nodes.

Glances is cross-platform because it is written in Python. It can be installed on Windows and other hosts that have current versions of Python installed.

## Summary section

The summary section of Glances contains most of the same information as the summary sections of the other monitors. If you have enough horizontal screen real estate, it can show CPU usage with both a bar graph and a numeric indicator; otherwise, it will show only the number.

I like this summary section better than those of the other monitors; I think it provides the right information in an easily understandable format. As with atop and htop, you can press the 1 key to toggle between a display of the individual CPU cores and a global one with all of the CPU cores as a single average as shown in Figure 4-9.

## Process section

The process section displays the standard information about each of the running processes. Processes can be sorted automatically (a) or by CPU (c), memory (m), command name (p), user (u), I/O rate (i), or time (t). When sorted automatically processes are first sorted by the most used resource. In Figure 4-9 the default sort column, TIME+, is highlighted. As with htop TIME+ is the accumulated CPU time for the process.

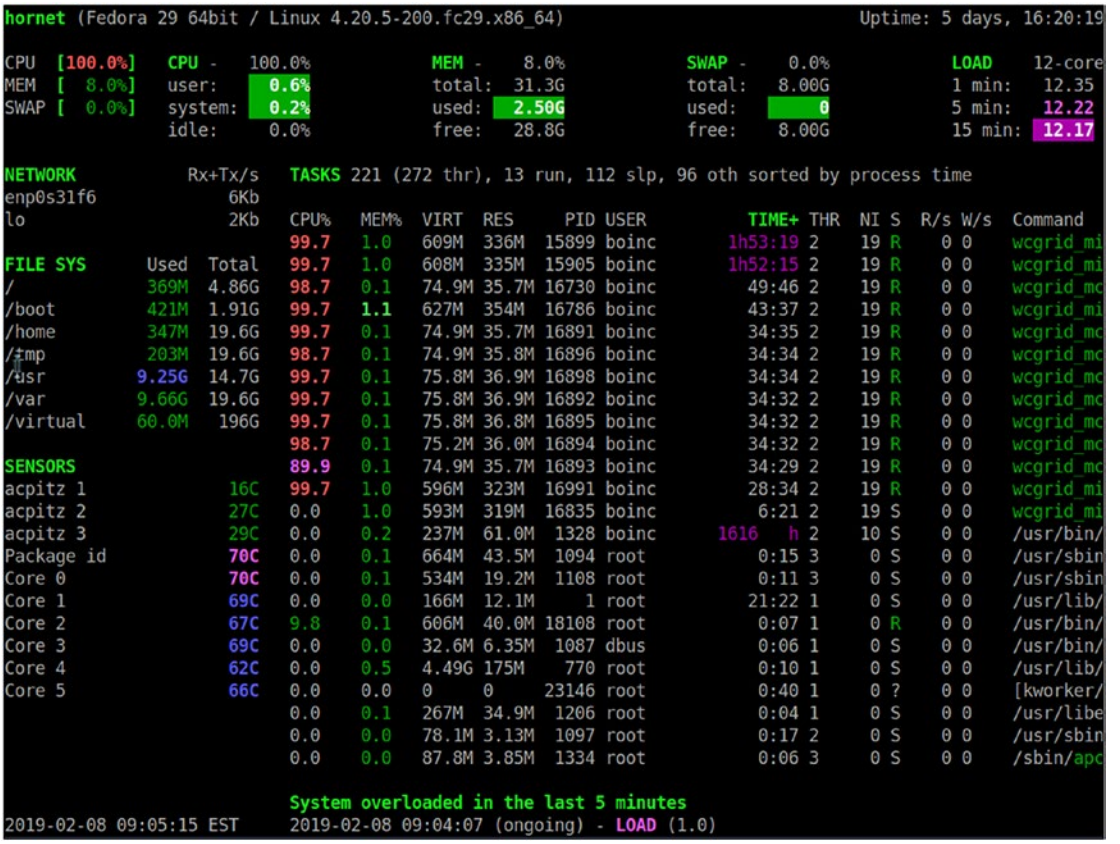


***Figure 4-9.*** *The Glances interface with network, filesystem, and sensor information*

Glances also shows warnings and critical alerts at the very bottom of the screen, including the time and duration of the event. This can be helpful when attempting to diagnose problems when you cannot stare at the screen for hours at a time. These alert logs can be toggled on or off with the **l** (El) key, warnings can be cleared with the **w** key, while alerts and warnings can all be cleared with **x**.

It is interesting that Glances is the only one of these monitors that cannot be used to either kill or renice a process. It is intended strictly as a monitor. You can use the external **kill** and **renice** commands to manage processes.

## Sidebar

Glances has a very nice sidebar that displays information that is not available in top or htop. atop does display some of this data, but Glances is the only monitor that displays the sensor data. Sometimes it is nice to see the temperatures inside your computer.

The individual modules, disk, filesystem, network, and sensors can be toggled on and off using the **d**, **f**, **n**, and **s** commands, respectively. The entire sidebar can be toggled using **2**. Docker stats can be displayed in the sidebar with **D**.

Note that the hardware sensors are not displayed when Glances is running on a virtual machine. For this reason, I have used a screenshot of Glances on one of my physical hosts.

## Configuration

Glances does not require a configuration file to work properly. If you choose to have one, the system-wide instance of the configuration file would be located in /etc/glances/glances.conf. Individual users can have a local instance at ~/.config/glances/glances.conf which will override the global configuration. The primary purpose of these configuration files is to set thresholds for warnings and critical alerts.

There is a document, /usr/share/doc/Glances/Glances-doc.html, that provides a great deal of information about using Glances, and it explicitly states that you can use the configuration file to configure which modules are displayed.

**EXPERIMENT 4-9**

Perform this experiment as the root user. To begin, install Glances and then start it.

```
[root@studentvm1 ~]# dnf -y install glances ; glances
```

Note that, due to the CPU load overhead incurred by **glances**, there may be some delay before the Glances display appears. View the Glances output for a few minutes and locate the various types of data that you have already seen in the previous monitors. Press the **h** key to display a Help menu. Most of the options here simply show/hide various data displays or select a sort column. Press **h** again to exit from the Help menu.

Press the **f** key to hide the filesystem usage statistics. Press **f** again to display them again. Note that the disk I/O statistics are not shown – I am not sure why this is the case. Because this is a VM, the sensor data will not be displayed because there is no physical hardware.

Take some time to experiment with Glances until you feel comfortable with it.

Press **q** or Esc to **exit** from Glances.

Despite its lack of interactive capabilities such as the ability to renice or kill processes and its own high CPU load, I find Glances to be a very useful tool. The complete Glances documentation[6] is available on the Internet, and the Glances man page has startup options and interactive command information.

## Other tools

Sometimes the static tools like the **ps** (process list) tend to get overlooked in our efforts to observe system performance in as near to real time as we can get. The **ps** command can produce a static list of processes. It can list all processes or it can list only the running processes for the user that issues the command. The kill command is used to kill running processes, and it can also send other signals that enable the user or SysAdmin to interact with them.

[6]Glances, *Glances*, https://Glances.readthedocs.io/en/latest/index.html

## EXPERIMENT 4-10

Perform this experiment as the student user. If there are not already some cpuHogs running, start four or five for use in this experiment.

The following command lists the currently running processes that belong to the user. This provides an easy way to find the PIDs of running processes that we might be interested in when troubleshooting performance problems.

```
[student@studentvm1 ~]$ ps -r
  PID TTY      STAT   TIME COMMAND
 5036 pts/6    RN+  193:24 /bin/bash ./cpuHog
 8531 pts/7    R<+  192:47 /bin/bash ./cpuHog
 8650 pts/8    R+   187:52 /bin/bash ./cpuHog
 8712 pts/9    R+   189:08 /bin/bash ./cpuHog
 8736 pts/10   R+   189:18 /bin/bash ./cpuHog
23463 pts/12   R+     0:00 ps -r
[student@studentvm1 ~]$
```

I like the next command because it lists all processes whether running or not.

```
[student@studentvm1 ~]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Feb02 ?        00:00:55 /usr/lib/systemd/systemd
--switched-root --system --deserialize 33
root         2     0  0 Feb02 ?        00:00:00 [kthreadd]
root         3     2  0 Feb02 ?        00:00:00 [rcu_gp]
root         4     2  0 Feb02 ?        00:00:00 [rcu_par_gp]
root         6     2  0 Feb02 ?        00:00:00 [kworker/0:0H-kblockd]
root         8     2  0 Feb02 ?        00:00:00 [mm_percpu_wq]
root         9     2  0 Feb02 ?        00:02:03 [ksoftirqd/0]
root        10     2  0 Feb02 ?        00:03:08 [rcu_sched]
root        11     2  0 Feb02 ?        00:00:00 [rcu_bh]
root        12     2  0 Feb02 ?        00:00:00 [migration/0]
root        14     2  0 Feb02 ?        00:00:00 [cpuhp/0]
root        15     2  0 Feb02 ?        00:00:00 [cpuhp/1]
root        16     2  0 Feb02 ?        00:00:00 [migration/1]
root        17     2  0 Feb02 ?        00:01:33 [ksoftirqd/1]
<snip>
```

```
student  25882  1408  0 Feb03 ?       00:00:00 /bin/sh /etc/xdg/xfce4/
                                      xinitrc -- vt
student  25966  4873  0 Feb03 ?        00:00:00 /usr/libexec/imsettings-daemon
student  25969  4873  0 Feb03 ?        00:00:00 /usr/libexec/gvfsd
student  25976  4873  0 Feb03 ?        00:00:00 /usr/lib64/xfce4/xfconf/xfconfd
student  26042     1  0 Feb03 ?        00:00:00 /usr/bin/VBoxClient --clipboard
student  26043 26042  0 Feb03 ?        00:00:00 /usr/bin/VBoxClient --clipboard
student  26053     1  0 Feb03 ?        00:00:00 /usr/bin/VBoxClient --display
student  26054 26053  0 Feb03 ?        00:00:00 /usr/bin/VBoxClient --display
student  26059     1  0 Feb03 ?        00:00:00 /usr/bin/VBoxClient --seamless
student  26073 25882  0 Feb03 ?        00:00:01 /usr/bin/ssh-agent /bin/sh -c
                                      exec -l /bin/bash -c "startxfce4"
student  26103 25882  0 Feb03 ?        00:00:01 xfce4-session
student  26104  4873  0 Feb03 ?        00:00:00 /usr/libexec/at-spi-bus-launcher
student  26110 26104  0 Feb03 ?        00:00:00 /usr/bin/dbus-daemon
<snip>
root     26363     1  0 Feb03 ?        00:00:03 /usr/sbin/abrt-dbus -t133
student  26415  4873  0 Feb03 ?        00:00:00 /usr/libexec/bluetooth/obexd
student  26483 26141  0 Feb03 ?        00:00:04 orage
```

I can use grep and other commands to locate specific processes using appropriate filters.

```
[root@studentvm1 ~]# ps -ef | grep xfce
student   1311  1283  0 Jul17 ?       00:00:00 /bin/sh /etc/xdg/xfce4/
                                      xinitrc -- vt
student   1399  1290  0 Jul17 ?       00:00:00 /usr/lib64/xfce4/xfconf/xfconfd
student   1501  1311  0 Jul17 ?       00:00:00 /usr/bin/ssh-agent /bin/sh -c
                                      exec -l /bin/bash -c "startxfce4"
student   1531  1311  0 Jul17 ?       00:00:00 xfce4-session
student   1554     1  0 Jul17 ?       00:00:05 xfce4-panel
student   1584  1531  0 Jul17 ?       00:00:00 /usr/libexec/xfce-polkit
student   1595     1  0 Jul17 ?       00:00:00 xfce4-power-manager
student   1723  1290  0 Jul17 ?       00:00:00 /usr/lib64/xfce4/notifyd/
                                      xfce4-notifyd
student   1944  1554  0 Jul17 ?       00:00:01 /usr/lib64/xfce4/panel/
                                      wrapper-2.0 /usr/lib64/xfce4/panel/
                                      plugins/libsystray.so 6 23068680
                                      systray Notification Area Area where
                                      notification icons appear
```

```
student   1950  1554  0 Jul17 ?        00:00:00 /usr/lib64/xfce4/panel/
                                       wrapper-2.0 /usr/lib64/xfce4/panel/
                                       plugins/libactions.so 2 23068681
                                       actions Action Buttons Log out, lock
                                       or other system actions
student   1951  1554  0 Jul17 ?        00:00:00 /usr/lib64/xfce4/panel/
                                       wrapper-2.0 /usr/lib64/xfce4/panel/
                                       plugins/libnotification-plugin.
                                       so 18 23068682 notification-plugin
                                       Notification Plugin Notification
                                       plugin for the Xfce panel
student   2019     1  0 Jul17 ?        00:00:05 /usr/bin/xfce4-terminal
root      5051 22865  0 12:19 pts/3    00:00:00 grep --color=auto xfce
```

The next command displays all processes running that belong to the user student. Note that this is *not* all of the processes that belong to the student user, just the ones that are running when the command is issued. The options are a for all and u for user. Note the interesting syntax that there must not be a space between the u option and the username, student.

```
[student@studentvm1 ~]$ ps -austudent
  PID TTY          TIME CMD
 2272 pts/7    06:23:28 cpuHog
 2273 pts/8    06:23:42 cpuHog
 2277 pts/11   06:23:14 cpuHog
 2278 pts/14   00:00:00 bash
 2302 pts/15   00:00:00 bash
 2692 pts/16   00:00:00 bash
 2845 pts/14   06:17:17 cpuHog
 2848 pts/16   06:20:10 cpuHog
 4873 ?        00:00:00 systemd
 4875 ?        00:00:00 (sd-pam)
 4880 ?        00:00:00 pulseaudio
```

What do we do once we have found the process(es) that we are looking for? We usually kill or renice them. You have already used both of these so you won't do that again. Let's look at a couple additional useful and interesting commands.

You should have a few cpuHogs still running and we want to find just those. The **pgrep** command lists the PID numbers for each process whose name matches the pattern specified as the argument.

```
[student@studentvm1 ~]$ pgrep cpuHog
2272
2273
2277
2845
2848
5096
5162
5285
5314
6006
[student@studentvm1 ~]$
```

That is all – nothing else, just the PIDs. You could use the -i option to ignore case in the names which would mean not having to be case specific when typing the argument. The -l (lowercase El) to list the names as well. It might be good to do this if several types of running processes might match the argument.

```
[student@studentvm1 ~]$ pgrep -l cpu
8 mm_percpu_wq
14 cpuhp/0
15 cpuhp/1
2272 cpuHog
2273 cpuHog
2277 cpuHog
2845 cpuHog
2848 cpuHog
5096 cpuHog
5162 cpuHog
5285 cpuHog
5314 cpuHog
6006 cpuHog
[student@studentvm1 ~]$
```

Just be careful that you know what will be killed or altered with some other commands, like this next one.

```
[student@studentvm1 ~]$ renice +4 $(pgrep cpuH)
```

What do you think that the preceding command does? Use one of the interactive monitors like top or htop to verify this. Here are the results from my VM.

```
2272 (process ID) old priority 0, new priority 4
2273 (process ID) old priority 0, new priority 4
2277 (process ID) old priority 0, new priority 4
2845 (process ID) old priority 0, new priority 4
2848 (process ID) old priority 0, new priority 4
renice: failed to set priority for 5096 (process ID): Permission denied
5162 (process ID) old priority 0, new priority 4
5285 (process ID) old priority 0, new priority 4
5314 (process ID) old priority 0, new priority 4
6006 (process ID) old priority 0, new priority 4
```

I had one cpuHog that was not killed because it was running as root and the student user does not have the authority to kill root or any other user's processes. Note that the **pgrep** command found all of the cpuHogs, but we could have used the -U option to specify that we only wanted to list those matching processes that were also running as the student user. For this reason, it is wise to be very careful when running commands like these as root so that you do not kill processes that belong to a user who does not want them terminated.

We have one more interesting command that we can use to kill multiple processes even if the number of them that are running and their PIDs are unknown. The **pkill** utility has the same matching capabilities as **pgrep**, but it simply sends the specified signals to the matching processes. The default is Signal 15, SIGTERM. The following command kills all running processes that match the string "cpuH."

```
[student@studentvm1 ~]$ pkill cpuH
```

At this point there should be no cpuHog instances running that belong to the student user.

## The impact of measurement

The observer effect[7] is a theory in the scientific discipline of physics that states, "simply observing a situation or phenomenon necessarily changes that phenomenon." This is also true when measuring Linux system performance.

The act of using these monitoring tools alters the system's use of resources including memory and CPU time. top and most of these monitors use perhaps 2% or 3% of a system's CPU time. The Glances utility has much more impact than the others and can use between 10% and 20% of CPU time; I have seen it use as much as 40% of one CPU in a very large and active system with 32 CPUs. Be sure to consider this when choosing your tools.

## Chapter summary

As we proceeded through this chapter, you probably observed some things about the processes running on your VM and the total amount of time that accrued to each. The cpuHogs together accumulated most of the CPU time, while the kernel threads accumulated very little by comparison. This is because most of the kernel threads do not need to run frequently and take very little time when they do. Other tools that don't accumulate much time, such as LibreOffice, simply spend most of their time waiting for the users to type or select tasks from menus or icon bars.

Be sure to read the man pages for each of the monitors we have experimented with in this chapter because there is a large amount of information about configuring and interacting with them. These tools are the ones I like best for managing processes but there are more.

These programs can tell you a great deal when you are looking for the cause of a problem. They can tell you when a process, and which one, is sucking up CPU time, whether there is enough free memory, whether processes are stalled while waiting for I/O such as disk or network access to complete, and much more.

I also highly recommend that you spend time watching these monitoring programs while they run on a system that is functioning normally. This way you will be able to differentiate those things that may be abnormal while you are looking for the cause of a problem. This is one of those tasks that may look to others like you are just sitting there doing nothing but which is an important part of being the lazy SysAdmin.

[7]Wikipedia, *Observer effect (physics)*, https://en.m.wikipedia.org/wiki/Observer_effect_(physics)

# Exercises

Complete the following exercises to finish this chapter:

1. There is a specification in the Linux FHS that defines a location for personal executable files that is within your own home directory structure. This allows typing the executable name without a directory path preceding it. What is it?
2. Set up your home directory structure in accordance with the FHS and move the executable files to that location. Test launching the cpuHog script from this new location without using a path.
3. Start an instance of top and set the refresh delay to 1 second, then observe the output for a few minutes. How much memory and swap space are free?
4. As both root and the student user, use the nice command to start instances of the cpuHog program with negative numbers. Does it work as you expected?
5. Why are non-root users restricted from lowering the nice numbers of their own processes?
6. What default PID is used by top when you use k to kill or r to renice a process?
7. What is the result of reducing the number of virtual CPUs allocated to the StudentVM1 virtual machine and then rerunning Experiment 4-4?
8. When using atop, what key would you use to freeze the display for longer inspection than the current interval would allow?
9. The htop utility allows filtering the process list so that only those that match the filter specification are displayed. Use that function to display only the running cpuHog instances.
10. Can multiple processes be terminated with a single kill command?
11. Do the accumulated times in the TIME+ columns of top and htop add up to the total uptime? Why?

CHAPTER 5

# Special Filesystems

## Objectives

In this chapter you will learn

- What constitutes a special filesystem
- Practical uses for two of the special filesystems, /proc and /sys
- The use of some of the tools that allow easy access to view system data in these special filesystems
- How to create and manage swap files
- Some differing recommendations for swap size

## Introduction

In Chapter 4, we looked at some tools like **`top`** that allow SysAdmins to look inside the running Linux kernel. We also discussed the observer effect[1] which is a theory in the scientific discipline of physics that states, "simply observing a situation or phenomenon necessarily changes that phenomenon." This is also true when measuring Linux system performance. The act of using those monitoring tools alters the system's use of resources including memory and CPU time.

Collecting data does not impact the overall performance in a Linux host. The Linux kernel is designed to always collect and store the performance data that is merely accessed and displayed by any and all performance monitoring tools. It is the tools' access of that data to read it and then to manipulate it and display it in a meaningful format that further affects the system performance.

[1]Wikipedia, *Observer effect (physics)*, https://en.m.wikipedia.org/wiki/Observer_effect_(physics)




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_5

Linux has some special filesystems that it creates at each boot, two of which are particularly interesting to SysAdmins: the /proc and /sys filesystems. It is in these filesystems that the kernel stores the performance data of the running kernel and much more. The data is always there and it is easy to access. These are virtual filesystems that exist only in RAM while the Linux host is running; they do not exist on any physical disk. Because they exist only in RAM, these filesystems are not persistent like filesystems that are stored on the hard drive. They disappear when the computer is turned off and are recreated anew each time Linux starts up.

The /proc, /sys, and swap filesystems are ones with which you will become well acquainted as a SysAdmin so we are going to explore them in some detail in this chapter.

# The /proc filesystem

The /proc filesystem is defined by the FHS, which we explored in Chapter 19, as the location for Linux to store information about the system, the kernel, and all processes running on the host. It is intended to be a place for the kernel to expose information about itself in order to facilitate access to data about the system. It is also designed to provide access to view kernel configuration parameters and to modify many of them when necessary in order to allow the SysAdmin to tune the running system without needing to perform reboots after making changes.

When used as a window into the state of the operating system and its view of the system and hardware, it provides easy access to virtually every bit of information you might want as a SysAdmin.

**EXPERIMENT 5-1**

For best results with this experiment, it must be performed as root.

Let's first look at the top-level contents of the /proc filesystem of a running Linux host. On your host you may see color coding to differentiate files from directories.

First, look at the numeric entries. The names of these directories are a PID, or process ID number. Each of those PID directories contains information about the running process that it represents. We will look at these directories in more detail in Experiment 5-2.

```
[root@studentvm1 proc]# cd /proc ; ls
1      124    20     26122  2692  4940  836  946        driver          pagetypeinfo
10     1256   21     26141  27    4968  846  95         execdomains     partitions
100    14     22     26143  28    5037  847  950        fb              sched_debug
1007   1408   2278   26153  29    5135  848  96         filesystems     schedstat
1008   14402  23     26158  3     516   849  961        fs              scsi
101    14831  2302   26159  30    5163  851  963        interrupts      self
102    14844  24     26166  31    5230  852  968        iomem           slabinfo
103    15     25     26167  32    5258  874  97         ioports         softirqs
104    16     25882  26171  33    526   875  98         irq             stat
105    17     25966  26174  34    5287  878  987        kallsyms        swaps
1060   17105  25969  26175  35    537   880  99         kcore           sys
107    17517  25976  26179  36    554   899  994        keys            sysrq-trigger
1075   17518  26     26180  386   555   9    995        key-users       sysvipc
109    17559  26042  26186  39    593   900  996        kmsg            thread-self
1090   17607  26043  26189  394   594   901  997        kpagecgroup     timer_list
1092   17649  26053  26191  4     6     902  acpi       kpagecount      tty
1096   17653  26054  26194  40    653   903  asound     kpageflags      uptime
11     17700  26059  26203  450   678   905  buddyinfo  latency_stats   version
1105   17704  26060  26205  4868  747   906  bus        loadavg         vmallocinfo
111    17706  26066  26209  4873  765   907  cgroups    locks           vmstat
113    17711  26067  26216  4875  767   908  cmdline    mdstat          zoneinfo
11594  17712  26073  26220  4880  769   909  consoles   meminfo
11598  17779  26103  26228  4881  794   929  cpuinfo    misc
117    17780  26104  26282  4882  8     930  crypto     modules
118    18218  26110  26363  4932  833   931  devices    mounts
12     19     26116  26415  4938  834   932  diskstats  mtrr
1233   2      26121  26483  4939  835   94   dma        net
[root@studentvm1 proc]#
```

Each of the files in the /proc directory contains information about some part of the kernel. Let's take a look at a couple of these files, cpuinfo and meminfo.

The cpuinfo file is mostly static. It contains the specifications for all installed CPUs.

```
[root@studentvm1 proc]# cat cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
```

```
model           : 58
model name      : Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
stepping        : 9
microcode       : 0x19
cpu MHz         : 3392.345
cache size      : 8192 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx rdtscp lm constant_tsc
rep_good nopl xtopology nonstop_tsc cpuid pni pclmulqdq monitor ssse3 cx16
sse4_1 sse4_2 popcnt aes xsave avx rdrand lahf_lm
bugs            :
bogomips        : 6784.69
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
<snip>
```

The data from the cpuinfo file includes the processor ID and model, its current speed in MHz, and the flags that can be used to determine the CPU features. Now let's look at memory. First **cat** the meminfo file and then use the **free** command to do a comparison.

```
[root@studentvm1 proc]# cat meminfo
MemTotal:        4044740 kB
MemFree:         2936368 kB
MemAvailable:    3484704 kB
Buffers:          108740 kB
Cached:           615616 kB
```

```
SwapCached:            0 kB
Active:           676432 kB
Inactive:         310016 kB
Active(anon):     266916 kB
Inactive(anon):      316 kB
Active(file):     409516 kB
Inactive(file):   309700 kB
Unevictable:        8100 kB
Mlocked:            8100 kB
SwapTotal:       4182012 kB
SwapFree:        4182012 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        270212 kB
Mapped:           148088 kB
Shmem:               988 kB
Slab:              80128 kB
SReclaimable:      64500 kB
SUnreclaim:        15628 kB
KernelStack:        2272 kB
PageTables:        11300 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     6204380 kB
Committed_AS:     753260 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
```

```
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       73664 kB
DirectMap2M:     4120576 kB
[root@studentvm1 proc]# free
              total        used        free      shared  buff/cache   available
Mem:        4044740      304492     2935748         988      804500     3484100
Swap:       4182012           0     4182012
```

There is a lot of information in the /proc/meminfo file. A few bits of that data are used by programs like the **free** command. If you want the complete picture of memory usage, look in /proc/meminfo. The **free** command, like **top**, **htop**, and many other core utilities, gets its data from the /proc filesystem.

Run the **cat meminfo** command several times in quick succession to see that the /proc/meminfo file is continuously changing. That indicates the file is being updated. You can do this with the **watch** command.

```
[root@studentvm1 proc]# watch cat meminfo
```

---

**Note** While doing research for this experiment, I discovered that a method I had used before to determine that files were being updated, even when the content had not changed, no longer worked. The command that I used was stat /proc/meminfo which should have shown continuously changing mtime, atime, and ctime, but which no longer did. This does work correctly in CentOS and Fedora 27 but not Fedora 28 or 29. I reported this problem as bug 1675440 on the Red Hat Bugzilla web site. It is very important that we SysAdmins report bugs when we find them.

---

Because the data in /proc is a nearly instantaneous picture of the state of the Linux kernel and the computer hardware, the data may change rapidly. Look at the interrupts file several times in a row.

---

Spend a little time to compare the data in the /proc/meminfo file against the information you get when using commands like **free** and **top**. Where do you think these utility tools and many others get their information? Right here in the /proc filesystem, that's where.

Let's look a little bit deeper into PID 1. Like all of the process directories, it contains information about the process with that ID. So let's look at some of that information.

## EXPERIMENT 5-2

Start this experiment as root.

Let's enter and look at the contents of the /proc/1 directory. Then view the contents of the cmdline file.

```
[root@studentvm1 proc]# cd /proc/1 ; cat cmdline
/usr/lib/systemd/systemd--switched-root--system--deserialize24
```

We can see from the contents of the cmdline that this is systemd, the mother of all programs. On all older and some current versions of Linux, PID 1 will be the init program.

If there are no cpuHogs running, start one instance in a terminal session as the student user. Use one of the monitoring tools like top to determine the PID of this cpuHog process. On my VM, the PID is 18107, but it will be different on your VM. Be sure to use the correct PID for the cpuHog on your VM.

Make the directory corresponding to the PID of your cpuHog instance the PWD. Then list the contents.

```
[root@studentvm1 18107]# cd /proc/18107 ; ll | less
total 0
dr-xr-xr-x. 2 student student 0 Feb 11 20:29 attr
--w-------. 1 student student 0 Feb 11 20:29 clear_refs
-r--r--r--. 1 student student 0 Feb 11 20:29 cmdline
-rw-r--r--. 1 student student 0 Feb 11 20:29 comm
-rw-r--r--. 1 student student 0 Feb 11 20:29 coredump_filter
-r--r--r--. 1 student student 0 Feb 11 20:29 cpuset
lrwxrwxrwx. 1 student student 0 Feb 11 20:29 cwd -> /home/student
-r--------. 1 student student 0 Feb 11 20:29 environ
lrwxrwxrwx. 1 student student 0 Feb 11 20:29 exe -> /usr/bin/bash
dr-x------. 2 student student 0 Feb 11 20:29 fd
<snip>
```

Note the entries for cwd and exe. The cwd entry points to the current working directory, a.k.a. the PWD, for the process. The exe entry points to the executable file for the process, which is the Bash shell. But look at the content of the cmdline file.

```
[root@studentvm1 18107]# cat cmdline
/bin/bash./cpuHog
```

This tells us that the program that is running is the cpuHog. It also gives us some insight into the manner in which programs – at least shell scripts – are run in Linux. When starting a shell program, the systemd[2] program first launches a shell, the default being Bash unless otherwise specified, and the shell program, cpuHog, is provided as an argument to the command.

If you are not already using **top** to monitor the ongoing activities on your VM, start an instance of it now. Look for the COMMAND column which, as you can see in Figure 5-1, shows the three running instances of the cpuHog.

```
top - 09:02:58 up 9 days, 14:54, 16 users,  load average: 5.22, 5.15, 5.10
Tasks: 212 total,   5 running, 207 sleeping,   0 stopped,   0 zombie
%Cpu0  : 36.0 us, 60.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  2.0 hi,  2.0 si,  0.0 st
%Cpu1  : 11.2 us, 74.5 sy,  0.0 ni,  0.0 id,  0.0 wa,  8.2 hi,  6.1 si,  0.0 st
MiB Mem : 21.9/3942.5   [||||||||||||                                          ]
MiB Swap:  0.0/4096.0   [                                                      ]

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1105 root      20   0  398352  85676  38904 R   8.9   2.1   7:33.78 Xorg
   17 root      20   0       0      0      0 S   0.0   0.0   6:27.09 ksoftirqd/1
11969 student   20   0  231608   3276   2048 R  29.7   0.1   5:41.53 screen
12019 student   20   0  216336   3188   2940 R  23.8   0.1   5:35.10 bash
11993 student   20   0  216336   1200   1052 R  32.7   0.0   5:32.69 cpuHog
12043 student   20   0  216336   3132   2880 R  22.8   0.1   5:28.42 cpuHog
12070 student   20   0  218500   3000   2720 R  30.7   0.1   5:04.96 ksh
<snip>
```

***Figure 5-1.*** *The top command showing three cpuHogs. They are there but are not easy to identify*

It does show all three instances of the cpuHog, really. It is just not easy to identify two of them. To make it obvious, press the **c** key to display the complete command. The result can be seen in Figure 5-2.

[2]systemd is the program that deals with starting, stopping, and managing all other running processes. We will cover systemd in detail in Volume 2, Chapter 13.

```
top - 09:11:56 up 9 days, 15:03, 16 users,  load average: 5.46, 5.27, 5.14
Tasks: 212 total,   5 running, 207 sleeping,   0 stopped,   0 zombie
%Cpu0  : 47.1 us, 49.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  2.0 hi,  2.0 si,  0.0 st
%Cpu1  : 10.9 us, 74.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  8.9 hi,  5.9 si,  0.0 st
MiB Mem : 21.9/3942.5   [||||||||||||                                          ]
MiB Swap:  0.0/4096.0   [                                                      ]

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
11969 student   20   0  231608   3276   2048 R  26.5   0.1   5:13.00 SCREEN
12019 student   20   0  216336   3188   2940 R  35.3   0.1   5:07.35 bash cpuHog
11993 student   20   0  216336   1200   1052 R  28.4   0.0   5:07.29 /bin/bash ./cpuHog
12043 student   20   0  216336   3132   2880 R  27.5   0.1   5:02.09 /bin/bash ./cpuHog
12070 student   20   0  218500   3000   2720 S  30.4   0.1   4:37.82 ksh ./cpuHog
<snip>
```

***Figure 5-2.*** *After pressing the* ***c*** *key, all three cpuHogs are easy to identify*

The c key toggles display of the complete command line on and off. Now that we can see the command line, it is obvious that the cpuHogs have PIDs of 12019, 11993, 12043, and 12070.

The **htop** utility displays the command line by default so start **htop** and look at the Command column. You can immediately see the three cpuHogs. Be sure to make a note of the three PIDs for the cpuHogs. Now press **F5** to show the process tree which allows us to see the process hierarchy as in Figure 5-3.

```
<snip>
11969 student 20 0 226M 3276 2048 R 37.4 0.1 4:48.82 ├──SCREEN
12044 student 20 0 220M 4924 3456 S 0.0 0.1 0:00.03  │   ├──/bin/bash
12070 student 20 0 213M 3000 2720 R 34.8 0.1 4:15.28 │   │   └── ksh ./cpuHog
12020 student 20 0 220M 4704 3356 S 0.0 0.1 0:00.05  │   ├──/bin/bash
12043 student 20 0 211M 3132 2880 R 37.4 0.1 4:38.81 │   │   └── /bin/bash ./cpuHog
11994 student 20 0 220M 4932 3464 S 0.0 0.1 0:00.07  │   ├──/bin/bash
12019 student 20 0 211M 3188 2940 R 36.1 0.1 4:43.72 │   │   └── bash cpuHog
11970 student 20 0 220M 4724 3372 S 0.0 0.1 0:00.02  │   └──/bin/bash
11993 student 20 0 211M 1200 1052 R 37.4 0.0 4:43.18 │   └──/bin/bash ./cpuHog
<snip>
```

***Figure 5-3.*** *The htop process tree view clarifies the process hierarchy*

Once again, this helps us to understand a bit more about how Linux launches command-line programs. We can see that in all three cases that systemd starts a sub-shell and then launches the program within that sub-shell.

Another tool that allows us to view the process tree is the **pstree** utility. Use the pstree utility to view the process tree.

```
[root@studentvm1 ~]# pstree  -Acp | less
```

Figure 5-4 shows portions of the data stream from the **pstree** command. Scroll through the output and find the cpuHogs. You should check the man page for pstree to discover the meanings of the options we used for this command.

```
[root@studentvm1 ~]# pstree -Acp | less
systemd(1)-+-ModemManager(899)-+-{ModemManager}(926)
           |                   `-{ModemManager}(962)
           |-NetworkManager(1060)-+-dhclient(1233)
           |                      |-dhclient(1256)
           |                      |-{NetworkManager}(1072)
           |                      `-{NetworkManager}(1074)
           |-VBoxClient(26042)---VBoxClient(26043)---{VBoxClient}(26049)
           |-VBoxClient(26053)---VBoxClient(26054)
<snip>
           |-screen(11969)-+-bash(11970)---cpuHog(11993)
           |               |-bash(11994)---bash(12019)
           |               |-bash(12020)---cpuHog(12043)
           |               `-bash(12044)---ksh(12070)
           |-smartd(929)
<snip>
```

***Figure 5-4.*** *The* ***pstree*** *utility can also show the process tree*

Our real purpose here was to learn the PID of the cpuHogs in order to explore them in the /proc filesystem. Now that we know multiple ways to do that, let's get back to our original objective.

Pick one of the cpuHogs and, as root, make /proc/<PID> the PWD. I chose PID 12070, but you should use the PID for an instance of cpuHog on your VM, and then list the contents of the directory.

```
[root@studentvm1 ~]# cd /proc/12070 ; ls
attr            cpuset  latency   mountstats    personality  smaps_rollup  timerslack_ns
autogroup       cwd     limits    net           projid_map   stack         uid_map
auxv            environ loginuid  ns            root         stat          wchan
cgroup          exe     map_files numa_maps     sched        statm
clear_refs      fd      maps      oom_adj       schedstat    status
```

```
cmdline         fdinfo  mem       oom_score     sessionid   syscall
comm            gid_map mountinfo oom_score_adj setgroups   task
coredump_filter io      mounts    pagemap       smaps       timers
```

Take some time to explore the content of some of these files and subdirectories. Be sure to view the content of the status, limits, loginuid, and maps files. The maps file is a memory map that lists executable and library locations in virtual memory. The status file contains a great deal of information including some interesting data about virtual memory usage. Also take some time to explore a few of the other files and subdirectories in this and other PID directories.

There is a huge amount of information available in the /proc filesystem, and it can be used to good advantage to solve problems. In fact, the capability to make changes to the running kernel on the fly and without a reboot is a powerful tool. It allows you to make instant changes to the Linux kernel to resolve a problem, enable a function, or tune performance. Let's look at one example.

Linux is very flexible and can do many interesting things. One of those cool things is that any Linux host with multiple network interface cards (NICs) can act as a router. All it takes is a little knowledge, a simple command, and some changes to the firewall.

Routing is a task managed by the kernel. So turning it on (or off) requires that we change a kernel configuration parameter. Fortunately, we do not need to recompile the kernel, and that is one of the benefits of exposing the kernel configuration in the /proc filesystem. We are going to turn on IP forwarding which provides the kernel's basic routing functionality.

## EXPERIMENT 5-3

This little command-line program makes the /proc/sys/net/ipv4 directory the PWD, prints the current state of the ip_forward file which should be zero (0), sets it to "1," and then prints its new state which should be 1. Routing is now turned on. Be sure to enter the command on a single line.

```
[root@studentvm1 ipv4]# cd /proc/sys/net/ipv4 ; cat ip_forward ; echo 1 >
ip_forward ; cat ip_forward
0
1
```

**Warning** I intentionally chose to modify a kernel parameter that I am familiar with and that won't cause any harm to your Linux VM. As you explore the /proc filesystem, you should not make any further changes.

Congratulations! You have just altered the configuration of the running kernel.

In order to complete the configuration of a Linux host to full function as a router, additional changes would need to be made to the iptables firewall, or to whatever firewall software you may be using, and to the routing table. Those changes will define the specifics of the routing such as which packets get routed where. Although beyond the scope of this book, I have written an article[3] with some detail about configuring the routing table to which you can refer if you want more information. I also wrote an article[4] which briefly covers all of the steps required to turn a Linux host into a router, including making IP forwarding persistent after a reboot.

While you are here in the /proc filesystem, look around some more – follow your own curiosity to explore different areas of this important filesystem.

## The /sys filesystem

The /sys directory is another virtual filesystem that is used by Linux to maintain specific data for use by the kernel and SysAdmins. The /sys directory maintains the list of hardware hierarchically for each bus type in the computer hardware.

A quick look at the /sys filesystem shows us its basic structure.

[3]David Both, "An introduction to Linux network routing," https://opensource.com/business/16/8/introduction-linux-network-routing

[4]David Both, "Making your Linux Box Into a Router," https://www.linux-databook.info/?page_id=697

## EXPERIMENT 5-4

In this experiment we look briefly at the contents of the /sys directory and then one of its subdirectories, /sys/block.

```
[root@studentvm1 sys]# cd /sys
[root@studentvm1 sys]# ls
block  bus  class  dev  devices  firmware  fs  hypervisor kernel module power
[root@studentvm1 sys]# ls block
dm-0  dm-1  sda  sr0
```

There are different types of disk (block) devices in /sys/block, and the sda device is one of them. Let's take a quick look at some of the contents of the sda directory.

```
[root@studentvm1 sys]# ls block/sda
alignment_offset   events_async       queue      slaves
bdi                events_poll_msecs  range      stat
capability         ext_range          removable  subsystem
dev                holders            ro         trace
device             inflight           sda1       uevent
discard_alignment  integrity          sda2
events             power              size
[root@studentvm1 sys]# cat block/sda/dev
8:0
[root@studentvm1 sys]# ls block/sda/device
block                                ncq_prio_enable
bsg                                  power
delete                               queue_depth
device_blocked                       queue_ramp_up_period
device_busy                          queue_type
dh_state                             rescan
driver                               rev
eh_timeout                           scsi_device
evt_capacity_change_reported         scsi_disk
evt_inquiry_change_reported          scsi_generic
evt_lun_change_reported              scsi_level
evt_media_change                     state
evt_mode_parameter_change_reported   subsystem
evt_soft_threshold_reached           sw_activity
```

```
generic                          timeout
inquiry                          type
iocounterbits                    uevent
iodone_cnt                       unload_heads
ioerr_cnt                        vendor
iorequest_cnt                    vpd_pg80
modalias                         vpd_pg83
model                            wwid
[root@studentvm1 sys]# cat block/sda/device/model
VBOX HARDDISK
```

For a bit more realistic information from this last command, I also performed this on my own physical hard drive rather than the VM I have been using for these experiments and that looks like this.

```
[root@david proc]# cat /sys/block/sda/device/model
ST320DM000-1BD14
```

This information is more like what you would see on one of your own hardware hosts rather than a VM. Now let's use the `smartctl` command to show that same bit of information and more. I used my physical host for this due to the more realistic data. I have also trimmed a large amount of output from the end of the results.

```
[root@david proc]# smartctl -x /dev/sda
smartctl 6.5 2016-05-07 r4318 [x86_64-linux-4.13.16-302.fc27.x86_64] (local
build)
Copyright (C) 2002-16, Bruce Allen, Christian Franke, www.smartmontools.org

=== START OF INFORMATION SECTION ===
Model Family:     Seagate Barracuda 7200.14 (AF)
Device Model:     ST320DM000-1BD14C
Serial Number:    Z3TT43ZK
LU WWN Device Id: 5 000c50 065371517
Firmware Version: KC48
User Capacity:    320,072,933,376 bytes [320 GB]
Sector Sizes:     512 bytes logical, 4096 bytes physical
Rotation Rate:    7200 rpm
Device is:        In smartctl database [for details use: -P show]
ATA Version is:   ATA8-ACS T13/1699-D revision 4
SATA Version is:  SATA 3.0, 6.0 Gb/s (current: 6.0 Gb/s)
```

```
Local Time is:    Wed Dec 13 13:31:36 2017 EST
SMART support is: Available - device has SMART capability.
SMART support is: Enabled
AAM level is:     208 (intermediate), recommended: 208
APM feature is:   Unavailable
Rd look-ahead is: Enabled
Write cache is:   Enabled
ATA Security is:  Disabled, frozen [SEC2]
Wt Cache Reorder: Enabled

=== START OF READ SMART DATA SECTION ===
SMART overall-health self-assessment test result: PASSED

General SMART Values:
<snip>
```

Had I not cut of the end of the results from this last command, it would also show things like failure indicators and a temperature history which can be helpful in determining the source of hard drive problems. The data for a virtual hard drive on your VM would be different and significantly less interesting.

The **`smartctl`** utility obtains the data it uses from the /sys filesystem, just as other utility programs obtain their data from the /proc filesystem.

The /sys filesystem contains data about the PCI and USB system bus hardware and any attached devices. The kernel can use this information to determine which device drivers to use for one example.

## EXPERIMENT 5-5

Let's look at some information about one of the buses on the computer, the USB bus. I am going to skip right to the locations of the devices in the /sys filesystem; you may need to do a little exploring on your own to find the items that interest you.

```
[root@studentvm1 ~]# ls /sys/bus/usb/devices/usb2
2-0:1.0              bMaxPacketSize0     driver                    quirks
authorized           bMaxPower           ep_00                  removable
authorized_default   bNumConfigurations  idProduct                 remove
avoid_reset_quirk    bNumInterfaces      idVendor                  serial
```

```
bcdDevice            busnum              interface_authorized_default   speed
bConfigurationValue  configuration       ltm_capable                 subsystem
bDeviceClass         descriptors         manufacturer                   uevent
bDeviceProtocol      dev                 maxchild                       urbnum
bDeviceSubClass      devnum              power                        version
bmAttributes         devpath             product
```

The preceding results show some of the files and directories which provide data about that particular device. But there is an easier way by using the core utilities so that we don't have to do all that exploration on our own. Once again this is from my own physical workstation.

If you do not have a usb2 directory or it is empty, that might be because the VirtualBox extensions were not installed. In that case, try this experiment in the **/sys/bus/usb/devices/usb** directory.

```
[root@david ~]# lsusb
Bus 002 Device 005: ID 1058:070a Western Digital Technologies, Inc. My
Passport Essential (WDBAAA), My Passport for Mac (WDBAAB), My Passport
Essential SE (WDBABM), My Passport SE for Mac (WDBABW)
Bus 002 Device 004: ID 05e3:0745 Genesys Logic, Inc. Logilink CR0012
Bus 002 Device 003: ID 1a40:0201 Terminus Technology Inc. FE 2.1 7-port Hub
Bus 002 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 006 Device 005: ID 0bc2:ab1e Seagate RSS LLC Backup Plus Portable Drive
Bus 006 Device 003: ID 2109:0812 VIA Labs, Inc. VL812 Hub
Bus 006 Device 002: ID 2109:0812 VIA Labs, Inc. VL812 Hub
Bus 006 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 005 Device 007: ID 2109:2812 VIA Labs, Inc. VL812 Hub
Bus 005 Device 004: ID 2109:2812 VIA Labs, Inc. VL812 Hub
Bus 005 Device 006: ID 04f9:0042 Brother Industries, Ltd HL-2270DW Laser Printer
Bus 005 Device 005: ID 04f9:02b0 Brother Industries, Ltd MFC-9340CDW
Bus 005 Device 003: ID 050d:0234 Belkin Components F5U234 USB 2.0 4-Port Hub
Bus 005 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 005: ID 046d:c52b Logitech, Inc. Unifying Receiver
Bus 001 Device 006: ID 17f6:0822 Unicomp, Inc
Bus 001 Device 003: ID 051d:0002 American Power Conversion Uninterruptible
                                 Power Supply
Bus 001 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
```

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 010: ID 0424:4063 Standard Microsystems Corp.
Bus 003 Device 009: ID 0424:2640 Standard Microsystems Corp. USB 2.0 Hub
Bus 003 Device 008: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Go ahead and try the **`lspci`** command on your own.

I sometimes find it helpful to find specific hardware devices, especially the newly added ones. As with the /proc directory, there are some core utilities like `lsusb` and `lspci` that make it easy for us to view information about the devices connected to the host.

## Swap space

Swap space is a common aspect of computing today, regardless of operating system. Linux uses swap space to increase the amount of virtual memory available to a host. It can use one or more dedicated swap partitions or a swap file on a regular filesystem or logical volume.

There are two basic types of memory in a typical computer. Random Access Memory (RAM) is used to store data and programs while they are being actively used by the computer. Programs and data cannot be used by the computer unless they are stored in RAM. RAM is volatile memory; that is, the data stored in RAM is lost if the computer is turned off.

Hard drives are magnetic media used for long-term storage of data and programs, and SSDs are the solid state equivalent. Magnetic and SSD media are nonvolatile; the data stored on a disk[5] remains even when power is removed from the computer. The CPU cannot directly access the programs and data on the hard drive; it must be copied into RAM first and that is where the CPU can access its programming instructions and the data to be operated on by those instructions. During the boot process, a computer copies specific operating system programs such as the kernel and init or systemd and

[5]I use the term “disk” to refer to both spinning hard drives and SSDs.

data from the hard drive into RAM where it is accessed directly by the computer's processor, the CPU (Central Processing Unit).

Swap space is the second type of memory in modern Linux systems. The primary function of swap space is to substitute disk space for RAM memory when real RAM fills up and more space is needed. For example, assume you have a computer system with 8GB of RAM. If you start up programs that don't fill that RAM, everything is fine and no swapping is required. But say that a hypothetical very large spreadsheet you are working on grows when you add more rows to it, and that plus everything else you have running now fills all of RAM. Without swap space available, you would have to stop work on the spreadsheet until you could free up some of your limited RAM by closing down some other programs.

The kernel uses a memory management program that locates blocks, a.k.a. pages, of memory in which the contents have not been used recently. The memory management program swaps enough of these relatively infrequently used pages of memory out to a special partition on the hard drive specifically designated for "paging" or swapping. This frees up RAM and makes room for more data to be entered into your spreadsheet. Those pages of memory swapped out to the hard drive are tracked by the kernel's memory management code and can be paged back into RAM if they are needed.

The total amount of memory in a Linux computer is the RAM plus active swap space and is referred to as virtual memory. Linux supports up to 32 swap areas, any or all of which can be online at the same time. A swap area can be a disk partition, a logical volume, or a file in a non-swap partition or volume. Multiple swap areas are usually referred to collectively as "swaps," such as "all active swaps."

## Types of Linux swap

Linux provides two types of swap area. By default, most Linux installations create a swap partition or volume, but it is also possible to use a specially configured file as a swap file. A swap partition is just what its name implies – a standard disk partition that is designated as swap space by the **`mkswap`** command. A logical volume used as a swap area works just like a standard disk partition for use as a swap area, but its size can be extended like any logical volume.

A swap file can be used if there is no free disk space in which to create a new swap partition or space in a volume group in which a logical volume can be created for swap space. This is just a regular file that is created and preallocated to a specified size.

Then the **mkswap** command is run to configure it as swap space. I don't recommend using a file for swap space unless absolutely necessary. A swap file may be a reasonable choice on systems with a lot of physical memory that never approaches filling up. Disk space is so cheap and plentiful now; there's no reason not to set up a permanent swap partition.

## Thrashing

Thrashing can occur when total virtual memory, both RAM and swap space, become nearly full. The system spends so much time paging blocks of memory between swap space and RAM and back; that little time is left for real work.

The typical symptoms of this are fairly obvious; the system becomes completely unresponsive or quite slow, and the hard drive activity light is on almost constantly. If you can manage to issue a command like free that shows CPU load and memory usage, you will see that the CPU load is very high, perhaps as much as 30 to 40 times the number of CPU cores in the system. Another symptom is that both RAM and swap space are almost completely allocated.

After the fact, looking at SAR (System Activity Report) data can also show these symptoms. I install SAR on every system I work on and use it for post-repair forensic analysis. We explored SAR in Volume 1, Chapter 13.

## What is the right amount of swap space?

Many years ago, the rule of thumb for the amount of swap space that should be allocated on the hard drive was 2X the amount of RAM installed in the computer. Of course that was when computers typically had RAM amounts measured in KB or MB. So if a computer had 64KB of RAM, a swap partition of 128KB would be an optimum size. This rule of thumb took into account the fact that RAM memory sizes were typically quite small at that time and the fact that allocating more than 2X RAM for swap space did not improve performance. With more than twice RAM for swap, most systems spent more time thrashing than actually performing useful work.

RAM memory has become a relatively inexpensive commodity, and most computers these days have amounts of RAM that extend into tens of gigabytes. Most of my newer computers have at least 8GB of RAM, one has 32GB, and my main workstation has 64GB. My older computers have from 4GB to 8GB of RAM.

When dealing with computers having huge amounts of RAM, the limiting performance factor for swap space is far lower than the 2X multiplier. The Fedora 28 online Installation Guide, which can be found online in the Fedora user documentation[6] site in the *Fedora Installation Guide*, defines current thinking about swap space allocation. I have included in the following some discussion and the table of recommendations from that document.

Figure 5-5 provides the recommended size of a swap partition depending on the amount of RAM in your system and whether you want sufficient memory for your system to hibernate. The recommended swap partition size is established automatically during installation. To allow for hibernation, however, you will need to edit the swap space in the custom partitioning stage.

| Amount of system RAM | Recommended swap space | Recommended swap with hibernation |
|---|---|---|
| less than 2 GB | 2 times the amount of RAM | 3 times the amount of RAM |
| 2 GB - 8 GB | Equal to the amount of RAM | 2 times the amount of RAM |
| 8 GB - 64 GB | 0.5 times the amount of RAM | 1.5 times the amount of RAM |
| more than 64 GB | workload dependent | hibernation not recommended |

***Figure 5-5.*** *Recommended system swap space in Fedora 28 documentation*

At the border between each range listed earlier (e.g., a system with 2GB, 8GB, or 64GB of system RAM), discretion can be exercised with regard to chosen swap space and hibernation support. If your system resources allow for it, increasing the swap space may lead to better performance.

Of course most Linux administrators have their own ideas about the appropriate amount of swap space – as well as pretty much everything else. Figure 5-6 contains my own recommendations based on my personal experiences in multiple environments. These may not work for you but, along with Figure 5-5, they may help you get started.

[6]Fedora Documentation, https://docs.fedoraproject.org/en-US/docs/

| Amount of RAM | Recommended swap space |
|---|---|
| ⩽ 2GB | 2X RAM |
| 2GB – 8GB | = RAM |
| >8GB | 8GB |

***Figure 5-6.*** *Recommended system swap space per the author*

One consideration in both tables is that as the amount of RAM increases, beyond a certain point adding more swap space simply leads to thrashing well before the swap space even comes close to being filled. If you have too little virtual memory while following these recommendations, you should add more RAM, if possible, rather than more swap space. As with all recommendations that affect system performance, you should use what works best for your specific environment. This will take time and effort to experiment and make changes based on the conditions in your Linux environment.

I mentioned that all Linux SysAdmins have their own ideas about swap space; Chris Short, one of my friends and a fellow community moderator at Opensource.com, pointed me to an old article[7] where he recommended using 1GB for swap space. Chris told me that he now recommends zero swap space.

So I got curious and created a poll[8] that was published on Opensource.com. Read the article and especially the comments to more fully understand the range of thought about swap space, but the 2164 votes tallied as of this writing pretty much tell the story. The results of that poll are shown in Figure 5-7.

You can formulate your own opinion about how much swap space is the right amount, but sometimes the amount currently available may not be enough. Let's look at how to add more swap space.

[7]Short, Chris, *Moving to Linux – Partitioning*, https://chrisshort.net/moving-to-linux-partitioning/

[8]Both, David, Opensource.com, *What's the right amount of swap space for a modern Linux system?*, https://opensource.com/article/19/2/swap-space-poll

| Amount of Swap Space | Votes |
|---|---|
| Zero | 413 |
| <1GB | 69 |
| 1GB | 73 |
| 2GB | 173 |
| 4GB | 258 |
| 8GB | 172 |
| Something similar to Figure 5-5 | 304 |
| Something similar to Figure 5-6 | 343 |
| Whatever my distro creates at installation time | 198 |
| What is swap space? | 38 |
| I don't care | 38 |
| Other—please explain in the comment section | 85 |

***Figure 5-7.*** *Opensource.com swap space poll results*

## Adding more swap space on a non-LVM disk partition

Due to changing requirements for swap space on hosts with Linux already installed, it may become necessary to modify the amount of swap space defined for the system. This procedure can be used for any general case where the amount of swap space needs to be increased. It assumes sufficient available disk space is available. This procedure also assumes that the disks are partitioned in "raw" EXT4 and swap partitions and do not use logical volume management (LVM).

The basic steps to take are simple and a reboot should not be necessary:

1. Turn off the existing swap space.
2. Create a new swap partition of the desired size.

3. Re-read the partition table.
4. Configure the partition as swap space.
5. Add the new partition/etc/fstab.
6. Turn on swap.

For safety sake, before turning off swap, at the very least you should ensure that no applications are running and that no swap space is in use. The free or top commands can tell you whether swap space is in use. To be even more safe, you could revert to the systemd rescue target which is the same as runlevel 1 using the old SystemV init system.

## EXPERIMENT 5-6

Perform this experiment as root. Although it would be safer in a production environment to take the system down to the rescue.target, it is not necessary in our student virtual machines. Turn off the swap partition with the **swapoff** command. The -a option turns off all swap space.

```
[root@studentvm1 ~]# swapoff -a
```

Find the current swap partition and look for a partition in which to create a new swap partition.

```
[root@studentvm1 ~]# lsblk -i
NAME                        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                           8:0    0   60G  0 disk
|-sda1                        8:1    0    1G  0 part /boot
`-sda2                        8:2    0   59G  0 part
  |-fedora_studentvm1-root 253:0    0    2G  0 lvm  /
  |-fedora_studentvm1-swap 253:1    0    4G  0 lvm
  |-fedora_studentvm1-usr  253:2    0   15G  0 lvm  /usr
  |-fedora_studentvm1-home 253:3    0    4G  0 lvm  /home
  |-fedora_studentvm1-var  253:4    0   10G  0 lvm  /var
  `-fedora_studentvm1-tmp  253:5    0    5G  0 lvm  /tmp
sdb                           8:16   0   20G  0 disk
|-sdb1                        8:17   0    2G  0 part /TestFS
|-sdb2                        8:18   0    2G  0 part
`-sdb3                        8:19   0   16G  0 part
  `-NewVG--01-TestVol1     253:6    0    4G  0 lvm
```

```
sdc                          8:32   0    2G  0 disk
`-NewVG--01-TestVol1       253:6    0    4G  0 lvm
sdd                          8:48   0    2G  0 disk
`-sdd1                       8:49   0    2G  0 part
sr0                         11:0    1 1024M  0 rom
```

The results of the **lsblk** command show that our current swap partition is part of the fedora_studentvm1 volume group. There are also a couple options for our new swap partition. We used the sdb1 partition for a demonstration of creating new partitions, and it is currently mounted with an entry in /etc/fstab although we are not using it for anything at this time. The sdd1 partition is also available, and it is not mounted nor is there an entry in the fstab. Let's take the easy way and use /dev/sdd1 for our additional swap space.

We first need to change the partition type of sdd1, so start fdisk in interactive mode with the following command. Be sure to use the correct device based on the output from the **lsblk** command.

```
[root@studentvm1 ~]# fdisk /dev/sdd

Welcome to fdisk (util-linux 2.32.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help):
```

The sub-command **t** allows you to specify the type of partition. So enter **t** and press **Enter**. Then type **L** to get a list of all available partition types supported by Linux. Because there is only one partition in this small virtual drive, **fdisk** automatically selects partition 1.

```
Command (m for help): t
Selected partition 1
Hex code (type L to list all codes): L

 0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
 1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec (FAT-
 2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec (FAT-
 3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden or  c6  DRDOS/sec (FAT-
 4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
 5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
 6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS / .
 7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell Utility
```

```
 8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
 9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
 a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
 b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
 c  W95 FAT32 (LBA) 52  CP/M            a0  IBM Thinkpad hi ea  Rufus alignment
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         eb  BeOS fs
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ee  GPT
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        ef  EFI (FAT-12/16/
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f0  Linux/PA-RISC b
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f1  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f4  SpeedStor
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      f2  DOS secondary
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fb  VMware VMFS
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fc  VMware VMKCORE
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fd  Linux raid auto
1c  Hidden W95 FAT3 75  PC/IX           bc  Acronis FAT32 L fe  LANstep
1e  Hidden W95 FAT1 80  Old Minix       be  Solaris boot    ff  BBT
Hex code (type L to list all codes):
```

When it asks for the hex code partition type, type in **82**, which is the Linux swap partition type, and press **Enter**. I then use the p sub-command to list the partitions to ensure that the new partition type is correct.

```
Hex code (type L to list all codes): 82
Changed type of partition 'Linux' to 'Linux swap / Solaris'.

Command (m for help): p
Disk /dev/sdd: 2 GiB, 2147483648 bytes, 4194304 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb1f99266

Device     Boot Start     End Sectors Size Id Type
/dev/sdd1        2048 4194303 4192256   2G 82 Linux swap / Solaris

Command (m for help):
```

When you are satisfied with the partition you have created, use the w sub-command to write the new partition table to the disk. The fdisk program will exit and return you to the command prompt after it completes writing the revised partition table.

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

At this point in the real world, you might get an error message that the re-read of the partition table failed. If so, use the **partprobe** command to force the kernel to re-read the partition table. It is not necessary to perform a reboot to force the kernel to re-read partition table.

You should now be able to use the command **fdisk -l /dev/sdd** to list the partitions, and the new swap partition should be listed. Be sure that the partition type is "Linux swap."

It is necessary to modify the /etc/fstab file to point to the new swap partition. Add a new line to identify the new swap partition.

```
       /dev/sdd1          swap        swap    defaults        0 0
```

Be sure to use the correct partition number. Now you can perform the final step in creating the swap partition. Use the **mkswap** command to define the partition as a swap partition.

```
[root@studentvm1 ~]# mkswap /dev/sdd1
mkswap: /dev/sdd1: warning: wiping old ext4 signature.
Setting up swapspace version 1, size = 2 GiB (2146430976 bytes)
no label, UUID=dc4802a7-bb21-4726-a20b-be0fbf906b24
```

The final step is to turn swap on using the **swapon** command. The -a parameter turns on all swap partitions that are not already turned on.

```
[root@studentvm1 ~]# swapon -a
```

Your new swap partition is now online along with the previously existing swap partition. You can use some of the utility commands we have learned previously to verify this. These utilities should now show 6GB of swap space.

## Adding swap to an LVM disk environment

If your disk setup uses LVM, adding swap space will be fairly easy. Again, this assumes that space is available in the volume group in which the current swap volume is located. By default, the installation procedures for Fedora in an LVM environment creates the swap partition as a logical volume. This makes it easy because we can simply extend the size of the swap volume.

These are the basic steps required to extend the amount of swap space in an LVM environment:

1. Turn off all swap.
2. Add a new hard drive or SSD if necessary.
3. Prepare the new device if one was installed and extend the existing volume group on which the swap volume resides to include the new space.
4. Increase (extend) the size of the logical volume designated for swap.
5. Configure the resized volume as swap space.
6. Turn on swap.

We will assume in Experiment 5-7 that we do not need to add another drive device because, if we followed the directions when creating the VM, installing Linux, and performing the rest of these experiments, there should be plenty of space available.

**EXPERIMENT 5-7**

This experiment must be performed as root on StudentVM1.

First, since it is always wise to verify things before we start rather than to rely upon our assumptions, let's verify that swap exists and at least some of it is a logical volume using the **`lvs`** command (list logical volume). We did create some additional swap space on a raw partition in Experiment 5-6.

```
[root@studentvm1 ~]# lvs
  LV       VG                Attr       LSize  Pool Origin Data%  Meta%  Move
Log Cpy%Sync Convert
  TestVol1 NewVG-01          -wi-a----- <4.00g
  home     fedora_studentvm1 -wi-ao----  4.00g
  root     fedora_studentvm1 -wi-ao----  2.00g
  swap     fedora_studentvm1 -wi-ao----  4.00g
  tmp      fedora_studentvm1 -wi-ao----  5.00g
  usr      fedora_studentvm1 -wi-ao---- 15.00g
  var      fedora_studentvm1 -wi-ao---- 10.00g
[root@studentvm1 ~]#
```

You can see from this that the current size of the LVM swap volume is 4GB. In this case we want to extend this swap volume by 2GB. We also should check to see if there is space available in the volume group to extend the swap volume.

```
[root@studentvm1 ~]# vgs
  VG                #PV #LV #SN Attr   VSize   VFree
  NewVG-01            3   1   0 wz--n- <19.99g  15.99g
  fedora_studentvm1   1   6   0 wz--n- <59.00g <19.00g
```

We can see that there is about 19GB of space available on the fedora_studentvm1 volume group. That is plenty of space to add 2GB to our swap volume and still leave space for extending the other volumes, if that were ever needed.

We now want to stop existing swap for only the swap volume and leave the raw swap partition running. Nothing we do would affect the swap space we created on /dev/sdd1 so we can allow that to continue to be active.

However, we need to know how to address the swap device we want to turn off and I have discovered that the best – and possibly the coolest – way to do this is to use the data from /proc/swaps.

```
[root@studentvm1 ~]# cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/dm-1                               partition       4194300 0       -2
/dev/sdd1                               partition       2096124 0       -3
[root@studentvm1 ~]#
```

Another command we could use for this purpose is the **swapon** command.

```
[root@studentvm1 ~]# swapon -s
Filename                                Type            Size    Used    Priority
/dev/sdd1                               partition       2096124 0       -2
/dev/dm-1                               partition       6291452 0       -3
[root@studentvm1 ~]#
```

We can also use the /dev directory to verify that. In this case we see that the swap partition points to /dev/dm-1. The good thing about using the /proc/swaps file is that it tells us which swap partitions, volumes, or files are active. It seems pretty obvious that the **swapon -s** command just uses the data in the /proc/swaps file and sends it to STDOUT.

```
[root@studentvm1 ~]# ll /dev/mapper/
total 0
crw-------. 1 root root 10, 236 Feb  2 13:08 control
lrwxrwxrwx. 1 root root      7 Feb  2 13:08 fedora_studentvm1-home -> ../dm-3
lrwxrwxrwx. 1 root root      7 Feb  2 13:08 fedora_studentvm1-root -> ../dm-0
lrwxrwxrwx. 1 root root      7 Feb 14 11:50 fedora_studentvm1-swap -> ../dm-1
lrwxrwxrwx. 1 root root      7 Feb  2 13:08 fedora_studentvm1-tmp -> ../dm-5
lrwxrwxrwx. 1 root root      7 Feb  2 13:08 fedora_studentvm1-usr -> ../dm-2
lrwxrwxrwx. 1 root root      7 Feb  2 13:08 fedora_studentvm1-var -> ../dm-4
lrwxrwxrwx. 1 root root      7 Feb  2 13:08 NewVG--01-TestVol1 -> ../dm-6
[root@studentvm1 ~]#
```

When referring to the swap – or any other volume, for that matter – in commands, we can use /dev/dm-X or /dev/mapper/fedora_studentvm1-<LV Name>. We cannot refer to swap volumes or partitions by a mount point because they are not mounted like other filesystems.

Now we turn off the swap volume and verify that the swap volume is no longer active while the swap partition at /dev/sdd1 is still active.

```
[root@studentvm1 ~]# swapoff /dev/dm-1 ; cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/sdd1                               partition       2096124 0       -2
[root@studentvm1 ~]#
```

Now we can increase the size of the logical volume and verify the result.

```
[root@studentvm1 ~]# lvextend -L +2G /dev/mapper/fedora_studentvm1-swap
```

```
  Size of logical volume fedora_studentvm1/swap changed from 4.00 GiB (1024
extents) to 6.00 GiB (1536 extents).
  Logical volume fedora_studentvm1/swap successfully resized.
[root@studentvm1 ~]# lvs
  LV       VG                Attr       LSize  Pool Origin Data%  Meta%  Move
Log Cpy%Sync Convert
  TestVol1 NewVG-01          -wi-a----- <4.00g
  home     fedora_studentvm1 -wi-ao----  4.00g
  root     fedora_studentvm1 -wi-ao----  2.00g
  swap     fedora_studentvm1 -wi-a-----  6.00g
  tmp      fedora_studentvm1 -wi-ao----  5.00g
  usr      fedora_studentvm1 -wi-ao---- 15.00g
  var      fedora_studentvm1 -wi-ao---- 10.00g
[root@studentvm1 ~]#
```

Run the **mkswap** command to make this entire 6GB partition into swap space.

```
[root@studentvm1 ~]# mkswap /dev/mapper/fedora_studentvm1-swap
mkswap: /dev/mapper/fedora_studentvm1-swap: warning: wiping old swap
signature.
Setting up swapspace version 1, size = 6 GiB (6442446848 bytes)
no label, UUID=696bc20e-71d6-45ef-a2b8-ef49b55a6a90
[root@studentvm1 ~]#
```

Turn swap back on. We could use the device file as the argument, but it is easier just to use -a for all to turn on any swaps that are off. You might use the

```
[root@studentvm1 ~]# swapon -a
[root@studentvm1 ~]# cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/dm-1                               partition       6291452 0       -3
/dev/sdd1                               partition       2096124 0       -2
[root@studentvm1 ~]#
```

Notice the priority settings in the rightmost column. This allows setting priorities for different swaps. For example, you might want the swap on a fast device to be the highest priority so that it gets used before other slower swaps. By default, higher priorities are used by the swapping mechanism before the lower priorities. Higher numbers mean higher priority. Also, when creating new swap areas, newer swap areas are always assigned a lower priority than the lowest existing one.

Before we proceed any further, read the parts on priority in the man pages in both Sections 2 and 8 for swapon

```
[root@studentvm1 ~]# man 2 swapon
```

and

```
[root@studentvm1 ~]# man 8 swapon
```

In neither document does it mention priority numbers less than -1 so I am not sure whether the -3 and -2 priorities are bugs or an undocumented extension. Change the swap lines in /etc/fstab to add priorities to the mount options as shown in the following.

```
/dev/mapper/fedora_studentvm1-swap swap    swap    pri=5,defaults        0 0
/dev/sdd1                          swap    swap    pri=2,defaults        0 0
```

Now stop all swaps and then start all swaps. Verify the new swap priorities.

## Other swap options with LVM

When the need arises to add swap space to an existing system, there may be no available disk space. In such a case, it is necessary to install a new disk device to hold the new swap space. This could be added as an extension of an existing logical volume already being used for a swap volume, or it could be used as a separate volume or partition.

My preference would be to add the new space as a logical volume and make it a higher priority than the existing swap area. This enables you to expand that swap as part of the new volume and eventually deactivate the old swap.

# Chapter summary

It is not possible to cover all of the possibilities that exist in these special filesystems. Hopefully this chapter will have at least helped you understand the vast amount of information that is available and the essential openness of Linux as an open source operating system that makes it possible to expose all of its internal data and the ability it gives us to alter the configuration of the running kernel.

Swap space and the philosophies and preferences that have accumulated around it have given rise to a situation in which every SysAdmin you ask will probably have a different answer to questions about how much swap space is the right amount. Many SysAdmins even recommend zero swap space.

My opinion is that regardless of how much RAM is installed in a system, having some minimal amount of swap space is a good idea. It is better to have the system slow down when RAM fills and swapping starts than it is to have a program or the entire system crash. I consider swap as an early warning system that tells me when I need to add more RAM memory. Of course there are limits on the amount of memory that can be added to even modern Linux hosts.

# Exercises

Perform the following exercises to complete this chapter:

1. Where do the utilities that display data about the running Linux system get all of their data?
2. What is the overall function of the /proc filesystem?
3. What type of overhead is incurred by utilities like swap and glances when they are running?
4. Does the sequence in which swap partitions appear in /etc/fstab affect their priority if no priorities are assigned in fstab?
5. What size swap space would you use for the StudentVM1 virtual machine based on the recommendations in this chapter?
6. Assume hypothetically that you are installing a new Linux host with 12GB of RAM installed. How much swap space would you create during installation?
7. If no priorities are specified in /etc/fstab for all swaps, does the sequence in which the swaps are started affect their priority?

CHAPTER 6

# Regular Expressions

## Objectives

In this chapter you will learn

- To define the term “regular expression”
- To describe the purpose of regular expressions and extended regular expressions
- To differentiate between different styles of regular expressions as used by different tools
- To state the difference between basic regular expressions and extended regular expressions
- To identify and use many of the metacharacters and expressions used to build regular expressions for typical administrative tasks
- To use regular expressions and extended regular expressions with tools like **grep** and **sed**

## Introducing regular expressions

In Volume 1 of this course, we explored the use of file name globbing using wildcard characters like * and ? as a means to select specific files or lines of data from a data stream. We have also seen how to use brace expansion and sets to provide more flexibility in matching more complex patterns. These tools are powerful and I use them many times a day. Yet there are things that cannot be done with wildcards.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_6

Regular expressions (REGEXes or REs) provide us with more complex and flexible pattern matching capabilities. Just as certain characters take on special meaning when using file globbing, REs also have special characters. There are two main types of regular expressions (REs), basic regular expressions (BREs) and extended regular expressions (EREs).

The first thing we need are some definitions. There are many definitions for the term "regular expressions," but many are dry and uninformative. Here are mine:

- **Regular expressions** are strings of literal and metacharacters that can be used as patterns by various Linux utilities to match strings of ASCII plain text data in a data stream. When a match occurs, it can be used to extract or eliminate a line of data from the stream or to modify the matched string in some way.
- **Basic regular expressions (BREs)** and **extended regular expressions (EREs)** are not significantly different in terms of functionality.[1] The primary difference is in the syntax used and how metacharacters are specified. In basic regular expressions, the metacharacters "**?**", "+", "{", "|", "(", and ")" lose their special meaning; instead, it is necessary to use the backslash versions "\?", "\+", "\{", "\|", "\(", and "\)". The ERE syntax is believed by many users to be easier to use.

Regular expressions (REs)[2] take the concept of using metacharacters to match patterns in data streams much further than file globbing and give us even more control over the items we select from a data stream. REs are used by various tools to parse[3] a data stream to match patterns of characters in order to perform some transformation on the data.

[1]See the grep info page in Section 3.6 Basic vs. Extended Regular Expressions.

[2]When I talk about regular expressions, in a general sense I usually mean to include both basic and extended regular expressions. If there is a differentiation to be made, I will use the acronyms BRE for basic regular expression and ERE for extended regular expression.

[3]One general meaning of parse is to examine something by studying its component parts. For our purposes, we parse a data stream to locate sequences of characters that match a specified pattern.

Regular expressions have a reputation for being obscure and arcane incantations that only those with special wizardly SysAdmin powers use. Figure 6-1 would seem to confirm this. The command pipeline appears to be an intractable sequence of meaningless gibberish to anyone without the knowledge of regex. It certainly seemed that way to me the first time I encountered something similar early in my career. As you will see, it is actually relatively simple once it is all explained.

```
cat Experiment_6-1.txt | grep -v Team | grep -v "^\s*$" | sed -e "s/[Ll]eader//" -e "s/\[//g"
-e "s/\]//g" -e "s/)//g" | awk '{print $1" "$2" <"$3">"}' > addresses.txt
```

***Figure 6-1.*** *A real-world sample of the use of regular expressions. It is actually a single line that I used to transform a file that was sent to me into a usable form*

We can only begin to touch upon all of the possibilities opened to us by regular expressions in a single chapter. There are entire books devoted exclusively to regular expressions so we will explore the basics in this chapter – just enough to get started with tasks common to SysAdmins.

# Getting started

Now we need a real-world example to use as a learning tool. Here is one I encountered several years ago.

## The mailing list

This example highlights the power and flexibility of the Linux command line, especially regular expressions, for their ability to automate common tasks. I have administered several listservs during my career and still do. People send me lists of email addresses to add to those lists. In more than one case, I have received a list of names and email addresses in a Word format that were to be added to one of the lists.

The list itself was not really very long, but it was very inconsistent in its formatting. An abbreviated version of that list, with name and domain changes, is shown in Figure 6-2. The original list has extra lines, characters like brackets and parentheses that need to be deleted whitespace such as spaces and tabs, and some empty lines. The format required to add these emails to the list is first last <email@example.com>. Our task is to transform this list into a format usable by the mailing list software.

```
Team 1 Apr 3
Leader  Virginia Jones  vjones88@example.com
Frank Brown  FBrown398@example.com
Cindy Williams  cinwill@example.com
Marge smith   msmith21@example.com
 [Fred Mack]   edd@example.com

Team 2 March 14
leader  Alice Wonder  Wonder1@example.com
John broth  bros34@example.com
Ray Clarkson  Ray.Clarks@example.com
Kim West    kimwest@example.com
[JoAnne Blank]  jblank@example.com

Team 3 Apr 1
Leader  Steve Jones  sjones23876@example.com
Bullwinkle Moose bmoose@example.com
Rocket Squirrel RJSquirrel@example.com
Julie Lisbon  julielisbon234@example.com
[Mary Lastware) mary@example.com
```

***Figure 6-2.*** *A partial, modified listing of the document of email addresses to add to a listserv*

It was obvious that I needed to manipulate the data in order to mangle it into an acceptable format for inputting to the list. It is possible to use a text editor or a word processor such as LibreOffice Writer to make the necessary changes to this small file. However, people send me files like this quite often so it becomes a chore to use a word processor to make these changes. Despite the fact that Writer has a good search and replace function, each character or string must be replaced singly and there is no way to save previous searches. Writer does have a very powerful macro feature, but I am not familiar with either of its two languages, LibreOffice Basic or Python. I do know Bash shell programming.

## The first solution

I did what comes naturally to a SysAdmin – I automated the task. The first thing I did was to copy the address data to a text file so I could work on it using command-line tools. After a few minutes of work, I developed the Bash command-line program in Figure 6-1 that produced the desired output as the file, addresses.txt. I used my normal approach to writing command-line programs like this by building up the pipeline one command at a time.

Let's break this pipeline down into its component parts to see how it works and fits together. All of the experiments in this chapter are to be performed as the student user.

### EXPERIMENT 6-1

First we download the sample file Experiment_6-1.txt from the Apress GitHub web site. Let's do all of this work in a new directory so we will create that too.

```
[student@studentvm1 ~]$ mkdir chapter6 ; cd chapter6
[student@studentvm1 chapter6]$ wget https://raw.githubusercontent.com/Apress/
using-and-administering-linux-volume-2/master/Experiment_6-1.txt
```

Now we just take a look at the file and see what we need to do.

```
[student@studentvm1 chapter6]$ cat Experiment_6-1.txt
Team 1  Apr 3
Leader  Virginia Jones  vjones88@example.com
Frank Brown  FBrown398@example.com
Cindy Williams  cinwill@example.com
Marge smith   msmith21@example.com
 [Fred Mack]   edd@example.com

Team 2  March 14
leader  Alice Wonder  Wonder1@example.com
John broth  bros34@example.com
Ray Clarkson  Ray.Clarks@example.com
Kim West    kimwest@example.com
[JoAnne Blank]  jblank@example.com
```

```
Team 3  Apr 1
Leader  Steve Jones  sjones23876@example.com
Bullwinkle Moose bmoose@example.com
Rocket Squirrel RJSquirrel@example.com
Julie Lisbon  julielisbon234@example.com
[Mary Lastware) mary@example.com

[student@studentvm1 chapter6]$
```

The first things I see that can be done are a couple easy ones. Since the Team names and dates are on lines by themselves, we can use the following to remove those lines that have the word "Team." I place the end of sentence period outside the quotes for clarity to ensure that only the intended string is inside the quotes.

```
[student@studentvm1 chapter6]$ cat Experiment_6-1.txt | grep -v Team
```

I won't reproduce the results of each stage of building this Bash program, but you should be able to see the changes in the data stream as it shows up on STDOUT, the terminal session. We won't save it in a file until the end.

In this first step in transforming the data stream into one that is usable, we use the grep command with a simple literal pattern, "Team." Literals are the most basic type of pattern we can use as a regular expression because there is only a single possible match in the data stream being searched, and that is the string "Team".

We need to discard empty lines so we can use another grep statement to eliminate them. I find that enclosing the regular expression for the second grep command ensures that it gets interpreted properly.

```
[student@studentvm1 chapter6]$ cat Experiment_6-1.txt | grep -v Team | grep
-v "^\s*$"
Leader  Virginia Jones  vjones88@example.com
Frank Brown  FBrown398@example.com
Cindy Williams  cinwill@example.com
Marge smith   msmith21@example.com
 [Fred Mack]   edd@example.com
leader  Alice Wonder  Wonder1@example.com
John broth  bros34@example.com
Ray Clarkson  Ray.Clarks@example.com
Kim West    kimwest@example.com
[JoAnne Blank]  jblank@example.com
```

```
Leader  Steve Jones  sjones23876@example.com
Bullwinkle Moose bmoose@example.com
Rocket Squirrel RJSquirrel@example.com
Julie Lisbon  julielisbon234@example.com
[Mary Lastware) mary@example.com
[student@studentvm1 chapter6]$
```

The expression **"^\s*$"** illustrates anchors and using the backslash (\) as an escape character to change the meaning of a literal, "s" in this case, to a metacharacter that means any whitespace such as spaces, tabs, or other characters that are unprintable. We cannot see these characters in the file, but it does contain some of them. The asterisk, a.k.a. splat (*), specifies that we are to match zero or more of the whitespace characters. This would match multiple tabs or multiple spaces or any combination of those in an otherwise empty line.

I configured my Vim editor to display whitespace using visible characters. Do this by adding the following line to your own ~.vimrc or the global /etc/vimrc files. Then start – or restart – Vim.

```
set listchars=eol:$,nbsp:_,tab:<->,trail:~,extends:>,space:+
```

I found a lot of bad, very incomplete, and contradictory information on the Internet in my searches on how to do this. The built-in Vim help has the best information, and the data line I have created from that here is one that works for me.

The result, before any operation on the file, is shown in Figure 6-3. Regular spaces are shown as +; tabs are shown as <, <>, or <-->, and fill the length of the space that the tab covers. The end of line (EOL) character is shown as $.

```
Team+1<>Apr+3~$
Leader++Virginia+Jones++vjones88@example.com<-->$
Frank+Brown++FBrown398@example.com<---->$
Cindy+Williams++cinwill@example.com<--->$
Marge+smith+++msmith21@example.com~$
+[Fred+Mack]+++edd@example.com<>$
$
Team+2<>March+14$
leader++Alice+Wonder++Wonder1@example.com<----->$
John+broth++bros34@example.com<>$
Ray+Clarkson++Ray.Clarks@example.com<-->$
Kim+West++++kimwest@example.com>$
[JoAnne+Blank]++jblank@example.com<---->$
$
Team+3<>Apr+1~$
Leader++Steve+Jones++sjones23876@example.com<-->$
Bullwinkle+Moose+bmoose@example.com<--->$
Rocket+Squirrel+RJSquirrel@example.com<>$
Julie+Lisbon++julielisbon234@example.com<------>$
[Mary+Lastware)+mary@example.com$
```

***Figure 6-3.*** *The Experiment_6-1.txt file showing all of the embedded whitespace*

You can see that there are a lot of whitespace characters that need to be removed from our file. We also need to get rid of the work "leader" which appears twice and is capitalized once. Let's get rid of "leader" first. This time we will use **sed** (stream editor) to perform this task by substituting a new string – or a null string in our case – for the pattern it matches. Adding **sed -e "s/[Ll]eader//"** to the pipeline does this.

```
[student@studentvm1 chapter6]$ cat Experiment_6-1.txt | grep -v Team | grep
-v "^\s*$" | sed -e "s/[Ll]eader//"
```

In this **sed** command, -e means that the quote enclosed expression is a script that produces a desired result. In the expression the s means that this is a substitution. The basic form of a substitution is **s/regex/replacement string/**. So **/[Ll]eader/** is our search string. The set **[Ll]** matches L or l so **[Ll]eader** matches leader or Leader. In this case the replacement string is null because it looks like this - // - a double forward slash with no characters or whitespace between the two slashes.

Now let's get rid of some of the extraneous characters like []() that will not be needed.

```
[student@studentvm1 chapter6]$ cat Experiment_6-1.txt | grep -v Team | grep
-v "^\s*$" | sed -e "s/[Ll]eader//" -e "s/\[//g" -e "s/]//g" -e "s/)//g" -e
"s/(//g"
```

We have added four new expressions to the **sed** statement. Each one removes a single character. The first of these additional expressions is a bit different. Because the left square brace [ character can mark the beginning of a set, we need to escape it to ensure that **sed** interprets it correctly as a regular character and not a special one.

We could use **sed** to remove the leading spaces from some of the lines, but the **awk** command can do that as well as reorder the fields if necessary, and add the <> characters around the email address.

```
[student@studentvm1 chapter6]$ cat Experiment_6-1.txt | grep -v Team | grep
-v "^\s*$" | sed -e "s/[Ll]eader//" -e "s/\[//g" -e "s/]//g" -e "s/)//g" -e
"s/(//g" | awk '{print $1" "$2" <"$3">"}'
```

The **awk** utility is actually a very powerful programming language that can accept data streams on its STDIN. This makes it extremely useful in command-line programs and scripts.

The **awk** utility works on data fields and the default field separator is spaces – any amount of whitespace. The data stream we have created so far has three fields separated by whitespace, first, last, and email. This little program **awk '{print $1" "$2" <"$3">"}'** takes each of the three fields, $1, $2, and $3, and extracts them without leading or trailing whitespace. It then prints them in sequence adding a single space between each as well as the <> characters needed to enclose the email address.

The last step here would be to redirect the output data stream to a file, but that is trivial so I leave it with you to perform that step. It is not really necessary that you do so.

---

I saved the Bash program in an executable file and now I can run this program any time I receive a new list. Some of those lists are fairly short, as is the one in Figure 6-3, but others have been quite long, sometimes containing up to several hundred addresses and many lines of "stuff" that do not contain addresses to be added to the list.

## The second solution

But now that we have a working solution, one that is a step-by-step exploration of the tools we are using, we can do quite a bit more to perform the same task in a more compact and optimized command-line program.

### EXPERIMENT 6-2

In this experiment we explore ways in which we can shorten and simplify the command-line program from Experiment 6-1. The final result of that experiment was the following CLI program.

```
cat Experiment_6-1.txt | grep -v Team | grep -v "^\s*$" | sed -e "s/[Ll]
eader//" -e "s/\[//g" -e "s/]//g" -e "s/)//g" -e "s/(//g" | awk '{print $1"
"$2" <"$3">"}'
```

Let's start near the beginning and combine the two **grep** statements. The result is shorter and more succinct. It also means faster execution because grep only needs to parse the data stream once.

---

**Tip** When the STDOUT from grep is not piped through another utility and when using a terminal emulator that supports color, the regex matches are highlighted in the output data stream. The default for the Xfce4-terminal is a black background, white text, and highlighted text in red.

---

In the revised **grep** command, **grep -vE "Team|^\s*$"**, we add the E option which specifies extended regex. According to the **grep** man page, "In GNU grep there is no difference in available functionality between basic and extended syntaxes." This statement is not strictly true because our new combined expression fails without the E option. Run the following to see the results.

```
[student@studentvm1 chapter6]$ cat Experiment_6-1.txt | grep -vE "Team|^\s*$"
```

Try it without the E option.

The **grep** tool can also read data from a file so we eliminate the **cat** command.

```
[student@studentvm1 chapter6]$ grep -vE "Team|^\s*$" Experiment_6-1.txt
```

This leaves us with the following, somewhat simplified CLI program.

```
grep -vE "Team|^\s*$" Experiment_6-1.txt | sed -e "s/[Ll]eader//" -e "s/\
[//g" -e "s/]//g" -e "s/)//g" -e "s/(//g" | awk '{print $1" "$2" <"$3">"}'
```

We can also simplify the sed command, and we will do so in Experiment 6-6 after we learn more about regular expressions.

It is important to realize that my solution is not the only one. There are different methods in Bash for producing the same output; there are other languages like Python and Perl that can also be used. And, of course, there are always LibreOffice Writer macros. But I can always count on Bash as part of any Linux distribution. I can perform these tasks using Bash programs on any Linux computer, even one without a GUI desktop or that does not have LibreOffice installed.

# grep

Because GNU **grep** is one of the tools I use the most that provides a more or less standardized implementation of regular expressions, I will use that set of expressions as the basis for the next part of this chapter. We will then look at **sed**, another tool that uses regular expressions.

Throughout this self-study course, you will have already encountered globs and regexes. Along with the previous experiments in this chapter, you should have at least a basic understanding of regexes and how they work. However, there are many details that are important to understanding some of the complexity and of regex implementations and how they work.

## Data flow

All implementations of regular expressions are line based. A pattern created by a combination of one or more expressions is compared against each line of a data stream. When a match is made, an action is taken on that line as prescribed by the tool being used. For example, when a pattern match occurs with **grep**, the usual action is to pass that line on to STDOUT and lines that do not match the pattern are discarded. As we have seen, the -v option reverses those actions so that the lines with matches are discarded.

Each line of the data stream is evaluated on its own, and the results of matching the expressions in the pattern with the data from previous lines are not carried over. It might be helpful to think of each line of a data stream as a record and that the tools that use regexes process one record at a time. When a match is made, an action defined by the tool in use is take on the line that contains the matching string.

## regex building blocks

Figure 6-4 contains a list of the basic building block expressions and metacharacters implemented by the GNU **grep** command and their descriptions. When used in a pattern, each of these expressions or metacharacters matches a single character in the data stream being parsed.

| Expression | Description |
|---|---|
| Alphanumeric characters<br>Literals<br>A-Z,a-z,0-9 | All alphanumeric and some punctuation characters are considered as literals. Thus the letter "a" in a regex will always match the letter "a" in the data stream being parsed. There is no ambiguity for these characters. Each literal character matches one and only one character. |
| . (dot) | The dot (.) metacharacter is the most basic form of expression. It matches any single character in the position it is encountered in a pattern. So the pattern b.g would match big, bigger, bag, baguette, and bog, but not dog, blog, hug, lag, gag, or leg, etc. |
| Bracket expression<br>[list of characters] | GNU grep calls this a bracket expression and it is the same as a set for the Bash shell. The brackets enclose a list of characters to match for a single character location in the pattern. [abcdABCD] matches the letters a, b, c, or d in either upper or lower case. [a-dA-D] specifies a range of characters that creates the same match. [a-zA-Z] matches the alphabet in upper and lower case. |
| [:class name:]<br>Character classes | This is a POSIX* attempt at regex standardization. The class names are supposed to be obvious. For example the [:alnum:] class matches all alphanumeric characters. Other classes are [:digit:] which matches any one digit 0-9, [:alpha:], [:space:], and so on. Note that there may be issues due to differences in the sorting sequences in different locales. Read the grep man page for details. |
| ^ and $<br>Anchors | These two metacharacters match the beginning and ending of a line, respectively. They are said to anchor the rest of the pattern to either the beginning or ending of a line. The expression ^b.g would only match big, bigger, bag, etc., as shown above, if they occur at the beginning of the line being parsed. The pattern b.g$ would match big or bag only if they occur at the end of the line, but not bigger. |

*Wikipedia, POSIX, https://en.wikipedia.org/wiki/POSIX

***Figure 6-4.*** *These expressions and metacharacters are implemented by grep and most other regex implementations*

Let's explore these building blocks before continuing on with some of the modifiers. The text file we will use for Experiment 6-3 is from a lab project I created for an old Linux class I used to teach. It was originally in a LibreOffice Writer ODT file, but I saved it to an ASCII text file. Most of the formatting of things like tables was removed, but the result is a long ASCII text file that we can use for this series of experiments.

## EXPERIMENT 6-3

We must download the sample file from the Apress GitHub web site. If the directory ~/chapter6 is not the PWD, make it so. This is a document containing lab projects from which I used to teach.

```
[student@studentvm1 chapter6]$ wget https://raw.githubusercontent.com/Apress/
using-and-administering-linux-volume-2/master/Experiment_6-3.txt
```

To begin, just use the less command to look at and explore the Experiment_6-3.txt file for a few minutes so you have an idea of its content.

Now we will use some simple expressions in grep to extract lines from the input data stream. The Table of Contents (TOC) contains a list of projects and their respective page numbers in the PDF document. Let's extract the TOC starting with lines ending in two digits.

```
[student@studentvm1 chapter6]$ grep [0-9][0-9]$ Experiment_6-3.txt
```

That is not really what we want. It displays all lines that end in two digits and misses TOC entries with only one digit. We will look at how to deal with an expression for one or more digits in a later experiment. Looking at the whole file in less, we could do something like this.

```
[student@studentvm1 chapter6]$ grep "^Lab Project" Experiment_6-3.txt | grep
"[0-9]$"
```

This is much closer to what we want but it is not quite there. We get some lines from later in the document that also match these expressions. If you study the extra lines and look at those in the complete document, you can see why they match while not being part of the TOC. This also misses TOC entries that do not start with "Lab Project." Sometimes this is the best you can do, but it does give a better look at the TOC than we had before. We will look at how to combine these two grep instances into a single one in a later experiment in this chapter.

Now let's modify this a bit and use the POSIX expression. Notice the double square braces around the POSIX expression. Single braces generate an error message.

```
[student@studentvm1 chapter6]$ grep "^Lab Project" Experiment_6-3.txt | grep
"[[:digit:]]$"
```

This gives the same results as the previous attempt. Let's look for something different.

```
[student@studentvm1 chapter6]$ grep systemd Experiment_6-3.txt
```

This lists all occurrences of "systemd" in the file. Try using the -i option to ensure that you get all instances including those that start with uppercase.[4] Or you could just change the literal expression to Systemd. Count the number of lines with the string systemd contained in them. I always use -i to ensure that all instances of the search expression are found regardless of case.

```
[student@studentvm1 chapter6]$ grep -i systemd Experiment_6-3.txt | wc
     20     478    3098
```

As you can see I have 20 lines and you should have the same number.

Here is an example of matching a metacharacter. the left bracket ([). First let's try it without doing anything special.

```
[student@studentvm1 chapter6]$ grep -i "[" Experiment_6-3.txt
grep: Invalid regular expression
```

This occurs because [ is interpreted as a metacharacter. We need to "escape" this character with a backslash so that it is interpreted as literal character and not as a metacharacter.

```
[student@studentvm1 chapter6]$ grep -i "\[" Experiment_6-3.txt
```

Most metacharacters lose their special meaning when used inside bracket expressions. To include a literal ], place it first in the list. To include a literal ^, place it anywhere but first. To include a literal [, place it last.

[4]The official form of systemd is all lowercase.

## Repetition

Regular expressions may be modified using some operators that allow specification of zero, one, or more repetitions of a character or expression. These repetition operators, shown in Figure 6-5, are placed immediately following the literal character or metacharacter used in the pattern.

| Operator | Description |
|---|---|
| ? | In regexes the ? means zero or one occurrence at most of the preceding character. So for example, "drives?" matches drive, and drives but not driver. Using “drive” for the expression would match drive, drives, and driver. This is a bit different from the behavior of ? in a glob. |
| * | The character preceding the * will be matched zero or more times without limit. In this example, "drives*" matches drive, drives, and drivesss but not driver. Again this is a bit different from the behavior of * in a glob. |
| + | The character preceding the + will be matched one or more times. The character must exist in the line at least once for a match to occur. As one example, "drives+" matches drives, and drivesss but not drive or driver. |
| {n} | This operator matches the preceding character exactly n times. The expression “drives{2}” matches drivess but not drive, drives, drivesss, or any number of trailing “s” characters. However, because drivesssss contains the string drivess, a match occurs on that string so the line would be a match by grep. |
| {n,} | This operator matches the preceding character n or more times. The expression “drives{2,}” matches drivess but not drive, drives, drivess, drives, or any number of trailing “s” characters. Because drivesssss contains the string drivess, a match occurs. |
| {,m} | This operator matches the preceding character no more than m times. The expression “drives{,2}” matches drive, drives, and drivess, but not drivesss, or any number of trailing “s” characters. Once again, because drivesssss contains the string drivess, a match occurs. |
| {n,m} | This operator matches the preceding character at least n times but no more than m times. The expression “drives{1,3}” matches drives, drivess, and drivesss, but not drivessss or any number of trailing “s” characters. Once again, because drivesssss contains a matching string, a match occurs. |

***Figure 6-5.*** *Metacharacter modifiers that specify repetition*

**EXPERIMENT 6-4**

Run each of the following commands and examine the results carefully so that you understand what is happening.

```
[student@studentvm1 chapter6]$ grep -E files? Experiment_6-3.txt
[student@studentvm1 chapter6]$ grep -Ei "drives*" Experiment_6-3.txt
[student@studentvm1 chapter6]$ grep -Ei "drives+" Experiment_6-3.txt
[student@studentvm1 chapter6]$ grep -Ei "drives{2}" Experiment_6-3.txt
[student@studentvm1 chapter6]$ grep -Ei "drives{2,}" Experiment_6-3.txt
[student@studentvm1 chapter6]$ grep -Ei "drives{,2}" Experiment_6-3.txt
[student@studentvm1 chapter6]$ grep -Ei "drives{2,3}" Experiment_6-3.txt
```

Be sure to experiment with these modifiers on other text in the sample file.

## Other metacharacters

There are still some interesting and important modifiers that we need to explore. These metacharacters are listed and described in Figure 6-6.

| Modifier | Description |
|---|---|
| \< | This special expression matches the empty string at the beginning of a word. The expression "\<fun" would match on " fun" and "Function" but not "refund". |
| \> | This special expression matches the normal space, or empty " " string at the end of a word as well as punctuation that typically appears in the single character string at the end of a word. So "environment\>" matches "environment", "environment,", and environment." but not environments or environmental. |
| ^ | In a character class expression, this operator negates the list of characters. Thus, while the class [a-c] matches a, b , or c, in that position of the pattern, the class [^a-c] matches anything but a, b, or c. |
| \| | When used in a regex, the \| metacharacter is a logical "or" operator. It is officially called the "infix" or "alternation" operator. We have already encountered this in Experiment 6-2, where we saw that the regex "Team\|^\s*$" means, "a line with 'Team' or ( \| ) an empty line including one that has zero, one, or more whitespace characters such as spaces, tabs, and other unprintable characters." |
| ( and ) | The parentheses ( and ) allow us to ensure a specific sea<br>might be used for logical comparisons in a programming language. |

***Figure 6-6.*** *Metacharacter modifiers*

We now have a way to specify word boundaries with the \< and \> metacharacters. This means we can now be even more explicit with our patterns. We can also use some logic in more complex patterns.

## EXPERIMENT 6-5

Start with a couple simple patterns. This first one selects all instances of drives but not drive, drivess, or additional trailing "s" characters.

```
[student@studentvm1 chapter6]$ grep -Ei "\<drives\>" Experiment_6-3.txt
```

Now let's build up a search pattern to locate references to tar, the tape archive command, and related references. The first two iterations display more than just tar-related lines.

```
[student@studentvm1 chapter6]$ grep -Ei "tar" Experiment_6-3.txt
[student@studentvm1 chapter6]$ grep -Ei "\<tar" Experiment_6-3.txt
[student@studentvm1 chapter6]$ grep -Ein "\<tar\>" Experiment_6-3.txt
```

The -n option in the last command displays the line numbers of each line in which a match occurred. This can assist in locating specific instances of the search pattern.

---

**Tip** Matching lines of data can extend beyond a single screen, especially when searching a large file. You can pipe the resulting data stream through the less utility and then use the less search facility which implements regexes too to highlight the occurrences of matches to the search pattern. The search argument in less is **\<tar\>**.

---

This next pattern searches for "shell script" or "shell program" or "shell variable" or "shell environment" or "shell prompt" in our test document. The parentheses alter the logical order in which the pattern comparisons are resolved.

```
[student@studentvm1 chapter6]$ grep -Eni "\<shell (script|program|variable|en
vironment|prompt)" Experiment_6-3.txt
```

Remove the parentheses from the preceding command and run it again to see the difference.

Although we have now explored the basic building blocks of regular expressions in **grep**, there are an infinite variety of ways in which they can be combined to create complex yet elegant search patterns. However, **grep** is a search tool and does not provide any direct capability to edit or modify the contents of a line of text in the data stream when a match is made.

# sed

The **sed** utility not only allows searching for text that matches a regex pattern, it can also modify, delete, or replace the matched text. I use **sed** at the command line and in Bash shell scripts as a fast and easy way to locate and text and alter it in some way. The name **sed** stands for stream editor because it operates on data streams in the same manner as other tools that can transform a data stream. Most of those changes simply involve selecting specific lines from the data stream and passing them on to another transformer[5] program.

We have already seen **sed** in action, but now, with an understanding of regular expressions, we can better analyze and understand our earlier usage.

## EXPERIMENT 6-6

In Experiment 6-2 we simplified the CLI program we used to transform a list of names and email addresses into a form that can be used as input to a listserv. That CLI program looks like this after some simplification.

```
grep -vE "Team|^\s*$" Experiment_6-1.txt | sed -e "s/[Ll]eader//" -e "s/\
[//g" -e "s/]//g" -e "s/)//g" -e "s/(//g" | awk '{print $1" "$2" <"$3">"}'
```

It is possible to combine four of the five expressions used in the **sed** command into a single expression. The **sed** command now has two expressions instead of five.

```
sed -e "s/[Ll]eader//" -e "s/[]()\[]//g"
```

[5]Many people call tools like grep "filter" programs because they filter unwanted lines out of the data stream. I prefer the term "transformers" because ones such as sed and awk do more than just filter. They can test the content for various string combinations and alter the matching content in many different ways. Tools like sort, head, tail, uniq, fmt, and more all transform the data stream in some way.

This makes it a bit difficult to understand the more complex expression. Note that no matter how many expressions a single **sed** command contains, the data stream is only parsed once to match all of the expressions.

Let's examine the revised expression, `-e "s/[]()\[]//g",` more closely. By default, **sed** interprets all [ characters as the beginning of a set and the last ] character as the end of that set, `-e "s/[]()\[]//g"`. The intervening ] characters are not interpreted as metacharacters. Since we need to match [ as a literal character in order to remove it from the data stream and **sed** normally interprets that as a metacharacter, we need to escape it so that it is interpreted as a literal ], `-e "s/[]()\[]//g"`. So now all of the metacharacters in this expression are highlighted. Let's plug this into the CLI script and test it.

```
[student@studentvm1 chapter6]$ grep -vE "Team|^\s*$" Experiment_6-1.txt |
sed -e "s/[Ll]eader//" -e "s/[]()\[]//g"
```

I know that you are asking "Why not place the \[ after the [ that opens the set and before the ] character." Try it as I did.

```
[student@studentvm1 chapter6]$ grep -vE "Team|^\s*$" Experiment_6-1.txt |
sed -e "s/[Ll]eader//" -e "s/[\[]()]//g"
```

I think that should work but it does not. Little unexpected results like this make it clear that we must be careful and test each regex carefully to ensure that it actually does what we intend. After some experimentation of my own, I discovered that the escaped left square brace \[ works fine in all positions of the expression except for the first one. This behavior is noted in the **grep** man page which I probably should have read first. However, I find that experimentation reinforces the things I read and I usually discover more interesting things than that for which I was looking.

Adding the last component, the awk statement, our optimized program looks like this and the results are exactly what we want.

```
[student@studentvm1 chapter6]$ grep -vE "Team|^\s*$" Experiment_6-1.txt |
sed -e "s/[Ll]eader//" -e "s/[]()\[]//g" | awk '{print $1" "$2" <"$3">"}'
```

## Other tools that implement regular expressions

Many Linux tools implement regular expressions. Most of those implementations are very similar to that of **awk**, **grep**, and **sed** so that it should be easy to learn the differences. Although we have not looked in detail at awk, it is a powerful text processing language that also implements regexes.

Most of the more advanced text editors use regexes. Vim, gVim, Kate, and GNU Emacs are no exceptions. The **less** utility implements regexes as does the search and replace facility of LibreOffice Writer.

Programming languages like Perl, awk, and Python also contain implementations of regexes which makes them well suited to writing tools for text manipulation.

## Resources

I have found some excellent resources for learning about regular expressions. There are more than I have listed here, but these are the ones I have found to be particularly useful.

The grep man page has a good reference but is not appropriate for learning about regular expressions. The O'Reilly book, *Mastering Regular Expressions,*[6] is a very good tutorial and reference for regular expressions. I recommend it for anyone who is or wants to be a Linux SysAdmin because you will use regular expressions. Another good O'Reilly book is *sed & awk*[7] which covers both of these powerful tools, and it also has an excellent discussion of regular expressions.

There are also some good web sites that can help you learn about regular expressions and which provide interesting and useful cookbook-style regex examples. There are some that ask for money in return for using them. Jason Baker, my technical reviewer for Volumes 1 and 2 of this course, suggests `https://regexcrossword.com/` as a good learning tool.

[6]Friedl, Jeffrey E. F., *Mastering Regular Expressions*, O'Reilly, 2012, Paperback ISBN-13: 978-0596528126

[7]Robbins, Arnold, and Dougherty, Dale, *sed & awk: UNIX Power Tools (Nutshell Handbooks)*, O'Reilly, 2012, ISBN-13: 978-1565922259

# Chapter summary

This chapter has provided a very brief introduction to the complex world of regular expressions. We have explored the regex implementation in the grep utility in just enough depth to give you an idea of some of the amazing things that can be accomplished with regexes. We have also looked at several Linux tools and programming languages that also implement regexes.

But make no mistake! We have only scratched the surface of these tools and regular expressions. There is much more to learn and there are some excellent resources for doing so.

# Exercises

Perform these exercises to complete this chapter:

1. In Experiment 6-1 we included a **sed** search for the ( character even though there was not one in the Experiment_6-1.txt data file. Why do you think that might be a good idea?
2. Consider the following problem regarding Experiments 6-1 and 6-2. What would happen to the resulting data stream if one or more lines had a different data format such as first, middle, last, or if it were last, first?
3. The following regex is used in Experiment 6-5: **`grep -Eni "\<shell (script|program|variable|environment|prompt)" Experiment_6-3.txt`**. Create a statement of the logic defined by this regex. Then create a statement of the logic of this regex with the parentheses removed.
4. The grep utility has an option that can be used to specify that only words are to be matched so that the \< and \> metacharacters are not required. In Experiment 6-6, eliminate the word metacharacters using that option and test the result.
5. Use the sed command to replace the grep command in the last iteration of the CLI program in Experiment 6-6: **`grep -vE "Team|^\s*$" Experiment_6-1.txt | sed -e "s/[Ll]eader//" -e "s/[]()\[]//g" | awk '{print $1" "$2" <"$3">"}'`**.

CHAPTER 7

# Printing

## Objectives

In this chapter you will learn

- How to install a printer and make it available to the VM
- To describe the flow of a print data stream
- To determine how well a printer is supported by CUPS under Linux
- To select an appropriately well-supported printer for Linux
- To select an appropriate PPD file for a printer when an exact match is not available
- To configure a print queue from the command line using CUPS
- To manage print queues; to enable and disable them and to move print jobs from one queue to another
- To create print queue that converts printer data streams to PDF format for storage as a file
- To convert ASCII plain text files to Postscript and PDF formats
- To convert Postscript and PDF files to ASCII text format
- To convert data files between Linux/Unix formats, DOS/Windows formats, and Apple formats




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_7

# Introduction

We have already explored many Linux tools that enable us to do some pretty amazing things. In this chapter we will look at some additional command-line tools that are all designed to manipulate text files and data streams in order to prepare them for printing. Some of these tools can convert data streams from ASCII text to PDF, Postscript, and back; some can convert MS Word and LibreOffice Writer documents to ASCII; other tools can convert from Apple or DOS text files to Linux text files.

We also look at command-line tools that enable us to create and manage print queues.

# About printers

Printers are hardware devices that are used to produce images or text on sheets of paper. Well, duh. But it is important to understand that printers build up an image of the page to be printed one line at a time. 3D printers can print objects one line at a time, but we will stick with printing words and images on paper for this course.

If you have an ink jet printer, you can watch the process as it takes place. The print head travels horizontally across the paper laying down the image of the text or graphic one line at a time. To be very clear, I don't necessarily mean one line of text at a time. I mean that one line of the print image that is as tall as the vertical size of the print head. This may encompass a single line of text, but it also may be the bottom of one line of text and the top of the next line of text or a combination of text and image.

If you try to equate the operation of a modern printer with that of an old dot matrix printer that printed one line of text at a time, you will not understand today's printers. At least some understanding of how a printer works is important to understanding how Linux prints and the tools used to print documents to paper as well as tools that create images that can be both printed to paper and viewed on a graphical display.

Even when printing text, modern printers, whether ink jet or laser, all print page images of the page or pages being printed. The ink jet printer builds up the image one line at a time directly onto the paper, while the laser printer generates the entire image on a drum inside the printer and then transfers it to the paper an entire page at a time.

## Print languages

To create the images to be printed, applications such as office suites, web browsers, financial software, and every other bit of software that can print to a printer must generate a data stream that can be interpreted and converted to an image for printing. These data streams are created using one of the common page description languages (PDL).

The function of these PDLs is to describe the appearance of the page when it is printed. They use commands like draw a box at this location with height and width and background color and center some specified text in it using a specific font face in a specific size and so on. Page description languages are designed to be independent of the hardware, application software, and operating systems on which they are used. This helps standardize the processes and tools used for printing. I won't afflict you with a historical discussion of the vast number of printer drivers that used to be required for each and every application program.

There are many page description languages, at least some of which are listed on Wikipedia,[1] but there are only three that are in common use by most of today's printers:

- **PCL:**[2] Hewlett-Packard's Printer Command Language.
- **Postscript:**[3] Adobe Systems first page description language.
- **PDF**:[4] Adobe Systems Portable Document Format. PDF has become a very common format for exchanging documents.

## Printers and Linux

If you have access to a physical printer from your virtual machine, you will be able to perform the experiments in this chapter, most of which relate to printing the files or preparing files for printing. I personally use Brother, HP, and Xerox printers because I have always found them to be well supported by Linux. I recommend reading "Choosing

[1]Wikipedia, *Page Description Language*, https://en.wikipedia.org/wiki/Page_description_language

[2]Wikipedia, *PCL*, https://en.wikipedia.org/wiki/Printer_Command_Language

[3]Wikipedia, *Postscript*, https://en.wikipedia.org/wiki/PostScript. This page also contains some of the history that explains how the current print architecture came to be.

[4]Wikipedia, *PDF*, https://en.wikipedia.org/wiki/PDF

a printer for Linux,[5]" and then you can check the Open Printing Database[6] at the Linux Foundation document wiki. The Open Printing Database lists printers by manufacturer in four categories that represent their level of compatibility with Linux:

- **Perfectly**: All printer functions work as expected.
- **Mostly**: Some printer features may not work as expected. For example, dual-sided printing or secondary tray paper selection may not work.
- **Partially**: It probably prints some documents, but others will not print correctly and many features don't work as expected.
- **Paperweight**: Not good for anything except being used to hold down documents printed by printers that do work with Linux.

Clearly, printers from one of the first two groups are to be preferred over those in the last two groups. If you need to purchase or recommend a printer for purchase that is compatible with Linux, the Open Printing Database is the place to start looking.

Just as a point of reference, Jason, my intrepid technical reviewer, tested this chapter with an HP Color LaserJet CP2025dn. It's about 10 years old, and he had to dig it out from a pile because it almost never gets used. It is marked as "Perfect" in the Open Printing Database.

Most printers are recognized automatically when you plug them into a USB port[7] on a Linux host, but they are not necessarily automatically configured. If possible, locate a printer that is compatible with Linux by searching the Open Printing Database. Plug that printer into a USB port on the physical host.

If you cannot locate a supported printer, you should skip Experiment 7-1, the second part of Experiment 7-2, and parts of some of the other experiments. Nevertheless, you should at least follow along and read these experiments to better understand the later experiments. Many of the later experiments in this chapter are still accessible to you because it is possible to print to a file, which we will do much of the time anyway, in order to save some trees.

---

[5]Watkins, Don, Opensource.com, *Choosing a printer for Linux*, https://opensource.com/article/18/11/choosing-printer-linux

[6]The Linux Foundation, Open Printing Database, https://www.openprinting.org/printers

[7]The role of D-Bus and udev in recognizing devices like printers when they are plugged in will be covered in Chapter 14 of this volume.

## EXPERIMENT 7-1

**Caution** If you do not have a physical printer, you should read this experiment but there is no part of it that you can perform.

Let's start by connecting a printer to the physical host and then making it available to your VM.

If the printer is not already set up, do so and plug it into a power source. Using a USB cable, plug the compatible printer into a USB port on the physical host. It is not necessary to configure the printer on the physical host.

In the window for StudentVM1, click **Devices ➤ USB** to view all of the USB devices attached to your physical host as shown in Figure 7-1. Click the USB printer. Your printer will probably be different but it should be listed. This is all that needs to be done to make the printer available to the VM.

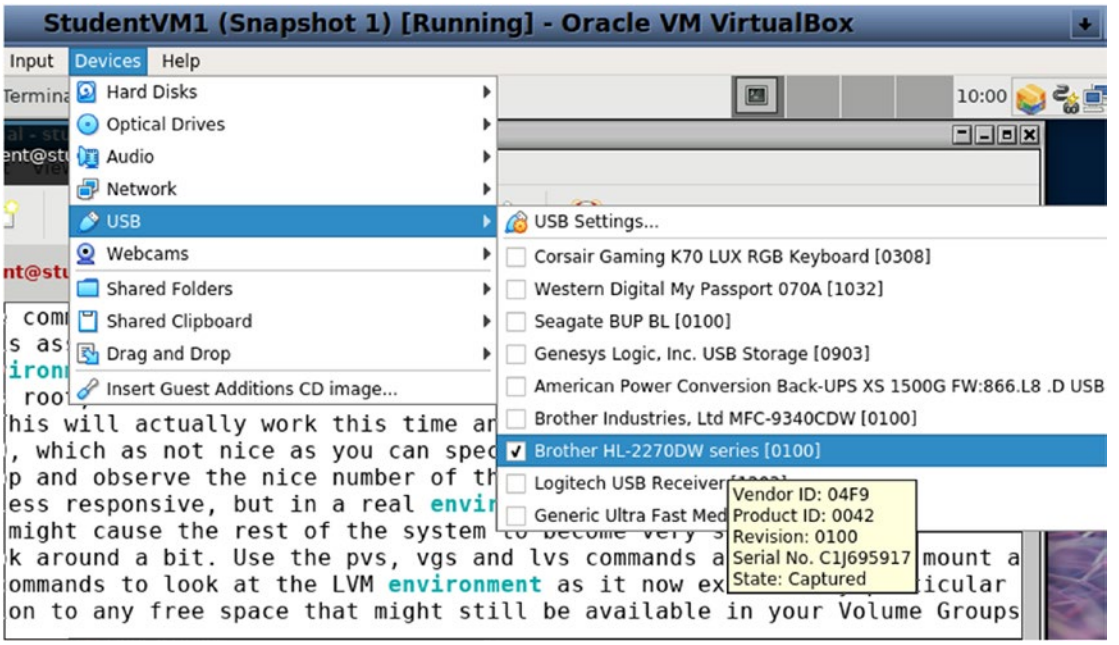


***Figure 7-1.*** *In the USB Settings menu, select the printer connected to the physical host to make it available to the virtual Applehine*

If by some chance your desktop GUI sees the printer and tries to configure it, ignore that, quit the configuration, and proceed with the manual configuration when you get to Experiment 7-2.

# CUPS

CUPS is the Common Unix Printing System.[8] Developed in the late 1990s by Michael Sweet, who was later hired by Apple, CUPS is a modular printing system that makes configuring and printing with most modern printers easy and nearly painless.

CUPS uses Postscript as the final printer data stream. Figure 7-2 illustrates the flow of data from the application programs through the layers of the CUPS subsystem. CUPS accepts input in ASCII text, PDF, HP/GL, and raster image formats and runs them through a filter that transforms the data stream into Postscript. If an incoming data stream is Postscript, no change is required. The data stream, now in Postscript form, then passes to a software layer that converts the data to a rasterized format that can be converted by various drivers to printer-specific language formats. The data stream is then passed to the back-end filters that transfer the printer commands to the printers or other printing devices.

If the input data stream is already in Postscript format, the initial conversion to Postscript is skipped. If the target printer uses Postscript as its print language, the data stream bypasses the rasterization stage and is sent to the back-end filters.

[8]WikiPedia, *CUPS*, https://en.wikipedia.org/wiki/CUPS

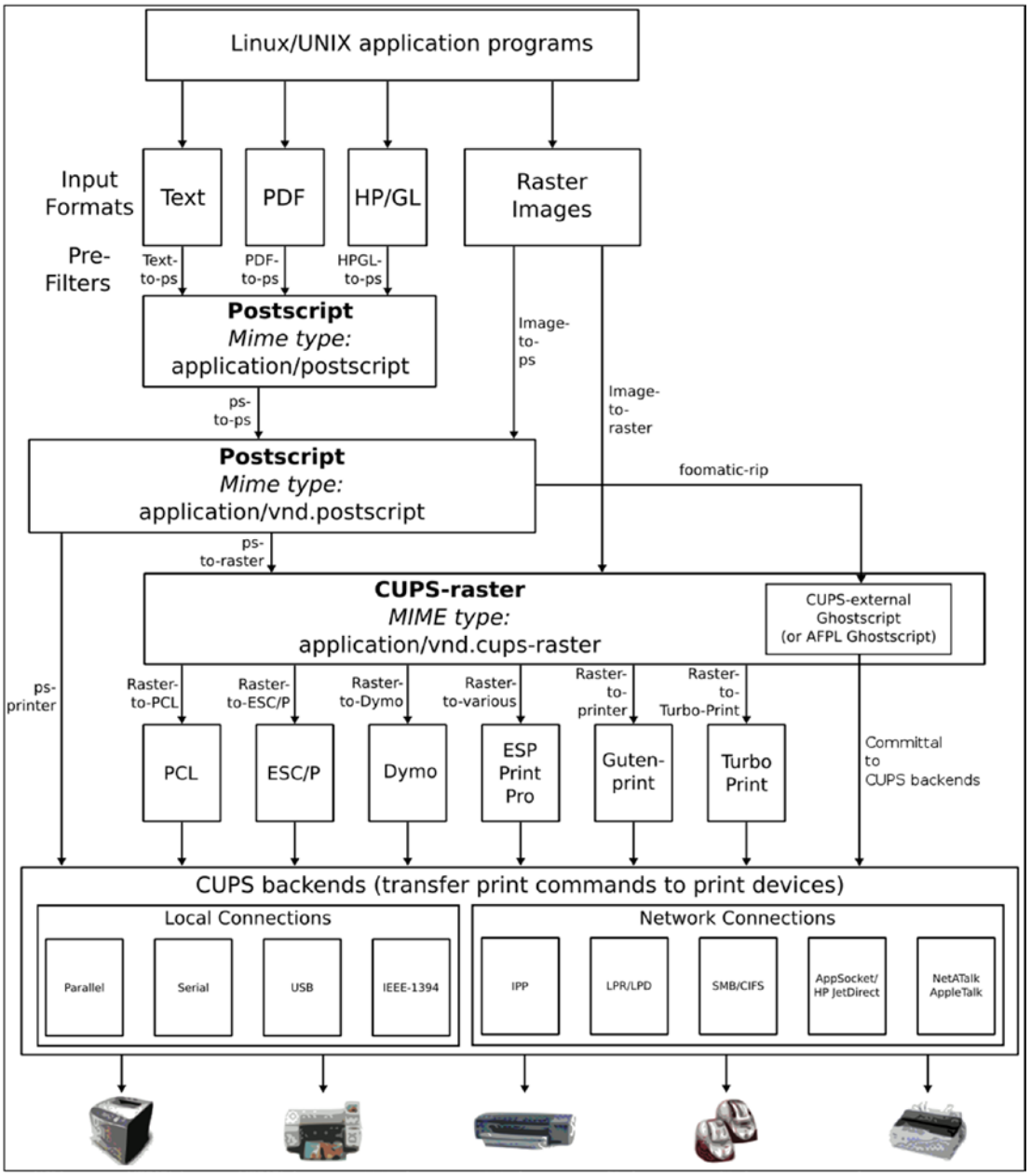


***Figure 7-2.*** *The Common Unix Printing System (CUPS) logical diagram.*
*Glenn Davis (SVG), Ta bu shi da yu (PNG), and Kurt Pfeifle (ASCII) Creative Commons*[9] *Attribution-Share Alike 3.0 Unported*[10] *license.*

[9] https://en.wikipedia.org/wiki/en:Creative_Commons
[10] https://creativecommons.org/licenses/by-sa/3.0/deed.en

CUPS uses Postscript Printer Description (PPD)[11] files to define the features available in each supported printer. Each PPD file contains information about the features and capabilities of the printer. This allows CUPS to interact with the various features such as paper trays with different paper sizes, print quality, duplex, color or black and white, and more. Although they are not true device drivers, PPD files perform somewhat the same functions.

## Creating the print queue

A print queue is a directory that is used to store the print job data stream while it is being spooled to the printer. Although modern-day printers have internal memory, large print jobs may fill that and more. In a high-volume print environment, many jobs may be queued up faster than the printer can print them so they remain in the queue until the printer is ready for them.

In general, the /var/spool directory contains data that is temporarily stored for later processing.[12] CUPS has a queue in /var/spool/cups which contains data about the print jobs as well as the data for the jobs themselves. The /var/spool/lpd/ directory contains a subdirectory for each printer which is used only to store lock files to prevent multiple overlapping attempts to access each printer.

Skip this experiment if you do not have a supported printer connected to the physical host. In Experiment 7-2 we use the command line to configure our printer. It is possible to perform this configuration using the desktop GUI, but there will be times when a GUI will not be available. Many servers do not use a GUI desktop so it will be necessary to use the CLI. However, when you have configured a printer using the CLI, you will also be able to configure one using the GUI tools on the desktop or the web interface that runs on port 631.

[11]Wikipedia, *Postscript Printer Description*, https://en.wikipedia.org/wiki/PostScript_Printer_Description

[12]Refer to the Linux Filesystem Hierarchical Standard (FHS) which we explored in Chapter 19.

Many of the commands we will encounter begin with "lp" which is a holdover from the early days of printing and refers to a "line printer." Note that what we are actually doing is creating a print queue for the printer.

## EXPERIMENT 7-2

If you do not have a physical printer connected to the virtual machine through the physical host, you can still perform the first part of this experiment. This experiment should be performed as root. We will use the command line to create a print queue for the attached printer.

First, let's find the printer on the USB bus just to verify that it has been recognized. The printer I am using is a Brother HL-2270DW. You will probably have a different printer. If you do not have a physical printer, nothing will show up here.

```
[root@studentvm1 ~]# lsusb
Bus 001 Device 002: ID 04f9:0042 Brother Industries, Ltd HL-2270DW Laser
Printer
Bus 001 Device 001: ID1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID1d6b:0001 Linux Foundation.1 root hub
[root@studentvm1 ~]#
```

And then find the uniform resource identifier (URI) that we will need to create the print queue. The **lpinfo -v** command is used to list all of the available buses, protocols, and any printers attached to each.

```
[root@studentvm1 ~]# lpinfo -v
network https
network ipp
network http
network ipps
network beh
direct usb://Brother/HL-2270DW%20series?serial=C1J695917
network socket
network lpd
network smb
```

The following command provides a more detailed listing and describes the bus or protocol for each possibility.

```
[root@studentvm1 ~]# lpinfo -lv
```

We also need to find the Postscript Printer Definition (PPD) file to use for this printer so we can pass that as an option argument when we create the print queue. First let's list all of the PPD files located in the printer model directory.

```
[root@studentvm1 ~]# lpinfo -m | less
```

Scroll through these results to get an idea of the printers that are supported by CUPS. There are well over 15,000 – yes fifteen thousand – entries in this file, although many of them are intended to support the same printers using different printer languages such as supporting a Xerox WorkCentre 7345 with HP PCL5C, LaserJet 4, LaserJet 4d, or Postscript.

Find the entries for the Brother HL-22 series of printers and notice that there is no entry for the HL-2270. In such a case, I always use a PPD file that seems to come the closest to the attached printer. For my system I am using the PPD file for the HL-2250. On my physical host, using this PPD and printer combination works fine including all of the functions such as double sided.

---

**Caution** If you do not have a physical printer attached to your VM, you should read the rest of this experiment but you should not actually enter these commands.

---

Now let's add the new print queue using the following options:

- **-p** specifies the name of the printer. This is a text name with no spaces and is how we identify the printer in commands.
- **-E** enables the printer so that it will accept print jobs.
- **-v** specifies the printer URI. This is the target URI to which the data streams for this print queue are sent.
- **-m** is the name of the standard Postscript Printer Definition (PPD) file contained in the model directory. If a PPD file is provided by the vendor and not by the standard CUPS model directory, as is the case with some commercial high-capacity Xerox printers that I have encountered, use the -P (uppercase) option and the fully qualified file name instead of the -m option.

Be sure to use the printer name, URI, and PPD file that matches the printer connected to your virtual machine.

**Tip** Where the -E flag is in the lpadmin command matters greatly, according to the man page. If you put it at the beginning, you'll encrypt the connection to the printer instead of enabling it.

```
[root@studentvm1 ~]# lpadmin -p Brother-HL-2270DW -E -v usb://Brother/HL-
2270DW%20series?serial=C1J695917 -m gutenprint.5.2://brother-hl-2250dn/simple
```

At this point the print queue should be displayed in the GUI Print Settings. As another point of comparison, Jason's command entry here was

```
lpadmin -p HP-Color-LaserJet-CP2025dn -E -v  usb://HP/Color%20LaserJet%20
CP2025dn?serial=00JPBFR09471 -m gutenprint.5.2://hp-clj_cp2025dn/simple
```

It is OK to verify this, but do not make any changes using the GUI. We can use the **lpstat** command to verify that the print queue has been created.

```
[root@studentvm1 ~]# lpstat -t
scheduler is running
no system default destination
device for Brother-HL-2270DW: usb://Brother/HL-2270DW%20
series?serial=C1J695917
Brother-HL-2270DW accepting requests since Thu4 Mar 2019 09:05:36 PM EDT
printer Brother-HL-2270DW is idle.  enabled since Thu4 Mar 2019 09:05:36 PM
EDT
```

We can also use the **lpstat** command to list the names of all print queues and whether the CUPS server is running on our localhost. The -e option lists the print queue names and -r displays the status of the server.

```
[root@studentvm1 ~]# lpstat -er
Brother-HL-2270DW
scheduler is running
```

We should test the printer and the newly created queue before we proceed. We can send an ASCII plain text file to the printer as our first test. Be sure to use the printer name exactly as it appears in the output from the **lpstat** command.

**Tip** Some printer commands use -P (uppercase) and some use -p (lowercase) options to specify the target print queue.

```
[root@studentvm1 ~]# lpr -P Brother-HL-2270DW /etc/fstab
```

A somewhat related part of the SysAdmin's job is to deal with broken and recalcitrant hardware. It is not all about dealing with software. While reviewing this chapter, Jason was delayed for a while by a troublesome bit of hardware. He says, "It also took a while because my jam sensor [on the printer] is overly sensitive and I had to use a piece of tape to fix it, but that's probably out of scope for this [course]."

Actually not so much – it is in the scope of this course. In fact, hardware problems like this are relatively frequent. Many times they can appear to be software problems and we need to track them down and fix them too. Mechanical devices fail rather too frequently for my liking. Things like fans, hard drives, and printers are all mechanical and have a tendency to fail when it is most inopportune.

The printer has now been created and tested, but we don't want to have to type its name every time we print a document. Just because we have only one printer does not make that printer the default. So we need to explicitly make it the default printer using the **lpoptions** command with the -d option.

```
[root@studentvm1 ~]# lpoptions -d Brother-HL-2270DW
```

This command sets the target printer as the default. Now we can print to the default printer without explicitly specifying the destination. The following command prints the /etc/bashrc file to the default printer.

```
[root@studentvm1 ~]# lpr /etc/bashrc
```

Let's use one of the print options to format a text file for printing. This is the prettyprint option which prints a small heading on each sheet of paper with the file name, the data and time, and a page number. Setting this option is easy.

```
[root@studentvm1 ~]# lpoptions -p Brother-HL-2270DW -o prettyprint=true
```

It should not be necessary to restart CUPS to make this change. As the student user, print the cpuHog file.

```
[student@studentvm1 ~]$ lpr cpuHog
```

Now, again as student, print the /etc/bashrc file. In this case the file does not print in prettyprint format as the cpuHog did. Although the man page does not state the reason, a bit of experimentation has led me to the conclusion that only Bash programs that begin with the shebang line, #!/bin/bash, will print in pretty format.

We could add multiple print queues if we have more than one printer, but only one can be the default. All print jobs are sent to the default queue unless a different one is specified.

The **lpr** and **lpoptions** commands have a number of options that can be used to set things such as print job priorities so that important jobs are printed first; specify text that describes the location of the printer; the number of copies of a print job to print; how many pages of the print job to print on a sheet of paper, for example, two pages of the print job on a side of each sheet of paper; whether to print in duplex mode, that is, front and back of the paper; banner pages to begin and end print jobs as a way to separate them in high-volume environments; and much more. These are all the same items that can be configured using the GUI printer settings interface.

## Printing to a PDF file

Now since at least some of us do not have a physical printer to use, we will install the cups-pdf RPM package that will allow us to print directly to a PDF file. Many GUI applications allow export to a PDF file, and the GUI print manager interface has an option to print to a file. This package accomplishes the same thing for us for printing from the command line.

**Note** Everyone should perform Experiment 7-3 because we will use the PDF print queue to do all further printing for the rest of the experiments in this chapter.

## EXPERIMENT 7-3

Start to perform this experiment as root; we will switch between the root and student users. First, as root, install the cups-pdf RPM package and verify the new print queue has been created.

```
[root@studentvm1 ~]# dnf -y install cups-pdf
<snip>
[root@studentvm1 ~]# lpstat -a
Brother-HL-2270DW accepting requests since Fri5 Mar 2019 04:17:14 PM EDT
Cups-PDF accepting requests since Fri5 Mar 2019 04:26:33 PM EDT
```

Now we make the Cups-PDF queue the default.

```
[root@studentvm1 ~]# lpoptions -d Cups-PDF
```

And verify

```
[root@studentvm1 ~]# lpq
Cups-PDF is ready
no entries
[root@studentvm1 ~]#
```

or

```
[root@studentvm1 ~]# lpstat -d
system default destination: Cups-PDF
```

Unfortunately, cups-pdf has been configured to place the PDF print files in the user's desktop directory, ~/Desktop, which is not what we want and, in my opinion, putting things like files and directories on the desktop is a horrible idea anyway. So we will change this to a new directory that we will create for the student user, ~/chapter7.

As the student user, create the new directory.

```
[student@studentvm1 ~]$ mkdir ~/chapter7
```

We need to change the /etc/cups/cups-pdf.config file to use this new directory. As root, edit the /etc/cups-pdf.config file.

Comment out the following line – shown as already commented out here.

```
# Out ${DESKTOP}
```

And add the following line immediately below it.

```
Out ${HOME}/chapter7
```

Restart CUPS.

```
[root@studentvm1 ~]# systemctl restart cups
```

As the student user, print a test file and verify that it was created in the ~/chapter7 directory.

```
[student@studentvm1 ~]$ lpr cpuHog ; ll chapter7
total 7172
-rw------- student student   20169 Mar5 22:36 cpuHog.pdf
```

As the student user, open the file manager using the Home icon on the desktop, if it is not already open. Navigate to the ~/chapter7 directory and double-click the cpuHog.pdf file. This opens the Evince document viewer and displays the content.

Figure 7-3 shows the two print queues we have created so far. It also shows the cpuHog.pdf file in the ~/chapter7 directory and the Evince document viewer showing the content of the print file.

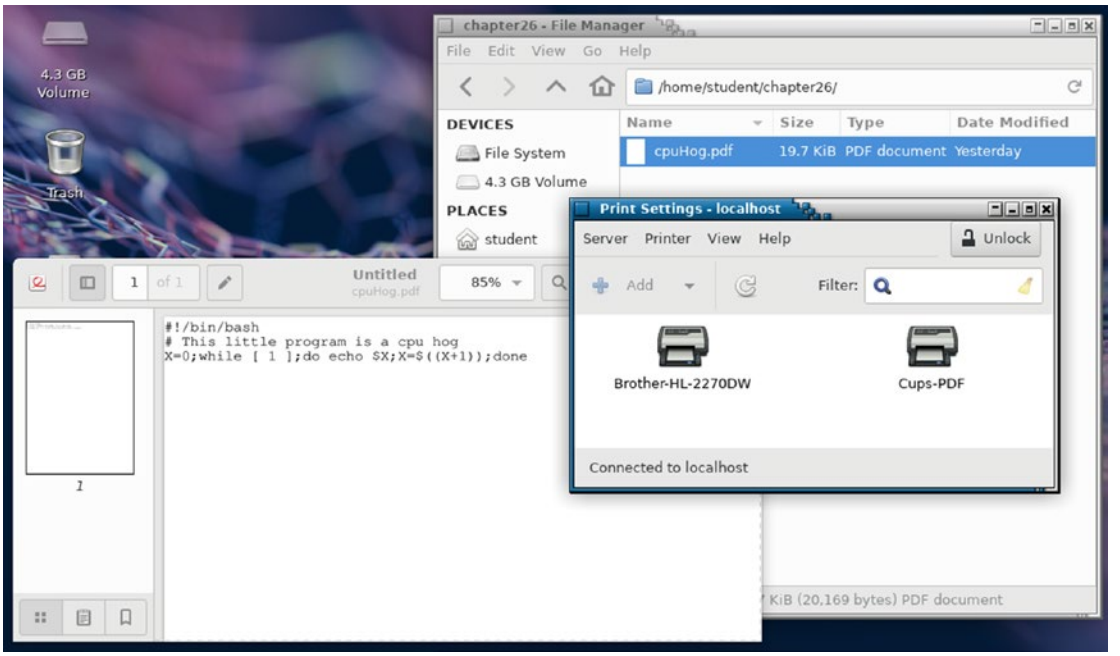


***Figure 7-3.*** *The two print queues created and the Evince document viewer showing the content of the cpuHog.pdf print file*

Now, in order to see the status of the print queue for the physical printer, in my case the HL-2270, I need to specify the desired print queue using the -P option.

```
[root@studentvm1 ~]# lpq -P Brother-HL-2270DW
Brother-HL-2270DW is ready
```

All print jobs sent to the default – Cups-PDF – print queue will now be saved as PDF files in the /home/student/chapter7 directory so they can be easily found.

The system-wide printer configuration files are located in the /etc/cups/ directory. This includes the lpoptions file that contains options settings that the root user has made from the command line. The printers.conf file contains the current status information for all defined print queues.

# File conversion tools

As SysAdmins, we frequently work with ASCII[13] plain text files and text files encoded as UTF-8 or UTF-16. ASCII text files, such as configuration files and those created by editors like Vim, can be printed directly from the command line, but they may not always be formatted well for printing. There are also times when it may be necessary – it has been for me – to convert a word processing document from MS Word or LibreOffice Writer format to ASCII text. In this section we will explore tools that can perform those conversions and more.

**Tip** Many non-US users perform operations on UTF-8 or UTF-16 plain text files. That's less of an issue with system files since they will always be encoded as ASCII, but once you get into file conversion and printing, it's something to watch out for. The **`lpr`** program won't print the parts in encodings that it doesn't recognize, because Postscript doesn't (natively) support those other encodings.

Let's start by installing a tool that is not installed by default.

[13]Wikipedia, *ASCII*, https://en.wikipedia.org/wiki/ASCII

**EXPERIMENT 7-4**

Perform this experiment and begin as the root user to one new package. Note that a large number of packages needed to meet dependencies are also installed.

```
[root@studentvm1 ~]# dnf -y install a2ps
```

This tool and some that are already installed provide us with the ability to manipulate and convert files into various formats. Although there can be many reasons for making these conversions, in this chapter we mostly use them to prepare files for printing. Although plain ASCII text files can be printed directly to a printer, the results are not always aesthetically pleasing because there are no margins, line wrap may be quite random, and some lines may be split with the top half of the characters printed at the bottom of one page and the bottom half printed at the top of the next. This makes reading the file difficult and frustrating so I like to prepare text files a bit so that they print in a more readable format.

## a2ps

The a2ps utility is my favorite for preparing text files for printing. It converts plain ASCII text files to Postscript files suitable for printing. By default, the resulting data stream in Postscript format is sent to the default printer. The original file is not altered.

The features I like most about a2ps are the page formatting ones. They allow me great flexibility to define the page format if I do not like the default format. This tool does not require the prettyprint option to be set and is more predictable in its results.

The default format is to print two pages per side of paper and to place a frame around the text. The top area of the frame contains the date the file was last modified, the file name, and the page number with the total number of pages. The bottom area of the frame contains the date the file was printed, the fully qualified file and path, and the sheet number and the total number of sheets.

## EXPERIMENT 7-5

This experiment should be performed as the student user. Be sure that ~/chapter7 is the PWD.

This command converts the data in the data stream from the /etc/bashrc file to Postscript. The -o (output file) option sends the resulting Postscript data stream to the file bashrc.ps. The .ps extension is for Postscript files.

```
[student@studentvm1 chapter7]$ a2ps /etc/bashrc -o bashrc.ps
[/etc/bashrc (plain): 2 pages on 1 sheet]
[Total: 2 pages on 1 sheet] saved into the file `bashrc.ps'
[3 lines wrapped]
```

Now verify the Postscript file was created.

```
[student@studentvm1 chapter7]$ ll
total 44
-rw-rw-r-- 1 student student 21326 Mar 17 22:10 bashrc.ps
-rw------- 1 student student 20169 Mar 17 15:02 cpuHog.pdf
```

The resulting bashrc.ps file can be sent to the printer. Ensure that Cups-PDF is the default print queue for the student user.

```
[student@studentvm1 chapter7]$ lpoptions -d Cups-PDF
```

---

**Tip** Because we have run the preceding lpoptions command as the student user, any options set by this command override – for the student user only – the system-wide option settings for the target printer. The local options are stored in the ~/.cups/lpoptions configuration file. Each user can have their own printer option settings. To return to use of the system-wide options, delete this file.

---

We can now print the Postscript file from the command line.

```
[student@studentvm1 chapter7]$ lpr bashrc.ps ; ll
total 60
-rw------- 1 student student 16369 Mar 18 12:29 bashrc.pdf
-rw-rw-r-- 1 student student 21326 Mar 17 22:10 bashrc.ps
-rw------- 1 student student 20169 Mar 17 15:02 cpuHog.pdf
```

Use the Thunar GUI file manager to double-click the bashrc.ps and bashrc.pdf files to open the Evince document viewer and look at each of these two documents. They are in different formats but they should appear identical.

Remember that the Cups-PDF printer queue converts files to PDF, and we reconfigured it to store them in the /home/student/chapter7 directory. Had we sent the bashrc.ps file to the print queue for the physical printer, it would have been printed.

I suggest reading the a2ps man page in order to understand the full range of its capabilities.

## ps2pdf

The ps2pdf utility converts Postscript files to PDF using Ghostscript.[14] Ghostscript is a tool that is typically used to rasterize page description languages into images that can be displayed on a graphical terminal or desktop or printed on a printer.

**EXPERIMENT 7-6**

This experiment should be performed as the student user.

Use the following command to convert the bashrc.ps file to bashrc-2.pdf. Add the -2 to the name because we already have bashrc.pdf in this directory and we do not want to overwrite the existing file.

```
[student@studentvm1 chapter7]$ ps2pdf bashrc.ps bashrc-2.pdf
```

Use the Evince viewer to check the results. You might find it interesting to compare the two files using the **cmp** (compare) and **diff** (difference) commands.

## pr

The **pr** utility converts plain ASCII text files into something a little prettier for printing. It simply paginates the text and adds a header to each page of text. The header consists of the date and time the file was run through the **pr** utility, the name of the file, and a page

[14]Wikipedia, *Ghostscript*, https://en.wikipedia.org/wiki/Ghostscript

number. The result is still an ASCII text data stream which can be redirected to a file, sent directly to a printer, or viewed with the **less** utility.

The **pr** utility has a few options that can be used to control things like page length, indent (left margin), line width for truncation, and more, but the essential function is still very simple.

**EXPERIMENT 7-7**

This experiment should be performed as the student user.

The following command does some very basic modification to the /etc/bashrc file to make it print with a little nicer formatting than to just print the raw data stream.

```
[student@studentvm1 ~]$ pr /etc/bashrc | less
```

Page through the data and view the headers and pagination. Print the same file using a bit of left margin.

```
[student@studentvm1 ~]$ pr -o 4 /etc/bashrc | less
```

## ps2ascii

The ps2ascii is a Ghostscript translator that can extract the ASCII text from Postscript or PDF files. It does not always work very well, sometimes producing no output when there should be some. Most of the time it just extracts the text without regard to any formatting. That does not mean it might not be useful. The output from this tool is in ASCII text to STDOUT.

**EXPERIMENT 7-8**

Perform this experiment as the student user. This is what the bashrc.ps file looks like after being run through the ps2ascii utility. I have removed much of the data stream to save space.

```
[student@studentvm1 chapter7]$ ps2ascii bashrc.ps                    1/1
       Page 2/2
     Printed by Student User
       bashrc
       Mar 17, 19 14:55
```

```
                        PATH=$1:$PATH              fi        esac    }    #
By default, we want umask to get set. This sets it for non-login
shell.    # Current threshold for system reserved uid/gids is 200    # You
could check uidgid reservation validity in    # /usr/share/doc/setup-*/
uidgid file    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ];
then       umask 002    else       umask 022    fi    SHELL=/bin/bash    #
Only display echos from profile.d scripts if we are no login shell    #
and interactive - otherwise just process them to set envvars    for i in /
etc/profile.d/*.sh; do        if [ -r "$i" ]; then          if [ "$PS1"
]; then                . "$i"          else                . "$i" >/dev/
null            fi        fi    done    unset i    unset -f pathmunge  fifi#
vim:ts=4:sw=4
                                                                  /etc/bashrc

       Page 1/2
       bashrc
       Mar 17, 19 14:55
         # /etc/bashrc# System wide functions and aliases# Environment stuff
         goes in /etc/profile# It's NOT a good idea to change this file
         unless you know what you# are doing. It's much better to

<snip>

# If you want to do so, just add e.g.    # if [ "$PS1" ]; then    #   PS1="[\
u@\h:\l \W]\\$ "    # fi    # to your custom modification shell script in
/etc/profile.d/ directory  fi  if ! shopt -q login_shell ; then # We're
not a login shell    # Need to redefine pathmunge, it gets undefined
at the end of /etc/profile    pathmunge () {        case ":${PATH}:"
in            *:"$1":*)               ;;            *)               if
[ "$2" = "after" ] ;
then                    PATH=$PATH:$1              elseSunday March 17, 2019
```

For obvious reasons, I always keep the original ASCII plain text versions of any files I convert to other formats.

# Operating system–related conversion tools

Different operating systems use slightly different non-text codes when storing ASCII text files. For example, DOS and Windows use a carriage-return/line-feed (CR-LF) sequence at the end of each line where Unix and Linux use a single line-feed character, which is called a newline in Unix and Linux. And Apple uses carriage return at the end of a line of text. These characters are not normally displayed using tools like editors and paging tools such as **less**. Thus, they are called whitespace characters because they are invisible.

We will use the cpuHog program we created earlier for these experiments because it is short.

## EXPERIMENT 7-9

Perform this experiment as the student user. Use the **od** command to view the cpuHog file as rendered into ASCII characters including whitespace character codes. The newline character is rendered as a \n.

```
[student@studentvm1 chapter7]$ od -c ../cpuHog
0000000   #   !   /   b   i   n   /   b   a   s   h  \n   #       T   h
0000020   i   s       l   i   t   t   l   e       p   r   o   g   r   a
0000040   m       i   s       a       c   p   u       h   o   g  \n   X
0000060   =   0   ;   w   h   i   l   e       [       1       ]   ;   d
0000100   o       e   c   h   o       $   X   ;   X   =   $   (   (   X
0000120   +   1   )   )   ;   d   o   n   e  \n  \n  \n
0000134
```

Notice the \n (newline) characters. Now look at it as Hexadecimal (Hex) code. The -x option specifies Hex.

```
[student@studentvm1 chapter7]$ od -x ../cpuHog
0000000 2123 622f 6e69 622f 7361 0a68 2023 6854
0000020 7369 6c20 7469 6c74 2065 7270 676f 6172
0000040 206d 7369 6120 6320 7570 6820 676f 580a
0000060 303d 773b 6968 656c 5b20 3120 5d20 643b
0000100 206f 6365 6f68 2420 3b58 3d58 2824 5828
0000120 312b 2929 643b 6e6f 0a65 0a0a
0000134
```

Can you decode the Hex into ASCII? You could use the tables in the Wikipedia article on ASCII referenced in footnote 11, but a good SysAdmin always has an ASCII table available. You can always find one on the ASCII man page.

**Tip** Some tools, such as the printf command which can be used to produce a formatted print in command-line programs and scripts, use ASCII codes like \n for newline and \t for a tab.

## unix2dos

Now that we can see the newline characters, let's convert this file (without changing the original) to a DOS format.

### EXPERIMENT 7-10

Perform this experiment as the student user. Convert the cpuHog file into a DOS format without changing the original. The default for the unix2dos utility is to convert the original file to the new format, and this is not what we want for this experiment.

```
[student@studentvm1 ~]$ unix2dos -n cpuHog cpuHog.dos
unix2dos: converting file cpuHog to file cpuHog.dos in DOS format…
[student@studentvm1 ~]$ od -c cpuHog.dos
0000000   #   !   /   b   i   n   /   b   a   s   h  \r  \n   #       T
0000020   h   i   s       l   i   t   t   l   e       p   r   o   g   r
0000040   a   m       i   s       a       c   p   u       h   o   g  \r
0000060  \n   X   =   0   ;   w   h   i   l   e       [       1       ]
0000100   ;   d   o       e   c   h   o       $   X   ;   X   =   $   (
0000120   (   X   +   1   )   )   ;   d   o   n   e  \r  \n  \r  \n  \r
0000140  \n
0000141
[student@studentvm1 ~]$ od -x cpuHog.dos
0000000 2123 622f 6e69 622f 7361 0d68 230a 5420
0000020 6968 2073 696c 7474 656c 7020 6f72 7267
0000040 6d61 6920 2073 2061 7063 2075 6f68 0d67
0000060 580a 303d 773b 6968 656c 5b20 3120 5d20
```

```
0000100 643b 206f 6365 6f68 2420 3b58 3d58 2824
0000120 5828 312b 2929 643b 6e6f 0d65 0d0a 0d0a
0000140 000a
0000141
```

The newline characters have been converted to CR-LF format.

We can use the **diff** command to verify that the files are different. See what the developers did with the name there?

```
[student@studentvm1 ~]$ diff cpuHog cpuHog.dos
1,5c1,5
< #!/bin/bash
< # This little program is a cpu hog
< X=0;while [ 1 ];do echo $X;X=$((X+1));done
<
<
---
> #!/bin/bash
> # This little program is a cpu hog
> X=0;while [ 1 ];do echo $X;X=$((X+1));done
>
>
```

Although this result shows that there is a difference in every line of the files, it does not show the actual differences because they are in the whitespace and cannot be directly displayed by this tool. We can however eliminate whitespace from comparison which will now show that there are no differences.

```
[student@studentvm1 ~]$ diff -Z cpuHog cpuHog.dos
[student@studentvm1 ~]$
```

This result tells us that the differences in the two files are all in the whitespace.

I find it unusual and somewhat less than helpful that the output data stream from the unix2dos utility cannot be sent to STDOUT and especially so that it does not use STDIO at all. This tool, while useful in a mixed operating system environment, does not follow the Linux Philosophy.

Refer to the unix2dos man page for details of its capabilities and syntax.

## dos2unix

The dos2unix utility performs the reverse of the unix2dos tool by converting Linux text files into ones suitable for DOS.

### EXPERIMENT 7-11

Perform this experiment as student. The syntax and lack of STDIO capabilities for **dos2unix** are the same as those of **unix2dos**.

```
[student@studentvm1 ~]$ dos2unix -n cpuHog.dos cpuHog.Linux
dos2unix: converting file cpuHog.dos to file cpuHog.Linux in Unix format...
```

Now use the **od** command to view the cpuHog.Linux file in ASCII columnar mode using the -c option and look for the newlines (\n). Note also the sizes of the cpuHog.Linux file and the original cpuHog files compared to the cpuHog.dos file. It is also informative to use the **diff** command to compare the original cpuHog file and the cpuHog.Linux file after the latter has been through two transitions.

```
[student@studentvm1 ~]$ diff cpuHog cpuHog.Linux
```

You can see that there are no differences.

## unix2mac and mac2unix

The **unix2mac** utility does just what its name implies; it converts text files from Linux formats to Apple formats. The **mac2unix** tool obviously performs the reverse process.

### EXPERIMENT 7-12

Perform this first part of the experiment as the student user. The syntax of these commands are the same as that for the previous commands. Let's convert the cpuHog file to Apple format and then view the content.

```
[student@studentvm1 ~]$ od -c cpuHog.mac
0000000   #   !   /   b   i   n   /   b   a   s   h  \r   #       T   h
0000020   i   s       l   i   t   t   l   e       p   r   o   g   r   a
0000040   m       i   s       a       c   p   u       h   o   g  \r   X
```

```
0000060   =   0   ;   w   h   i   l   e       [       1       ]   ;   d
0000100   o       e   c   h   o       $   X   ;   X   =   $   (   (   X
0000120   +   1   )   )   ;   d   o   n   e  \r  \r  \r
0000134
```

Apple text files use only the carriage-return (CR), the \r character at the end of a line of text. There is a little secret here. Let's look at the /usr/bin directory which is where these binary files are kept.

```
[student@studentvm1 ~]# ll /usr/bin | grep unix
-rwxr-xr-x. 1 root root      55192 Jul 23  2018 dos2unix
lrwxrwxrwx. 1 root root          8 Jul 23  2018 mac2unix -> dos2unix
-rwxr-xr-x. 1 root root      55184 Jul 23  2018 unix2dos
lrwxrwxrwx. 1 root root          8 Jul 23  2018 unix2mac -> unix2dos
```

Think about what that means.

# Miscellaneous tools

There are some additional tools that I have found interesting and useful. Let's look at three of them.

## lpmove

The **lpmove** command can be used to move a print job from one queue to another. This might be necessary if you print a file to a print queue and the printer is out of paper or toner and is not accepting print jobs. The jobs will enter the queue and stay there until the printer is resupplied and once again accepting print jobs.

Moving a print job from one queue to another allows completion of your print job without the need to wait for the original printer to become ready again. To set up this next experiment so that everyone can do it, we will create a dummy printer that sends all print jobs to /dev/null.

A dummy printer can also be useful for testing code that sends a data stream to a printer but where you don't care about the print job itself. You don't want to waste paper on a real printer, and you don't want to clean up a bunch of files that were created during testing. In such an instance, you will test the contents of the print jobs at a different time to verify that they are correct.

## EXPERIMENT 7-13

Start this experiment as the root user. Because some of you do not have access to a physical printer, we are going to create a dummy printer that sends all of the print jobs directly to the /dev/null device special file. We explored device special files and /dev/null in Chapter 3 of this volume.

This gives us another queue to work with for this experiment so that everyone can do this. Add the new dummy print queue and verify it.

```
[root@studentvm1 ~]# lpadmin -p DummyPrinter -E -v file:/dev/null
[root@studentvm1 ~]# lpstat -t
scheduler is running
system default destination: Cups-PDF
device for Brother-HL-2270DW: usb://Brother/HL-2270DW%20
series?serial=C1J695917
device for Cups-PDF: cups-pdf:/
device for DummyPrinter: /dev/null
Brother-HL-2270DW accepting requests since Sun 17 Mar 2019 02:51:32 PM EDT
Cups-PDF accepting requests since Mon 18 Mar 2019 12:29:36 PM EDT
DummyPrinter accepting requests since Thu 21 Mar 2019 01:33:26 PM EDT
printer Brother-HL-2270DW is idle.  enabled since Sun 17 Mar 2019 02:51:32 PM EDT
printer Cups-PDF is idle.  enabled since Mon 18 Mar 2019 12:29:36 PM EDT
printer DummyPrinter is idle.  enabled since Thu 21 Mar 2019 01:33:26 PM EDT
```

Now let's print a test to our dummy printer. Nothing should be printed but we should also get no errors. Do this as the student user.

```
[student@studentvm1 ~]# lpr -P DummyPrinter cpuHog
[student@studentvm1 ~]# lpq -P DummyPrinter
DummyPrinter is ready
no entries
```

Now, as root, let's disable the dummy printer.

```
[root@studentvm1 ~]# cupsdisable DummyPrinter
[root@studentvm1 ~]# lpstat -t
scheduler is running
system default destination: Cups-PDF
```

```
device for Brother-HL-2270DW: usb://Brother/HL-2270DW%20
series?serial=C1J695917
device for Cups-PDF: cups-pdf:/
device for DummyPrinter: /dev/null
Brother-HL-2270DW accepting requests since Sun 17 Mar 2019 02:51:32 PM EDT
Cups-PDF accepting requests since Mon 18 Mar 2019 12:29:36 PM EDT
DummyPrinter accepting requests since Thu 21 Mar 2019 02:06:07 PM EDT
printer Brother-HL-2270DW is idle.  enabled since Sun 17 Mar 2019 02:51:32 PM EDT
printer Cups-PDF is idle.  enabled since Mon 18 Mar 2019 12:29:36 PM EDT
printer DummyPrinter disabled since Thu 21 Mar 2019 02:06:07 PM EDT -
        Paused
```

As the student user, send a job to the dummy printer now that it is disabled and then check the queue.

```
[student@studentvm1 ~]$ lpr -P DummyPrinter cpuHog
[student@studentvm1 ~]$ lpq -P DummyPrinter
DummyPrinter is not ready
Rank    Owner   Job     File(s)                         Total Size
1st     student 32      cpuHog                          1024 bytes
[student@studentvm1 ~]$
```

Our print job is still in the queue belonging to the DummyPrinter. The print job would "print" to /dev/null if we re-enable the DummyPrinter, but we are going to move it to the Cups-PDF queue instead.

```
[student@studentvm1 ~]$ lpmove 32 Cups-PDF
[student@studentvm1 ~]$ lpq
Cups-PDF is ready
no entries
```

Remember that print jobs sent to the Cups-PDF printer are processed and sent as PDF files to ~/chapter7, so we can look there for the resulting "print" document. The date on the cpuHog.pdf file should be only a few seconds old, which tells us that it was just printed. You should delete this file and then do this part of the experiment over just to verify that.

```
[student@studentvm1 ~]$ ll chapter7
total 76
-rw-rw-r-- 1 student student 16286 Mar 19 08:19 bashrc-2.pdf
-rw------- 1 student student 16369 Mar 18 12:29 bashrc.pdf
-rw-rw-r-- 1 student student 21326 Mar 17 22:10 bashrc.ps
-rw------- 1 student student 20169 Mar 21 15:27 cpuHog.pdf
```

Look at the Print Settings GUI tool on the desktop. There are now three print queues and the newest, DummyPrinter, is one of them. Notice that the DummyPrinter print queue is paused because the icon for that queue has a pause symbol (||) superimposed on it. Leave that DummyPrinter disabled.

## wvText and odt2txt

The **wvText** tool can be used to convert MS Word documents to text format and **odt2txt** converts LibreOffice documents from OpenDocument Text format to plain ASCII text. I have not used **wvText**, but I have used **odt2txt**.

Sometimes I need to determine whether I have discussed a particular topic in another chapter of this book. Each chapter is a separate LibreOffice Writer file. So I do something like that in Figure 7-4 to search all of these chapters for a word or phrase.

```
[dboth@david RevisionsCompleted]$ for I in `ls *odt`; do echo "### Working on $I" ; odt2txt
$I ; done | grep -Ei "columns|## Working"
### Working on Chapter-01.odt
### Working on Chapter-02.odt
### Working on Chapter-03.odt
### Working on Chapter-04.odt
on your screen if there are not enough columns in your terminal
your screen if there are not enough columns in your terminal
### Working on Chapter-05.odt
### Working on Chapter-06.odt
### Working on Chapter-07.odt
### Working on Chapter-08.odt
### Working on Chapter-09.odt
### Working on Chapter-10.odt
### Working on Chapter-11.odt
### Working on Chapter-12.odt
### Working on Chapter-13.odt
load the system. It is also interactive and the data columns to
terminal display. The default columns displayed by top are
described below. Several other columns are available and each
any of the displayed columns including CPU and memory usage. By
enough width (columns) the output may be misaligned and
### Working on Chapter-14.odt
is opened. I currently have this adjusted to 130 columns by 65
### Working on Chapter-15.odt
### Working on Chapter-16.odt
### Working on Chapter-17.odt
What is the value of the COLUMNS variable in each of the open
```

***Figure 7-4.*** *Using the odt2txt utility to search for specific words*

By using the echo command to list the chapters as they are converted and scanned, it is easy to identify in which chapter each instance was found. I used the time utility to perform this task to see how long it takes. The conversions and content search took less than 2 seconds of real time. This is much faster than opening each LibreOffice file and using the LibreOffice search tool. Way faster.

Neither of these tools deal with tables well. The data in the tables is converted but the organization is lost.

## Chapter summary

This chapter has explored in some detail the creation and use of print queues and printing from the command line. Although all of this can be accomplished from the GUI and GUI applications, using the command line to perform these tasks fosters a more complete understanding of how printing works in Linux.

We looked at the tools used in this chapter primarily from the standpoint of printing or preparing ASCII plain text files for printing. However, these same tools can be used to manipulate text files in various ways to produce PDF and Postscript files for sending to others as well as long-term retention of text document files in forms that are easy to read and understand. These tools can also be used to allow ASCII text file sharing between operating systems that use somewhat different encoding standards.

Because of the large number of options that the tools we have explored in this chapter can use to control print jobs, it is a good idea to read the man pages for each. There are many possibilities.

Leave the external physical printer connected, if you have one. It will be used in the next chapter.

## Exercises

Perform the following exercises to complete this chapter:

1. If you have a physical printer, add some text to the printer queue that describes the printers' physical location.
2. If you have a physical printer, reprint the /etc/bashrc file and format it for two sided on the long edge and with a double border.

3. Print (to the display) the list of all printer queues on the VM.
4. If you have a physical printer, devise and conduct an experiment which proves that the prettyprint option only works on Bash files with the shebang line.
5. Does prettyprint work when using the Cups-PDF print queue?
6. Can users set different default printers from each other and from the root user?
7. Which Hex character represents a newline?
8. What is the ASCII representation for a tab character and the Hex code for it?
9. In Experiment 7-12 you looked at the unix2dos, unix2mac, dos2unix, and mac2unix executable files. What did you conclude from that observation?
10. What happens to the print queue for a printer when the physical printer is disconnected from the VM?

CHAPTER 8

# Hardware Detection

## Objectives

In this chapter you will learn

- To use common Linux tools to detect and identify the hardware installed in a Linux host.
- To determine the motherboard information such as vendor, make, model, and serial number
- To determine the memory type, speed, and size
- To find and list peripheral hardware connected to the system such as printers, mice, keyboards, and more
- To generate a list of the hardware attached to the USB and PCI buses

## Introduction

What exactly do I mean by hardware detection? For me, hardware detection is the ability to identify what hardware is installed in a Linux host and other information such as vendor, model, serial number, memory and hard/SSD drive sizes, and other specific identifying information that might be useful. And I specifically mean without having to take the system apart to do so.

When upgrading memory for a Linux host, for example, I have used the tools we will explore in this chapter to determine the maximum memory supported by a motherboard and what type, as well as how many open memory DIMM slots were available. I could order online or go to my local computer store – a real computer store with people who know what I am talking about, not a big box monstrosity – and tell them what I want, knowing that it will work when I get home and install it.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_8

In Chapter 7 we saw how to use one of these tools to determine that a printer was installed and what make and model. Another useful purpose for these tools is to use them in automated scripts to document the hardware and software installed in a Linux host. We will look at that application for these tools in the next two chapters.

We have already looked briefly at these tools in Chapter 13 of Volume 1, but we will now explore them in more detail. Although the experimental output from these tools will be for the VM, I will sometimes also include data from one or more of my hardware systems so that you can see the results of these commands on physical hardware.

I mentioned this in Chapter 13 of Volume 1 and it bears repeating here. The **lshw** (list hardware) and **dmidecode** (Desktop Management Interface[1] decode) commands both display as much hardware information as is available in SMBIOS.[2] The man page for **dmidecode** states, "SMBIOS stands for System Management BIOS, while DMI stands for Desktop Management Interface. These two standards are tightly related and were developed by the DMTF (Desktop Management Task Force)."

These two utilities use data stored in SMBIOS which is a data storage area on system motherboards that allows the BIOS boot process to access data about the system hardware. This hardware data is collected from the SMBIOS and stored in the running system in the /sysfs special filesystem.

Because the task of collecting hardware data is performed by BIOS during the initial BIOS boot, the operating system does not need to probe the hardware directly in order to collect information. The information collected can be used to perform tasks such as determination of which hardware-related kernel modules to load during the Linux kernel portion of the boot and startup process. We will explore this particular usage with udev and D-Bus in Chapter 14 in this volume.

Much of the data stored in SMBIOS is text data placed there explicitly by the hardware vendors. It is not obtained by actually probing the hardware. The data may be missing some information or some may even be incorrect. I have not found this to be common but it is possible. Nevertheless, the vendors have good reason to ensure that the essential data is accurate. Other data, the actual hardware information such as CPU, memory, and installed devices, is obtained at each boot and stored in the SMBIOS.

[1]Wikipedia, *Desktop Management Interface*, https://en.wikipedia.org/wiki/Desktop_Management_Interface

[2]Wikipedia, *System Management BIOS*, https://en.wikipedia.org/wiki/System_Management_BIOS

# dmidecode

Let's start our hardware exploration with the dmidecode utility which can provide us with an amazing amount of hardware information. It does have some limitations as we will see.

**EXPERIMENT 8-1**

Perform this experiment as root. If you have disabled the print queue for the physical printer, reattach the printer to your VM and re-enable the print queue. Be sure to use the name of your printer.

```
[root@studentvm1 ~]# cupsenable Brother-HL-2270DW
[root@studentvm1 ~]# lpstat -t
scheduler is running
system default destination: Cups-PDF
device for Brother-HL-2270DW: usb://Brother/HL-2270DW%20
series?serial=C1J695917
device for Cups-PDF: cups-pdf:/
device for DummyPrinter: /dev/null
Brother-HL-2270DW accepting requests since Sun 24 Mar 2019 09:22:04 AM EDT
Cups-PDF accepting requests since Thu 21 Mar 2019 03:27:41 PM EDT
DummyPrinter accepting requests since Thu 21 Mar 2019 02:08:37 PM EDT
printer Brother-HL-2270DW is idle.  enabled since Sun 24 Mar 2019 09:22:04 AM EDT
printer Cups-PDF is idle.  enabled since Thu 21 Mar 2019 03:27:41 PM EDT
printer DummyPrinter disabled since Thu 21 Mar 2019 02:08:37 PM EDT -
        Paused
```

Starting very simply, we look at the information supplied by dmidecode with no options, which is to say, all the information available in SMBIOS.

```
[root@studentvm1 ~]# dmidecode | less
```

There is too much data to reproduce here, so you will need to refer to the results from your VM. As we proceed through this experiment, I will use data from my primary workstation to illustrate data that would be seen on a physical host. You can follow along using data from your VM or from another physical Linux host if you have root access to one.

Let's explore some of the individual DMI types starting with BIOS itself, DMI type 0. You can find the type codes for all hardware types in the dmidecode man page.

```
[root@david ~]# dmidecode -t 0
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 3.0.0 present.

Handle 0x0000, DMI type 0, 24 bytes
BIOS Information
    Vendor: American Megatrends Inc.
    Version: 0503
    Release Date: 07/11/2017
    Address: 0xF0000
    Runtime Size: 64 kB
    ROM Size: 16 MB
    Characteristics:
           PCI is supported
           APM is supported
           BIOS is upgradeable
           BIOS shadowing is allowed
           Boot from CD is supported
           Selectable boot is supported
           BIOS ROM is socketed
           EDD is supported
           5.25"/1.2 MB floppy services are supported (int 13h)
           3.5"/720 kB floppy services are supported (int 13h)
           3.5"/2.88 MB floppy services are supported (int 13h)
           Print screen service is supported (int 5h)
           8042 keyboard services are supported (int 9h)
           Serial services are supported (int 14h)
           Printer services are supported (int 17h)
           ACPI is supported
           USB legacy is supported
           BIOS boot specification is supported
           Targeted content distribution is supported
           UEFI is supported
    BIOS Revision: 5.13
```

This information lists the AMI as the BIOS vendor along with the BIOS version number and release date. This information might be useful when making a determination of whether a BIOS upgrade is needed along with the information that the BIOS is upgradable.

It lists the device types supported by this BIOS but that are not necessarily installed. For example, various types of floppy diskettes are supported, but I have not installed a floppy drive on any of my system since … well, I cannot remember when because it has been so long.

DMI type 1 contains data about the assembled system. I built my own system so there is only default data for this type on my workstation. The information on your VM should be more informative.

```
[root@david ~]# dmidecode -t 1
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 3.0.0 present.

Handle 0x0001, DMI type 1, 27 bytes
System Information
    Manufacturer: System manufacturer
    Product Name: System Product Name
    Version: System Version
    Serial Number: System Serial Number
    UUID: 27191c80-d7da-11dd-9360-b06ebf3a431f
    Wake-up Type: Power Switch
    SKU Number: SKU
    Family: To be filled by O.E.M.
```

Type 2 contains data for the motherboard, in this case, an ASUSTeK TUF X299.

```
[root@david ~]# dmidecode -t 2
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 3.0.0 present.

Handle 0x0002, DMI type 2, 15 bytes
Base Board Information
    Manufacturer: ASUSTeK COMPUTER INC.
    Product Name: TUF X299 MARK 2
    Version: Rev 1.xx
    Serial Number: 170807951700403
```

```
        Asset Tag: Default string
        Features:
                Board is a hosting board
                Board is replaceable
        Location In Chassis: Default string
        Chassis Handle: 0x0003
        Type: Motherboard
        Contained Object Handles: 0
```

DMI type 4 contains a great deal of information about the processor installed in the host. It has data about the vendor, the CPU flags which help define its functional capabilities, the product version or name, and the current and maximum clock speeds. Most guest systems will show very little data from this command.

```
[root@david ~]# dmidecode -t 4
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 3.0.0 present.

Handle 0x0057, DMI type 4, 48 bytes
Processor Information
    Socket Designation: LGA 2066 R4
    Type: Central Processor
    Family: Xeon
    Manufacturer: Intel(R) Corporation
    ID: 54 06 05 00 FF FB EB BF
    Signature: Type 0, Family 6, Model 85, Stepping 4
    Flags:
            FPU (Floating-point unit on-chip)
            VME (Virtual mode extension)
            DE (Debugging extension)
            PSE (Page size extension)
            TSC (Time stamp counter)
            MSR (Model specific registers)
            PAE (Physical address extension)
            MCE (Machine check exception)
            CX8 (CMPXCHG8 instruction supported)
```

```
        APIC (On-chip APIC hardware supported)
        SEP (Fast system call)
        MTRR (Memory type range registers)
        PGE (Page global enable)
        MCA (Machine check architecture)
        CMOV (Conditional move instruction supported)
        PAT (Page attribute table)
        PSE-36 (36-bit page size extension)
        CLFSH (CLFLUSH instruction supported)
        DS (Debug store)
        ACPI (ACPI supported)
        MMX (MMX technology supported)
        FXSR (FXSAVE and FXSTOR instructions supported)
        SSE (Streaming SIMD extensions)
        SSE2 (Streaming SIMD extensions 2)
        SS (Self-snoop)
        HTT (Multi-threading)
        TM (Thermal monitor supported)
        PBE (Pending break enabled)
Version: Intel(R) Core(TM) i9-7960X CPU @ 2.80GHz
Voltage: 1.6 V
External Clock: 100 MHz
Max Speed: 4000 MHz
Current Speed: 2800 MHz
Status: Populated, Enabled
Upgrade: Other
L1 Cache Handle: 0x0054
L2 Cache Handle: 0x0055
L3 Cache Handle: 0x0056
Serial Number: Not Specified
Asset Tag: UNKNOWN
Part Number: Not Specified
Core Count: 16
Core Enabled: 16
Thread Count: 32
```

```
        Characteristics:
                64-bit capable
                Multi-Core
                Hardware Thread
                Execute Protection
                Enhanced Virtualization
                Power/Performance Control
```

Even on a physical host, some DMI types are empty. Notice that the **dmidecode** utility always prints the fact that it is obtaining its SMBIOS data from the /sysfs filesystem.

```
[root@david ~]# dmidecode -t 5
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 3.0.0 present.

[root@david ~]#
```

As you can see from the **dmidecode** man page, DMI type 16 contains data for the physical memory array. In the case of my main workstation, it is telling us that there are two physical arrays – sets – of four memory slots for a total of eight DIMM[3] slots.

The maximum capacity stated for each array is 1536GB for a total of 3072GB of RAM – 3TB. However, the official ASUS specification is for 8 DIMMs with a capacity of 128GB each for a total of 1024GB.

```
[root@david ~]# dmidecode -t 16
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 3.0.0 present.

Handle 0x0044, DMI type 16, 23 bytes
Physical Memory Array
        Location: System Board Or Motherboard
        Use: System Memory
        Error Correction Type: None
        Maximum Capacity: 1536 GB
        Error Information Handle: Not Provided
        Number Of Devices: 4
```

[3]Dual Inline Memory Module

```
Handle 0x004C, DMI type 16, 23 bytes
Physical Memory Array
        Location: System Board Or Motherboard
        Use: System Memory
        Error Correction Type: None
        Maximum Capacity: 1536 GB
        Error Information Handle: Not Provided
        Number Of Devices: 4
```

The ASUS web site[4] has a marketing description and pictures of this motherboard in case you want to see what it looks like.

Now let's look at the actual memory that is installed. DMI type 17 contains information about each memory slot, whether it is empty or specific data about the installed DIMM. I won't reproduce all eight sets of slot data here, just the first two which are then duplicated for the rest of the slots.

```
[root@david ~]# dmidecode -t 17
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 3.0.0 present.

Handle 0x0046, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x0044
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 16384 MB
        Form Factor: DIMM
        Set: None
        Locator: DIMM_A1
        Bank Locator: NODE 1
        Type: DDR4
        Type Detail: Synchronous
        Speed: 2133 MT/s
```

[4]ASUS, *TUF X299 Mark 2 Motherboard*, https://www.asus.com/us/Motherboards/TUF-X299-MARK-2/

```
        Manufacturer: Corsair
        Serial Number: 00000000
        Asset Tag:
        Part Number: CMK64GX4M4B3600C18
        Rank: 2
        Configured Memory Speed: 2133 MT/s
        Minimum Voltage: 1.2 V
        Maximum Voltage: 1.2 V
        Configured Voltage: 1.2 V

Handle 0x0048, DMI type 17, 40 bytes
Memory Device
        Array Handle: 0x0044
        Error Information Handle: Not Provided
        Total Width: Unknown
        Data Width: Unknown
        Size: No Module Installed
        Form Factor: DIMM
        Set: None
        Locator: DIMM_A2
        Bank Locator: NODE 1
        Type: Unknown
        Type Detail: Synchronous
        Speed: Unknown
        Manufacturer: NO DIMM
        Serial Number: NO DIMM
        Asset Tag:
        Part Number: NO DIMM
        Rank: Unknown
        Configured Memory Speed: Unknown
        Minimum Voltage: 1.2 V
        Maximum Voltage: 1.2 V
        Configured Voltage: 1.2 V
```

The bottom line for memory is that I have installed four 16GB DDR4[5] DIMMs, and there are four empty memory slots. You can also see the speed of 2133 MT/s (MegaTransfers per second), voltage specs, and a "Locator" which tells us which slot the DIMM is installed in.

We have not looked at all of the DMI types in this experiment that the **dmidecode** tool exposes to us. If you want to save some time and dump only the DMI types that actually have some usable data, you can use the following command.

```
[root@david ~]# dmidecode -q
```

You should explore those that we skipped. Of course the results will be more interesting on a physical host.

Have you figured out the limitations to DMI? It can be hard to see what is missing so let's continue.

# lshw

The **lshw** utility is similar in function to the **dmidecode** utility, but it produces output that is a bit more terse.

**EXPERIMENT 8-2**

Perform this experiment as the root user. Install the lshw package if it is not already.

```
[root@studentvm1 ~]# dnf install -y lshw
```

This program lists data about the motherboard, CPU, and other installed hardware. Run the following command to list the hardware on your host. Look through the data to see all of the (virtual) hardware in your VM. My personal workstation is more interesting, but I won't reproduce it all here. I just have enough so you can see some differences and the similarities.

First, note that the **lshw** utility shows the hostname. SMBIOS does not have that information because that data is scanned for long before the operating system startup sequence sets the hostname.

[5]Wikipedia, *Double data rate*, https://en.wikipedia.org/wiki/Double_data_rate. This article also contains a link to Transfer Rates, (MT/s).

```
[root@studentvm1 ~]# lshw | less
david
    description: Desktop Computer
    product: System Product Name (SKU)
    vendor: System manufacturer
    version: System Version
    serial: System Serial Number
    width: 64 bits
    capabilities: smbios-3.0.0 dmi-3.0.0 smp vsyscall32
    configuration: boot=normal chassis=desktop family=To be filled by O.E.M.
    sku=SKU uuid=801C1927-DAD7-DD11-
9360-B06EBF3A431F
```

The motherboard and memory information are essentially the same, but **lshw** has a nice summary of installed RAM.

```
  *-core
       description: Motherboard
       product: TUF X299 MARK 2
       vendor: ASUSTeK COMPUTER INC.
       physical id: 0
       version: Rev 1.xx
       serial: 170807951700403
       slot: Default string
     *-firmware
          description: BIOS
          vendor: American Megatrends Inc.
          physical id: 0
          version: 0503
          date: 07/11/2017
          size: 64KiB
          capacity: 16MiB
          capabilities: pci apm upgrade shadowing cdboot bootselect
          socketedrom edd int13floppy1200 int13flop
```

```
py720 int13floppy2880 int5printscreen int9keyboard int14serial int17printer
acpi usb biosbootspecification ue
fi
     *-memory
          description: System Memory
          physical id: 44
          slot: System board or motherboard
          size: 64GiB
        *-bank:0
             description: DIMM DDR4 Synchronous 2133 MHz (0.5 ns)
             product: CMK64GX4M4B3600C18
             vendor: Corsair
             physical id: 0
             serial: 00000000
             slot: DIMM_A1
             size: 16GiB
             width: 64 bits
             clock: 2133MHz (0.5ns)
        *-bank:1
             description: DIMM Synchronous [empty]
             product: NO DIMM
             vendor: NO DIMM
             physical id: 1
             serial: NO DIMM
             slot: DIMM_A2
        *-bank:2
             description: DIMM DDR4 Synchronous 2133 MHz (0.5 ns)
             product: CMK64GX4M4B3600C18
             vendor: Corsair
             physical id: 2
             serial: 00000000
             slot: DIMM_B1
             size: 16GiB
             width: 64 bits
             clock: 2133MHz (0.5ns)
<snip>
```

We also see external devices such as the keyboard

```
*-usb:0
     description: Keyboard
     product: Corsair Gaming K70 LUX RGB Keyboard
     vendor: Corsair
     physical id: 2
     bus info: usb@1:2
     version: 3.08
     serial: 1602B030AF0E98A8596A6476F5001BC6
     capabilities: usb-2.00
     configuration: driver=usbfs maxpower=500mA speed=12Mbit/s
```

<snip>

and attached printers that were not shown in the DMI database. I did see the printers on my physical workstation but not the one attached to the VM. So, you may see the printer on your VM or not.

```
*-usb:0 UNCLAIMED
     description: Printer
     product: MFC-9340CDW
     vendor: Brother Industries, Ltd
     physical id: 1
     bus info: usb@1:a.1
     version: 1.00
     serial: U63481A5J631227
     capabilities: usb-2.00 bidirectional
     configuration: maxpower=2mA speed=480Mbit/s
*-usb:1
     description: Printer
     product: HL-2270DW series
     vendor: Brother
     physical id: 3
     bus info: usb@1:a.3
     version: 1.00
     serial: C1J695917
     capabilities: usb-2.00 bidirectional
     configuration: driver=usblp maxpower=2mA speed=480Mbit/s
```

# lsusb

There are some commands available to list devices connected on the Universal Serial Bus[6] (USB). Let's start with **lsusb** – list USB devices – which lists devices on the USB bus, including USB hubs and the devices connected to them.

**EXPERIMENT 8-3**

Perform this experiment as the root user. You won't see much on the VM you are using for this course, just a couple virtual USB hubs and probably a printer if you have one attached to the VM. But here are the results from my workstation.

```
[root@david ~]# lsusb
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 006: ID 0bc2:ab1e Seagate RSS LLC Backup Plus Portable Drive
Bus 002 Device 003: ID 2109:0812 VIA Labs, Inc. VL812 Hub
Bus 002 Device 002: ID 2109:0812 VIA Labs, Inc. VL812 Hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 005: ID 2109:2812 VIA Labs, Inc. VL812 Hub
Bus 001 Device 003: ID 2109:2812 VIA Labs, Inc. VL812 Hub
Bus 001 Device 002: ID 1b1c:1b33 Corsair
Bus 001 Device 015: ID 1058:070a Western Digital Technologies, Inc. My
Passport Essential (WDBAAA), My Passport for Mac (WDBAAB), My Passport
Essential SE (WDBABM), My Passport SE for Mac (WDBABW
Bus 001 Device 014: ID 05e3:0745 Genesys Logic, Inc. Logilink CR0012
Bus 001 Device 012: ID 046d:c52b Logitech, Inc. Unifying Receiver
Bus 001 Device 010: ID 1a40:0201 Terminus Technology Inc. FE 2.1 7-port Hub
Bus 001 Device 013: ID 0424:4063 Standard Microsystems Corp.
Bus 001 Device 011: ID 0424:2640 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 008: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 006: ID 051d:0002 American Power Conversion Uninterruptible
Power Supply
Bus 001 Device 017: ID 04f9:0042 Brother Industries, Ltd HL-2270DW Laser Printer
```

[6]Wikipedia, *USB*, https://en.wikipedia.org/wiki/USB

```
Bus 001 Device 007: ID 04f9:02b0 Brother Industries, Ltd MFC-9340CDW
Bus 001 Device 004: ID 050d:0234 Belkin Components F5U234 USB 2.0 4-Port Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

You can see that there are multiple hubs here, some, like the root hubs, are internal and others like the Terminus Technology 7-port hub is an external add-on USB 3.0 hub. The APC UPS is listed as are both of my Brother laser printers.

You can also use the -t option to display the output in a tree format.

```
[root@david ~]# lsusb -t
/:  Bus 04.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/2p, 10000M
/:  Bus 03.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/2p, 480M
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/10p, 5000M
    |__ Port 8: Dev 2, If 0, Class=Hub, Driver=hub/4p, 5000M
        |__ Port 1: Dev 3, If 0, Class=Hub, Driver=hub/4p, 5000M
            |__ Port 2: Dev 6, If 0, Class=Mass Storage, Driver=uas, 5000M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/16p, 480M
    |__ Port 2: Dev 2, If 0, Class=Human Interface Device, Driver=usbfs, 12M
    |__ Port 2: Dev 2, If 1, Class=Human Interface Device, Driver=usbfs, 12M
    |__ Port 8: Dev 3, If 0, Class=Hub, Driver=hub/4p, 480M
        |__ Port 1: Dev 5, If 0, Class=Hub, Driver=hub/4p, 480M
    |__ Port 10: Dev 4, If 0, Class=Hub, Driver=hub/4p, 480M
        |__ Port 3: Dev 17, If 0, Class=Printer, Driver=usblp, 480M
        |__ Port 1: Dev 7, If 2, Class=Vendor Specific Class, Driver=, 480M
        |__ Port 1: Dev 7, If 0, Class=Printer, Driver=, 480M
        |__ Port 1: Dev 7, If 1, Class=Vendor Specific Class, Driver=, 480M
    |__ Port 11: Dev 6, If 0, Class=Human Interface Device, Driver=usbhid, 12M
    |__ Port 12: Dev 8, If 0, Class=Hub, Driver=hub/3p, 480M
        |__ Port 1: Dev 11, If 0, Class=Hub, Driver=hub/3p, 480M
            |__ Port 1: Dev 13, If 0, Class=Mass Storage, Driver=usb-storage, 480M
    |__ Port 14: Dev 10, If 0, Class=Hub, Driver=hub/7p, 480M
        |__ Port 1: Dev 12, If 2, Class=Human Interface Device, Driver=usbhid, 12M
        |__ Port 1: Dev 12, If 0, Class=Human Interface Device, Driver=usbhid, 12M
        |__ Port 1: Dev 12, If 1, Class=Human Interface Device, Driver=usbhid, 12M
        |__ Port 2: Dev 14, If 0, Class=Mass Storage, Driver=usb-storage, 480M
        |__ Port 4: Dev 15, If 0, Class=Mass Storage, Driver=usb-storage, 480M
```

The physical printer does show up on both my workstation and the VM in this experiment.

This view makes it easier to follow the device connections through multiple hubs and to see which root hub each device is ultimately connected to. This second syntax shows less information about the connected devices, but that can be cross-referenced with the first syntax using the bus, port, and device numbers.

# usb-devices

The **usb-devices** utility performs the same task as **lsusb**, that of listing devices attached to the bus, but it can provide more information about each device.

**EXPERIMENT 8-4**

Perform this experiment as root. You will only get two entries – or three if you have a physical printer attached – so I have reproduced some of the output from my workstation because it has some interesting entries you won't see on your VM.

```
[root@david ~]# usb-devices

T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480 MxCh=16
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0002 Rev=04.20
S:  Manufacturer=Linux 4.20.14-200.fc29.x86_64 xhci-hcd
S:  Product=xHCI Host Controller
S:  SerialNumber=0000:00:14.0
C:  #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=0mA
I:  If#=0x0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub

T:  Bus=01 Lev=01 Prnt=01 Port=09 Cnt=01 Dev#=  4 Spd=480 MxCh= 4
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=02 MxPS=64 #Cfgs=  1
P:  Vendor=050d ProdID=0234 Rev=00.00
C:  #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=2mA
I:  If#=0x0 Alt= 1 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=02 Driver=hub
```

```
<snip>

T:  Bus=01 Lev=02 Prnt=04 Port=02 Cnt=02 Dev#= 17 Spd=480 MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=04f9 ProdID=0042 Rev=01.00
S:  Manufacturer=Brother
S:  Product=HL-2270DW series
S:  SerialNumber=C1J695917
C:  #Ifs= 1 Cfg#= 1 Atr=c0 MxPwr=2mA
I:  If#=0x0 Alt= 0 #EPs= 2 Cls=07(print) Sub=01 Prot=02 Driver=usblp

T:  Bus=01 Lev=01 Prnt=01 Port=10 Cnt=02 Dev#=  6 Spd=12  MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=051d ProdID=0002 Rev=00.90
S:  Manufacturer=American Power Conversion
S:  Product=Back-UPS XS 1500G FW:866.L8 .D USB FW:L8
S:  SerialNumber=3B1551X04045
C:  #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=2mA
I:  If#=0x0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=00 Prot=00 Driver=usbhid

<snip>

T:  Bus=01 Lev=03 Prnt=11 Port=00 Cnt=01 Dev#= 13 Spd=480 MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=0424 ProdID=4063 Rev=01.91
S:  Manufacturer=Generic
S:  Product=Ultra Fast Media Reader
S:  SerialNumber=000000264001
C:  #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=96mA
I:  If#=0x0 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage

<snip>

T:  Bus=01 Lev=01 Prnt=01 Port=01 Cnt=05 Dev#=  2 Spd=12  MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=1b1c ProdID=1b33 Rev=03.08
S:  Manufacturer=Corsair
S:  Product=Corsair Gaming K70 LUX RGB Keyboard
S:  SerialNumber=1602B030AF0E98A8596A6476F5001BC6
```

```
C:  #Ifs= 2 Cfg#= 1 Atr=a0 MxPwr=500mA
I:  If#=0x0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=01 Driver=usbfs
I:  If#=0x1 Alt= 0 #EPs= 2 Cls=03(HID  ) Sub=00 Prot=00 Driver=usbfs
```

I have removed the entries for most of the USB hubs for my workstation, but this data shows the physical devices such as external USB hard drives, a media reader, and the keyboard.

# lspci

The **lspci** utility lists devices on the Peripheral Component Interconnect[7] (PCI) bus and its extensions, PCI-X, and PCI Express (PCIe). This includes many motherboard devices such as memory and bus controllers, as well as devices like audio, Ethernet, SATA, and video devices.

**EXPERIMENT 8-5**

Perform this experiment as root. Without any options, the **lspci** command provides a list of PCI hardware that starts with the PCI bus address of each device and a simple description. This results in a little more interesting output on the VM, but I have also reproduced a shortened list from my workstation.

```
[root@david ~]# lspci
00:00.0 Host bridge: Intel Corporation Sky Lake-E DMI3 Registers (rev 04)
00:04.0 System peripheral: Intel Corporation Sky Lake-E CBDMA Registers (rev 04)
00:04.1 System peripheral: Intel Corporation Sky Lake-E CBDMA Registers (rev 04)
<snip>
00:04.7 System peripheral: Intel Corporation Sky Lake-E CBDMA Registers (rev 04)
00:05.0 System peripheral: Intel Corporation Sky Lake-E MM/Vt-d Configuration
        Registers (rev 04)
00:05.2 System peripheral: Intel Corporation Sky Lake-E RAS (rev 04)
00:05.4 PIC: Intel Corporation Sky Lake-E IOAPIC (rev 04)
00:08.0 System peripheral: Intel Corporation Sky Lake-E Ubox Registers (rev 04)
```

[7]Wikipedia, *Peripheral Component Interconnect*, https://en.wikipedia.org/wiki/Conventional_PCI

```
00:08.1 Performance counters: Intel Corporation Sky Lake-E Ubox Registers (rev 04)
00:08.2 System peripheral: Intel Corporation Sky Lake-E Ubox Registers (rev 04)
00:14.0 USB controller: Intel Corporation 200 Series/Z370 Chipset Family USB 3.0
        xHCI Controller
00:14.2 Signal processing controller: Intel Corporation 200 Series PCH Thermal
        Subsystem
00:16.0 Communication controller: Intel Corporation 200 Series PCH CSME HECI #1
00:17.0 SATA controller: Intel Corporation 200 Series PCH SATA controller [AHCI mode]
00:1c.0 PCI bridge: Intel Corporation 200 Series PCH PCI Express Root Port #1 (rev f0)
00:1c.2 PCI bridge: Intel Corporation 200 Series PCH PCI Express Root Port #3 (rev f0)
00:1c.4 PCI bridge: Intel Corporation 200 Series PCH PCI Express Root Port #5 (rev f0)
00:1f.0 ISA bridge: Intel Corporation X299 Chipset LPC/eSPI Controller
00:1f.2 Memory controller: Intel Corporation 200 Series/Z370 Chipset Family Power
        Management Controller
00:1f.3 Audio device: Intel Corporation 200 Series PCH HD Audio
00:1f.4 SMBus: Intel Corporation 200 Series/Z370 Chipset Family SMBus Controller
00:1f.6 Ethernet controller: Intel Corporation Ethernet Connection (2) I219-V
02:00.0 SATA controller: Marvell Technology Group Ltd. Device 9215 (rev 11)
03:00.0 USB controller: ASMedia Technology Inc. ASM2142 USB 3.1 Host Controller
16:05.0 System peripheral: Intel Corporation Sky Lake-E VT-d (rev 04)
16:05.2 System peripheral: Intel Corporation Sky Lake-E RAS Configuration
        Registers (rev 04)
16:05.4 PIC: Intel Corporation Sky Lake-E IOxAPIC Configuration Registers (rev 04)
16:08.0 System peripheral: Intel Corporation Sky Lake-E CHA Registers (rev 04)
16:08.1 System peripheral: Intel Corporation Sky Lake-E CHA Registers (rev 04)
<snip>
64:0d.2 System peripheral: Intel Corporation Sky Lake-E LMS Channel 2 (rev 04)
64:0d.3 System peripheral: Intel Corporation Sky Lake-E LMDP Channel 2 (rev 04)
65:00.0 VGA compatible controller: Advanced Micro Devices, Inc. [AMD/ATI] Barts XT
        [Radeon HD 6870]
65:00.1 Audio device: Advanced Micro Devices, Inc. [AMD/ATI] Barts HDMI Audio
        [Radeon HD 6790/6850/6870 / 7720 OEM]
<snip>
```

Using the -v option for verbose output produces several lines of data for each device. The total output for this command is very long so you should pipe it through the **less** paging tool. I have only reproduced a few stanzas from my workstation in order to save space.

```
[root@studentvm1 ~]# lspci -v | less
```

```
0:00.0 Host bridge: Intel Corporation Sky Lake-E DMI3 Registers (rev 04)
        Subsystem: ASUSTeK Computer Inc. Device 873c
        Flags: fast devsel, NUMA node 0
        Capabilities: [90] Express Root Port (Slot-), MSI 00
        Capabilities: [e0] Power Management version 3
        Capabilities: [100] Vendor Specific Information: ID=0002 Rev=0 Len=00c <?>
        Capabilities: [144] Vendor Specific Information: ID=0004 Rev=1 Len=03c <?>
        Capabilities: [1d0] Vendor Specific Information: ID=0003 Rev=1 Len=00a <?>
        Capabilities: [250] Secondary PCI Express <?>
        Capabilities: [280] Vendor Specific Information: ID=0005 Rev=3 Len=018 <?>
        Capabilities: [300] Vendor Specific Information: ID=0008 Rev=0 Len=038 <?>
<snip>
00:17.0 SATA controller: Intel Corporation 200 Series PCH SATA controller
        [AHCI mode] (prog-if 01 [AHCI 1.0])
        Subsystem: ASUSTeK Computer Inc. Device 873c
        Flags: bus master, 66MHz, medium devsel, latency 0, IRQ 29, NUMA node 0
        Memory at 92f68000 (32-bit, non-prefetchable) [size=8K]
        Memory at 92f6c000 (32-bit, non-prefetchable) [size=256]
        I/O ports at 3050 [size=8]
        I/O ports at 3040 [size=4]
        I/O ports at 3020 [size=32]
        Memory at 92f6b000 (32-bit, non-prefetchable) [size=2K]
        Capabilities: [80] MSI: Enable+ Count=1/1 Maskable- 64bit-
        Capabilities: [70] Power Management version 3
        Capabilities: [a8] SATA HBA v1.0
        Kernel driver in use: ahci
<snip>
00:1f.0 ISA bridge: Intel Corporation X299 Chipset LPC/eSPI Controller
        Subsystem: ASUSTeK Computer Inc. Device 873c
        Flags: bus master, medium devsel, latency 0, NUMA node 0

00:1f.2 Memory controller: Intel Corporation 200 Series/Z370 Chipset Family
        Power Management Controller
        Subsystem: ASUSTeK Computer Inc. Device 873c
        Flags: fast devsel, NUMA node 0
        Memory at 92f44000 (32-bit, non-prefetchable) [disabled] [size=16K]
<snip>
65:00.0 VGA compatible controller: Advanced Micro Devices, Inc. [AMD/ATI]
        Barts XT [Radeon HD 6870] (prog-if
```

```
00 [VGA controller])
        Subsystem: Gigabyte Technology Co., Ltd Device 21fa
        Flags: bus master, fast devsel, latency 0, IRQ 42, NUMA node 0
        Memory at c0000000 (64-bit, prefetchable) [size=256M]
        Memory at d8e20000 (64-bit, non-prefetchable) [size=128K]
        I/O ports at b000 [size=256]
        Expansion ROM at 000c0000 [disabled] [size=128K]
        Capabilities: [50] Power Management version 3
        Capabilities: [58] Express Legacy Endpoint, MSI 00
        Capabilities: [a0] MSI: Enable+ Count=1/1 Maskable- 64bit+
        Capabilities: [100] Vendor Specific Information: ID=0001 Rev=1 Len=010 <?>
        Capabilities: [150] Advanced Error Reporting
        Kernel driver in use: radeon
        Kernel modules: radeon
<snip>
```

Now try this to get a tree view.

```
[root@studentvm1 ~]# lspci -tv
```

The motherboard chip set information provided by the **lspci** tool can be used to determine memory and CPU compatibility. It could also be used to help determine whether to purchase a used computer or explore a new one without having to take it apart.

# Cleanup

Let's clean up a little. We no longer need the physical printer so we can disable it and remove it from the VM.

### CLEANUP

Perform this cleanup as root. First we disable the printer. Be sure to use the correct printer name for your physical printer.

```
[root@studentvm1 spool]# cupsdisable Brother-HL-2270DW
```

Refer to Figure 7-1 if necessary and remove the check mark from the printer so that it is no longer associated with the VM.

## Chapter summary

The commands we have explored in this chapter can provide an easy way to examine the hardware components of any computer. It is possible to discover information about a computer that cannot be obtained even by taking it apart. If the computer does not have Linux installed, just boot to a live Linux USB thumb drive and use these tools to examine the hardware in detail not available in any other way. I have used these tools as part of a first look when people have asked me to "fix" their computers or when making a determination of whether I wanted to purchase a specific model that was on display at the local computer store.

All of this information is available using the /proc and /sys filesystems, but these tools make it easier because you don't need to search for it.

## Exercises

Perform the following exercises to complete this chapter:

1. What is the motherboard serial number of your VM?
2. If you have root access to a physical Linux host, what is the motherboard serial number?
3. What type of device is reported by DMI type 5?
4. Why do you think that external devices like printers, pointing devices, and keyboards show up in the `lshw` data but are not in the DMI database?
5. What other devices are missing from the DMI database that appear in the **`lshw`** output?
6. What make and model Ethernet controller is virtualized for your VM?
7. What happens to the queue for the physical printer if you disconnect the printer from the VM without first having disabled the queue?
8. What happens if a print job is then sent to an enabled print queue that has no printer?

CHAPTER 9

# Command-Line Programming

## Objectives

In this chapter you will learn

- A definition for command-line programs
- To create Bash command-line programs
- To use logical comparisons to alter the execution path of command-line programs
- To use "for" loops to iterate over a code segment a specified list of items
- To use "while" and "until" to loop over a portion of code a specified number of times.

## Introduction

We have already used command-line programs in earlier chapters of this course. Those were all fairly simple and straightforward with simple requirements. There are many times when SysAdmins create simple command-line programs to perform a series of tasks. They are a common tool and can save time and effort.

My personal objective when writing CLI programs is to save time and to "be the lazy SysAdmin." CLI programs allow me to accomplish this by listing several commands in a specific sequence so that they will execute one after another. As a result, I do not need to watch the progress of one command and, when it has finished, type in the next




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_9

command. I can go do other things and not be concerned about having to continually monitor the progress of each command.

It is not possible to include a complete training course on Bash command-line programming and shell scripts in this course which is already quite lengthy. This chapter and Chapter 10 are intended as an introduction to these concepts and many of the tools available for Bash programming. There are numerous books and courses available that do provide in-depth instruction on Bash usage and programming and, if you are interested, you should use one of those to learn more.

Also, I do not set up some sort of bogus objective to provide a framework for building an application that you will never use. I find that my best learning experiences are when I am working on my own projects. My objective in this chapter is to introduce you to many of the typical forms and structures used in Bash command-line programming and scripts. When you encounter a task that requires CLI programming or a script, you will likely remember that you have seen a method for accomplishing that task and at least know where to start looking for details.

One excellent source is the Bash Manual[1] which can be found at `www.gnu.org` and which is sponsored by the Free Software Foundation (FSF) at `www.fsf.org/`. This free manual is available in many formats and can be downloaded or browsed online.

# Definition of a program

Let's define what we mean by a program. The Free On-line Dictionary of Computing[2] (FOLDOC) defines a program as "The instructions executed by a computer, as opposed to the physical device on which they run." Other sources such as Princeton University's WordNet[3] define a program as "…a sequence of instructions that a computer can interpret and execute…." Wikipedia also has a good article about computer programs.[4]

[1]GNU `https://www.gnu.org/software/bash/manual/`

[2]You can install the "dict" package and then use the command `dict <word>` to look up any word – without the <>. Results are presented from one or more of several online dictionaries including the FOLDOC. Or you can go to `http://foldoc.org/` to look up computing terms. I find it helpful to use the dict command which enables me to see multiple definitions for many terms.

[3]WordNet, `https://wordnet.princeton.edu/`

[4]Wikipedia, *Computer Program*, `https://en.wikipedia.org/wiki/Computer_program`

Based on these definitions, a program can consist of one or more instructions that perform a specific related task. A computer program instruction is also called a program statement. For SysAdmins, a program is usually a sequence of shell commands. All of the shells available for Linux, at least the ones with which I am familiar, have at least some basic form of programming capability, and Bash, the default shell for most Linux distributions, is no exception. This chapter uses Bash because it is so ubiquitous. You may already prefer or later learn about and come to prefer a different shell. If so, the programming concepts will be the same though the constructs and syntax may differ somewhat. Some shells may support some features that others do not. But they all provide some programming capability.

These shell programs can be stored in a file for repeated use, or they may be simply created on an ad hoc basis at the command line as needed. In this chapter we will start working directly at the command line. In Chapter 10, we will discuss storing our simple programs in files for sharing and reuse and more complex and lengthy programs.

# Simple CLI programs

The simplest command-line programs are one or two consecutive program statements, which may be related or not, that are entered on the command line before the Enter key is pressed. For example, the second statement, if there is one, may be dependent upon the actions of the first but it does not need to be.

There is also one bit of syntactical punctuation that needs to be clearly stated. When entering a single command on the command line, pressing the Enter key terminates the command with an implicit semicolon (;). When used in a CLI shell program entered on the command line, the semicolon must be used to terminate each statement and separate it from the next one. The last statement in a CLI shell program can use an explicit or implicit semicolon.

## Some basic syntax

Let's look at a couple examples which will clarify this syntax. All of the experiments in this chapter should be performed as the student user.

## EXPERIMENT 9-1

For now we will use the explicit semicolon to terminate our program statements. The **echo** command is used to print data to the screen. This terminology is a legacy of the old teletype machines when all of the output from a program was printed to a roll of paper.

First ensure that we are in the student user's home directory. Then enter the following program.

```
[student@studentvm1 ~]$ echo "Hello world." ;
Hello world.
```

That may not seem like much of a program, but it is the same first program I have encountered with every new programming language I have ever learned. The syntax may be a bit different for each language but the result is the same.

Let's expand on this trivial but ubiquitous program a little.

```
[student@studentvm1 ~]$ echo "Hello world." ; ls ;
Hello world.
chapter25   cpuHog.Linux  dmesg2.txt  Downloads  newfile.txt  softlink1  testdir6
chapter26   cpuHog.mac    dmesg3.txt  file005    Pictures     Templates  testdir7
cpuHog      Desktop       dmesg.txt   link3      Public       testdir    Videos
cpuHog.dos  dmesg1.txt    Documents   Music      random.txt   testdir1
```

OK, that is interesting but the initial statement is not very meaningful in this context so let's change it a bit.

```
[student@studentvm1 ~]$ echo "My home directory." ; ls ;
My home directory.
chapter25   cpuHog.Linux  dmesg2.txt  Downloads  newfile.txt  softlink1  testdir6
chapter26   cpuHog.mac    dmesg3.txt  file005    Pictures     Templates  testdir7
cpuHog      Desktop       dmesg.txt   link3      Public       testdir    Videos
cpuHog.dos  dmesg1.txt    Documents   Music      random.txt   testdir1
```

That makes a bit more sense. The results are related, but the individual program statements are independent of each other. Notice that I like spaces before and after the semicolon which makes the code a bit easier to read. Try this little CLI program again without an explicit semicolon at the end.

```
[student@studentvm1 ~]$ echo "My home directory." ; ls
```

There is no difference in the output data.

## Output to the display

Many CLI programs are intended to produce output of some type. The **echo** command is commonly used in a simple manner as seen in Experiment 9-1. We can use the -e option to enable escape codes that allow us to do a little more complex output. For example, we might want to print multiple lines of output. It is not necessary to use multiple **echo** commands to do this.

### EXPERIMENT 9-2

Enter and run the following single statement program.

```
[student@studentvm1 ~]$ echo "Twinkle, twinkle, little star How I wonder what
you are Up above the world so high Like a diamond in the sky" ;
Twinkle, twinkle, little star How I wonder what you are Up above the world so
high Like a diamond in the sky
```

That is not easy to read so we could break it up like this into four separate **echo** statements.

```
[student@studentvm1 ~]$ echo "Twinkle, twinkle, little star" ; echo "How
I wonder what you are" ; echo "Up above the world so high" ; echo "Like a
diamond in the sky" ;
Twinkle, twinkle, little star
How I wonder what you are
Up above the world so high
Like a diamond in the sky
```

But this next method is, if not easier, at least cleaner.

```
[student@studentvm1 ~]$ echo -e "Twinkle, twinkle, little star\nHow I wonder
what you are\nUp above the world so high\nLike a diamond in the sky\n" ;
Twinkle, twinkle, little star
How I wonder what you are
Up above the world so high
Like a diamond in the sky
```

This method of using a single echo command to print multiple lines can save typing as well so that we can be the lazy SysAdmin. In this case, 19 characters fewer are required. It not only saves typing, it also saves disk space, memory space, and CPU time. One command executes faster than four. I know that these are all relatively cheap commodities these days, but many of us inherit or are gifted with perfectly good, old computers whose specifications are not nearly as generous as newer ones we might purchase.

In my book *The Linux Philosophy for SysAdmins*[5] I devote a whole chapter to simplicity. One command is always simpler than four commands.

The **printf** command (print formatted) provides even more capabilities because it allows more complex data formatting. It can do the same thing as we did previously too because it recognizes the same escape codes, but without needing an option.

```
[student@studentvm1 ~]$ printf "Twinkle, twinkle, little star\nHow I wonder
what you are\nUp above the world so high\nLike a diamond in the sky\n" ;
Twinkle, twinkle, little star
How I wonder what you are
Up above the world so high
Like a diamond in the sky
```

The printf command can also use all of the C language printf format specifications so that very complex formats such as numerical field widths, decimal precision, and locale formatting can be specified. Those capabilities are beyond the scope of this course, but details can be found using **man 3 printf**.

The **echo** man page contains a complete list of the escape codes that it recognizes.

[5]Both, David, *The Linux Philosophy for SysAdmins*, Apress, 2018, Chapter 18.

## Something about variables

Like all programming languages, the Bash shell can deal with variables. A variable is a symbolic name that refers to a specific location in memory which contains a value of some sort. The value of a variable is changeable, that is, it is variable.

Bash does not type variables like C and related languages, defining them as integers, floating point, or string types. In Bash, all variables are strings. A string that is an integer can be used in integer arithmetic which is all that Bash has the capability of doing. If more complex maths are required, the **bc** command can be used in CLI programs and scripts.

Variables are assigned values and can then be used to refer to this values in CLI programs and scripts. The value of a variable is set using its name but not preceded by a $ sign. The assignment **VAR=10** sets the value of the variable VAR to 10. To print the value of the variable, we can use the statement **echo $VAR**. Let's start with text variables, that is, non-numeric.

Bash variables become part of the shell environment until they are unset.

### EXPERIMENT 9-3

To start, we check the initial value of a variable that has not been assigned. It should be null. Then we will assign a value to the variable and print it to verify its value. We will do all of this in a single CLI program.

---

**Note** The syntax of variable assignment is very strict. There must be no spaces on either side of the equal (=) sign in the assignment statement.

---

```
[student@studentvm1 ~]$ echo $MyVar ; MyVar="Hello World" ; echo $MyVar ;

Hello World
```

The empty line indicates that the initial value of MyVar is null. Changing the value of a variable is the same as setting it in the first place. In this example, we can see both the original and the new value.

```
[student@studentvm1 ~]$ echo $MyVar ; MyVar="Hello World; Welcome to Linux" ;
echo $MyVar ;
Hello World
Hello World; Welcome to Linux
```

Variables can also be "unset" and returned to a null value.

```
[student@studentvm1 ~]$ unset MyVar ; echo $MyVar ;

[student@studentvm1 ~]$
```

Text string variables can also be combined in various ways.

```
[student@studentvm1 ~]$ Var1="Hello World!" ; Var2="Welcome to Bash CLI
programming." ; printf "$Var1\n$Var2\n" ;
Hello World!
Welcome to Bash CLI programming.
```

Note that these variables remain set in the shell environment until they are unset as we did with $MyVar.

---

**Note** I will no longer use the explicit semicolon (;) at the end of a command-line program and will rely on the one implied by pressing the Enter key at the end of the line.

---

```
[student@studentvm1 ~]$ echo "$Var1 $Var2"
Hello World! Welcome to Bash CLI programming.
[student@studentvm1 ~]$ set | grep Var
Var1='Hello World!'
Var2='Welcome to Bash CLI programming.'
```

Reliance upon specific variables to already be set in the shell environment from one instance of running a CLI program to the next is poor practice. It is usually a best practice to set those variables needed by a program within the program itself – unless the intent is to check the current value of a shell variable.

Now let's look at doing a bit of math in Bash. As I have mentioned, Bash can only perform integer math but that can be sufficient for many purposes. In this little program, we reassign $Var1 and $Var2 and then use them in a Bash integer calculation.

```
[student@studentvm1 ~]$ Var1="7" ; Var2="9" ; echo "Result = $((Var1*Var2))"
Result = 63
```

What happens when we perform a math operation that results in a floating-point number?

```
[student@studentvm1 ~]$ Var1="7" ; Var2="9" ; echo "Result = $((Var1/Var2))"
Result = 0
[student@studentvm1 ~]$ Var1="7" ; Var2="9" ; echo "Result = $((Var2/Var1))"
Result = 1
[student@studentvm1 ~]$
```

The result is the nearest integer. Be sure to notice that the calculation was performed as part of the echo statement; no intermediate result is required, but we could have done something like the following if we wanted to keep the result for use more than once in a later part of the program.

```
[student@studentvm1 ~]$ Var1="7" ; Var2="9" ; Result=$((Var1*Var2)) ; echo
"Result = $Result"
Result = 63
```

Use variables wherever possible in CLI programs instead of hard-coded values. Even if you think you will only use a particular value once, such as a directory name or a file name, create a variable and use the variable where you would have placed the hard-coded name.

# Control operators

Shell control operators are one of the syntactical operators that allow us to create some interesting command-line programs. We have seen that the simplest form of CLI program is just stringing several commands together in a sequence on the command line.

```
command1 ; command2 ; command3 ; command4 ; . . . ; etc. ;
```

Those commands will all run without a problem so long as no errors occur. But what happens when an error occurs? We can anticipate and allow for errors using the && and || built-in bash control operators. These two control operators provide us with some flow control and enable us to alter the sequence of code execution. The semicolon is also considered to be a bash control operator as is the newline character.

The && operator simply says that if command1 is successful then run command2. If command1 fails for any reason, then command2 is skipped. That syntax looks like this.

```
command1 && command2
```

## Return codes

The `command1 && command2` syntax works because every command sends a return code (RC) to the shell that indicates whether it completed successfully or whether there was some type of failure during execution. By convention, a return code of zero (0) indicates success and any positive number indicates some type of failure. Some of the tools we use as SysAdmins return only a one (1) to indicate a failure, but many can return other codes as well to further indicate the type of failure that occurred.

The bash shell has a variable $? which contains the return code from the last command. This return code can be checked very easily by a script, the next command in a list of commands, or even us SysAdmins.

### EXPERIMENT 9-4

Let's start by looking at return codes. We can run a simple command and then immediately check the return code. The return code will always be for the last command that was run before we look at it.

```
[student@studentvm1 ~]$ ll ; echo "RC = $?"
total 1264
drwxrwxr-x  2 student student   4096 Mar  2 08:21 chapter25
drwxrwxr-x  2 student student   4096 Mar 21 15:27 chapter26
-rwxr-xr-x  1 student student     92 Mar 20 15:53 cpuHog
<snip>
drwxrwxr-x. 2 student student 663552 Feb 21 14:12 testdir7
drwxr-xr-x. 2 student student   4096 Dec 22 13:15 Videos
RC = 0
[student@studentvm1 ~]$
```

The return code (RC) in this case is zero (0) which means the command completed successfully. Now try the same command on a directory for which we do not have permissions, root's home directory.

```
[student@studentvm1 ~]$ ll /root ; echo "RC = $?"
ls: cannot open directory '/root': Permission denied
RC = 2
[student@studentvm1 ~]$
```

In this case the return code is 2 which specifically means that permission was denied for a non-root user to access a directory to which the user is not permitted access. The control operators use these return codes to enable us to alter the sequence of program execution.

## The operators

Let's try the control operators as they might be used in a command-line program. In this simple case, we want to create a new directory and add a new, empty file to it.

### EXPERIMENT 9-5

We start with something simple. Our objective is to create a new directory and create a file in it. We only want to do this if the directory has been created successfully.

```
[student@studentvm1 ~]$ Dir=chapter9 ; mkdir $Dir && touch $Dir/testfile1 ; ll $Dir
total 0
-rw-rw-r-- 1 student student 0 Mar 29 15:18 testfile1
```

Everything worked as it should because the chapter9 directory is accessible and writable. Change the permissions on chapter9 so it is no longer accessible to the student user.

```
[student@studentvm1 ~]$ Dir=chapter9 ; chmod 076 $Dir ; ls -l | grep chapter9
d---rwxrw-  2 student student   4096 Mar 29 15:18 chapter9
[student@studentvm1 ~]$
```

You can see from the listing that the user student no longer has any access to the chapter9 directory. Now let's run some commands that will create a new directory in the chapter9 directory and then – if the new directory creation is successful – it will create a new file in that subdirectory. This time we will also use the && control operator.

```
[student@studentvm1 ~]$ Dir=chapter9 ; mkdir $Dir/subdirectory && touch $Dir/
subdirectory/testfile
mkdir: cannot create directory ‘chapter9/subdirectory’: Permission denied
[student@studentvm1 ~]$
```

The error seen in the preceding example was emitted by the **mkdir** command. We did not receive an error indicating that the file could not be created because creation of the directory failed. The && control operator sensed the non-zero return code so the **touch** command was skipped. Using the && control operator prevents the **touch** command from running because there was an error in creating the directory. This type of command-line program flow control can prevent errors from compounding and making a real mess of things. But let's get a little more complicated.

The || control operator allows us to add another program statement that executes when the initial program statement returns a code greater than zero. The basic syntax looks like this.

```
command1 || command2
```

This syntax reads, "If command1 fails, execute command2." That implies that if command1 succeeds, command2 is skipped. Let's try this with our attempt to create a new directory.

```
[student@studentvm1 ~]$ Dir=chapter9 ; mkdir $Dir/subdirectory || echo "New
file was not created."
mkdir: cannot create directory ‘chapter9/subdirectory’: Permission denied
New file was not created.
[student@studentvm1 ~]$
```

This is exactly what we expected. Because the new directory could not be created, the first command failed which resulted in execution of the second command.

Combining these two operators gives us the best of both. Our control operator syntax using some flow control now takes this general form when we use the && and || control operators.

```
preceding commands ; command1 && command2 || command3 ; following commands
```

This syntax can be stated like so: if command1 exits with a return code of 0 then execute command2, otherwise execute command3. Let's try it.

```
[student@studentvm1 ~]$ Dir=chapter9 ; mkdir $Dir/subdirectory && touch $Dir/
subdirectory/testfile || echo "New file was not created."
mkdir: cannot create directory ‘chapter9/subdirectory’: Permission denied
New file was not created.
[student@studentvm1 ~]$
```

Now reset the permissions on ~/chapter9 to 755 and try this last command again.

```
[student@studentvm1 ~]$ chmod 755 chapter9
[student@studentvm1 ~]$ Dir=chapter9 ; mkdir $Dir/subdirectory && touch $Dir/
subdirectory/testfile || echo "New file was not created."
[student@studentvm1 ~]$ ll $Dir/subdirectory/
total 0
-rw-rw-r-- 1 student student 0 Mar 29 15:40 testfile
[student@studentvm1 ~]$
```

The program using the control operators may be preceded and followed by other commands that can be related to the ones in the flow control section but which are unaffected by the flow control. All of the preceding and following commands will execute without regard to anything that takes place inside the flow control command.

These control operators provide us some interesting and powerful capabilities for doing program flow control on the command line. These Bash control operators can also be used in scripts. I use these operators quite frequently both on the command line and in scripts because – automate everything.

Commands that use these operators for flow control are also called compound commands.

# Program flow control

Every programming language I have ever used has some form of an "if" flow control structure and Bash is no exception. The Bash **if** statement can be used to test whether a condition is true or false in order to determine which execution path to take. This flow control structure is flexible and can be adjusted to meet needs ranging from very simple to quite complex.

We will start with some simple examples and move up in complexity. However, this structure can be less suitable for use in CLI programs as it becomes more complex. So we will keep things fairly simple here.

The logical syntax for this control statement is

```
if condition1; then list1; [ elif condition2; then list2; ] ... [ else
list3; ] fi
```

where "condition" is a list of one or more conditions to be tested and "list" is a list of shell program statements that are to be executed if the condition is true. The elif and else phrases in this construct are optional. I like to read this syntax as "If condition1 is true, then execute the statements in list1, or else if condition2 is true, then execute the statements in list2, or else execute the statements in list3."

The fi statement at the end of the if control structure provides an explicit syntactical ending to the if statement. It is not optional.

Regardless of how many conditional expressions are contained in an if-elif... else compound command, only one list of program statements is executed, the ones associated with the first conditional to return "true."

## true and false

Before we get further into the subject of flow control, we need to explicitly define some things that were not always made clear when I was learning Unix and Linux.

First, there is always only one return code that indicates "true" or "success." Zero (0) is always returned when a command or program completes successfully. Any positive integer when used as a return code represents an error, or a failure, with the specific number representing a particular type of error. For logical operations zero (0) is always true and one (1) is always false.

And of course, because Linux is so amazingly cool, it has two commands that can be used for testing or for obtaining specific true or false return code results in a CLI program or a script. What do you think they are? Naturally, they are **`true`** and **`false`**. The **`true`** command always generates a return code of zero (0), which is true, and the **`false`** command always generates a return code of one (1), which is false.

### EXPERIMENT 9-6

Let's just look at the return codes from the **`true`** and **`false`** commands. Remember that the shell variable $? always contains the return code from the last command that was run.

```
[dboth@david trunk]$ true ; echo $?
0
[dboth@david trunk]$ false ; echo $?
1
```

Now let's use **true** and **false** with the control operators with which we are already familiar.

```
[dboth@david trunk]$ true && echo "True" || echo "False"
True
[dboth@david trunk]$ false && echo "True" || echo "False"
False
```

I frequently use these two simple commands to test the logic of a complex command with control operators when I absolutely need to know whether the RC was true or false.

## Logical operators

Bash has a large set of logical operators that can be used in conditional expressions. The most basic form of the **if** control structure is to test for a condition and then execute a list of program statements if the condition is true. There are three types of operators, file, numeric, and non-numeric operators. Each operator returns true (0) as its return code if the condition is met and false (1) if the condition is not met.

The operators and their descriptions are listed in the following section and are taken from the Bash man page. I have added some additional explanations in some instances to help clarify the meanings.

### Syntax

The functional syntax of these comparison operators is one or two arguments with an operator that are placed within square braces. Then we have a list of program statements that are executed if the condition is true, and an optional list of program statements if the condition is false.

```
if [ arg1 operator arg2 ] ; then list
```

or

```
if [ arg1 operator arg2 ] ; then list ; else list ; fi
```

The spaces in the comparison are required as shown. The single square braces, [ and ], are the traditional Bash symbols that are equivalent to the **test** command.

```
if test arg1 operator arg2 ; then list
```

There is also a more recent syntax that offers a few advantages and which some SysAdmins prefer. This format is a bit less compatible with different versions of Bash and other shells such as ksh (Korn shell).

```
if [[ arg1 operator arg2 ]] ; then list
```

## File operators

File operators provide easy determination of various file tests. For example, these tests can determine whether a file exists and is empty or contains some data, whether it is a regular file, a link, or a directory. These operators can also be used to detect various attributes such as user and group ID (ownership) and file permissions.

Figure 9-1 lists these file operators.

| Operator | Description |
|---|---|
| -a filename | True if the file exists. It can be empty or have some content but so long as it exists this will be true. |
| -b filename | True if file exists and is a block special file such as a hard drive like /dev/sda or /dev/sda1. |
| -c filename | True if the file exists and is a character special file such as a TTY device like /dev/TTY1. |
| -d filename | This is true if the file exists and is a directory. |
| -e filename | True if file exists. This is the same as -a, above. |
| -f filename | True if the file exists and is a regular file as opposed to a directory a device special file or a link, among others. |
| -g filename | True if the file exists and is set-group-id, SETGID. |
| -h filename | This is true if file exists and is a symbolic link. |
| -k filename | True if file exists and its ``sticky'' bit is set. |
| -p filename | True if file exists and is a named pipe (FIFO). |
| -r filename | True if file exists and is readable, i.e., has its read bit set. |
| -s filename | True if file exists and has a size greater than zero. A file that exists but that has a size of zero will return false. |
| -t fd | True if the file descriptor fd is open and refers to a terminal. |
| -u filename | True if file exists and its set-user-id bit is set. |
| -w filename | True if file exists and is writable. |
| -x filename | True if file exists and is executable. |
| -G filename | True if file exists and is owned by the effective group id. |
| -L filename | True if file exists and is a symbolic link. |
| -N filename | True if file exists and has been modified since it was last read. |
| -O filename | True if file exists and is owned by the effective user id. |
| -S filename | True if file exists and is a socket. |
| file1 -ef file2 | True if file1 and file2 refer to the same device and iNode numbers. |
| file1 -nt file2 | True if file1 is newer (according to modification date) than file2, or if file1 exists and file2 does not. |
| file1 -ot file2 | True if file1 is older than file2, or if file2 exists and file1 does not. |

***Figure 9-1.*** *File operators*

Let's explore a few of these file operators to get an idea on how we might use them in CLI programs and scripts.

## EXPERIMENT 9-7

You should already have the files we will need for this experiment in your home directory. Make your home directory the PWD.

First we simply look to see if a file exists

```
[student@studentvm1 ~]$ File="cpuHog" ; if [ -e $File ] ; then echo "The file
$File exists." ; fi
The file cpuHog exists.
```

or

```
[student@studentvm1 ~]$ File="cpuHog" ; if [[ -e $File ]] ; then echo "The
file $File exists." ; fi
The file cpuHog exists.
```

or

```
[student@studentvm1 ~]$ File="cpuHog" ; if test -e $File ; then echo "The
file $File exists." ; fi
The file cpuHog exists.
```

Now add a bit of code in case the file does not exist. In this case it does exist so the end result is the same as the previous test.

```
[student@studentvm1 ~]$ File="cpuHog" ; if [ -e $File ] ; then echo "The file
$File exists." ; else echo "The file $File does not exist." ; fi
The file cpuHog exists.
```

But let's change the file name to one that we know does not exist. And note how easy it is to change the value of the $File variable rather than a text string for the file name in multiple locations in this short CLI program.

```
[student@studentvm1 ~]$ File="Non-ExistentFile" ; if [ -e $File ] ; then echo
"The file $File exists." ; else echo "The file $File does not exist." ; fi
The file Non-ExistentFile does not exist.
```

Now let's determine whether a file exists and has a non-zero length which means it contains data. We will work in the ~/chapter9 directory for this. We have three conditions we want to test for so we need a more complex set of tests. The three conditions are (1) the file does not exist, (2) the file exists and is empty, and (3) the file exists and contains data. To accomplish

this, we need to use the elif stanza in the if-elif-else construct to test for all of the conditions. But let's build up the logic a step at a time.

Our first step is to simply see if the file exists or not and print a message to STDOUT if it does.

```
[student@studentvm1 chapter9]$ File="Exp-9-7" ; if [ -s $File ] ; then echo
"$File exists and contains data." ; fi
[student@studentvm1 chapter9]$
```

We get no output because the file does not exist and we have not added an explicit test for that. The other possibility in the real world is that the file exists but does not contain data. But let's create the empty file first and see what happens.

```
[student@studentvm1 chapter9]$ File="Exp-9-7" ; touch $File ; if [ -s $File ] ;
then echo "$File exists and contains data." ; fi
[student@studentvm1 chapter9]$
```

In this case, the file exists but does not contain any data. Let's add some data and try again.

```
[student@studentvm1 chapter9]$ File="Exp-9-7" ; echo "This is file $File" >
$File ; if [ -s $File ] ; then echo "$File exists and contains data." ; fi
Exp-9-7 exists and contains data.
[student@studentvm1 chapter9]$
```

That works but it is only truly accurate for one specific condition out of the three possible ones we have identified. Let's add an else stanza so we can be somewhat more accurate and delete the file so that we can fully test this new code.

```
[student@studentvm1 chapter9]$ File="Exp-9-7" ; rm $File ; if [ -s $File ] ;
then echo "$File exists and contains data." ; else echo "$File does not exist
or is empty." ; fi
Exp-9-7 does not exist or is empty.
```

Now let's create an empty file to test.

```
[student@studentvm1 chapter9]$ File="Exp-9-7" ; touch $File ; if [ -s $File ] ;
then echo "$File exists and contains data." ; else echo "$File does not exist
or is empty." ; fi
Exp-9-7 does not exist or is empty.
[student@studentvm1 chapter9]$
```

Finally, let's add some content to the file and test again.

```
[student@studentvm1 chapter9]$ File="Exp-9-7" ; echo "This is file $File" >
$File ; if [ -s $File ] ; then echo "$File exists and contains data." ; else
echo "$File does not exist or is empty." ; fi
Exp-9-7 exists and contains data.
[student@studentvm1 chapter9]$
```

Now we add the elif stanza to discriminate between a file that does not exist and one that is empty.

```
[student@studentvm1 chapter9]$ File="Exp-9-7" ; touch $File ; if [ -s $File ]
; then echo "$File exists and contains data." ; elif [ -e $File ] ; then echo
"$File exists and is empty." ; else echo "$File does not exist." ; fi
Exp-9-7 exists and is empty.
[student@studentvm1 chapter9]$ File="Exp-9-7" ; echo "This is $File" > $File
; if [ -s $File ] ; then echo "$File exists and contains data." ; elif [ -e
$File ] ; then echo "$File exists and is empty." ; else echo "$File does not
exist." ; fi
Exp-9-7 exists and contains data.
[student@studentvm1 chapter9]$
```

So now we have a Bash CLI program that can test for these three different conditions but the possibilities are endless.

---

It is easier to see the logic structure of the more complex compound commands like we used in Experiment 9-7 if we arrange the program statements more like we would in a script that we saved in a file. Figure 9-2 shows how this would look. The indents of the program statements in each stanza of the if-elif-else structure help clarify the logic.

Note that we are not using explicit statement termination with semicolons because we are using ones implicit in a newline at the end of each statement. In this example, each program statement is on a line by itself. We also used a variable for the file name because it appears in seven places in this little program.

```
File="Exp-9-7"
echo "This is $File" > $File
if [ -s $File ]
   then
   echo "$File exists and contains data."
elif [ -e $File ]
   then
   echo "$File exists and is empty."
else
   echo "$File does not exist."
fi
```

***Figure 9-2.*** *This line-by-line listing of the Bash CLI program we used in Experiment 9-7 enables us to see the logic more clearly*

Logic of this complexity becomes too lengthy for most CLI programs. Although any Linux or Bash built-in commands may be used in CLI programs, as the CLI programs get longer and more complex, it makes sense to create a script that is stored in a file and which can be executed at any time now or in the future. We will explore scripts in more detail in Chapter 10.

## String comparison operators

String comparison operators enable comparison of alphanumeric strings of characters. There are only a few of these operators which are listed in Figure 9-3.

| Operator | Description |
|---|---|
| -z string | True if the length of string is zero. |
| -n string | True if the length of string is non-zero. |
| string1 == string2<br>or<br>string1 = string2 | True if the strings are equal. = should be used with the test command for POSIX conformance. When used with the [[ command, this performs pattern matching as described above (Compound Commands) |
| string1 != string2 | True if the strings are not equal. |
| string1 < string2 | True if string1 sorts before string2 lexicographically[6]. |
| string1 > string2 | True if string1 sorts after string2 lexicographically. |

***Figure 9-3.*** *Bash string logical operators*

## EXPERIMENT 9-8

First, let's look at string length. Notice that the quotes around $MyVar in the comparison itself must be there for the comparison to work.

```
[student@studentvm1 ~]$ MyVar="" ; if [ -z "" ] ; then echo "MyVar is zero
length." ; else echo "MyVar contains data" ; fi
MyVar is zero length.
[student@studentvm1 ~]$ MyVar="Random text" ; if [ -z "" ] ; then echo "MyVar
is zero length." ; else echo "MyVar contains data" ; fi
MyVar is zero length.
```

We could also do it this way.

```
[student@studentvm1 ~]$ MyVar="Random text" ; if [ -n "$MyVar" ] ; then echo
"MyVar contains data." ; else echo "MyVar is zero length" ; fi
MyVar contains data.
[student@studentvm1 ~]$ MyVar="" ; if [ -n "$MyVar" ] ; then echo "MyVar
contains data." ; else echo "MyVar is zero length" ; fi
MyVar is zero length
```

Since we are already talking about strings and whether they are zero length or more than zero, it might make sense that we sometimes need to know the exact length. Although this is not a comparison, it is related to them.

Unfortunately, there is no simple way to determine the length of a string. There are a couple ways to do this, but I think using the **expr** (evaluate expression) is the easiest. Read the man page for **expr** for more of what it can do. Quotes are required around the string or variable being tested.

```
[student@studentvm1 ~]$ MyVar="" ; expr length "$MyVar"
0
[student@studentvm1 ~]$ MyVar="How long is this?" ; expr length "$MyVar"
17
[student@studentvm1 ~]$ expr length "We can also find the length of a literal
string as well as a variable."
70
```

Back to our comparison operators, I use a lot of testing in my scripts to determine whether two strings are equal, that is, identical. I use the non-POSIX version of this comparison operator.

```
[student@studentvm1 ~]$ Var1="Hello World" ; Var2="Hello World" ; if [
"$Var1" == "$Var2" ] ; then echo "Var1 matches Var2" ; else echo "Var1 and
Var2 do not match." ; fi
Var1 matches Var2
[student@studentvm1 ~]$ Var1="Hello World" ; Var2="Hello world" ; if [
"$Var1" == "$Var2" ] ; then echo "Var1 matches Var2" ; else echo "Var1 and
Var2 do not match." ; fi
Var1 and Var2 do not match.
```

## Numeric comparison operators

These numeric operators make comparisons between two numeric arguments. Like the other operator classes, most of these are easy to understand.

| Operator | Description |
|---|---|
| arg1 -eq arg2 | True if arg1 equals arg2. |
| arg1 -ne arg2 | True if arg1 is not equal to arg2. |
| arg1 -lt arg2 | True if arg1 is less than arg2. |
| arg1 -le arg2 | True if arg1 is less than or equal to arg2. |
| arg1 -gt arg2 | True if arg1 is greater than arg2. |
| arg1 -ge arg2 | True if arg1 is greater than or equal to arg2. |

***Figure 9-4.*** *Bash numeric comparison logical operators*

### EXPERIMENT 9-9

Let's start with some simple examples in this experiment. In the first instance, we set the variable $X to 1 and then test to see if $X is equal to 1. It is, so the message does not get printed. In the second instance, X is set to 0 so the comparison is not true so the message is not printed.

```
[student@studentvm1 ~]$ X=1 ; if [ $X -eq 1 ] ; then echo "X equals 1" ; fi
X equals 1
[student@studentvm1 ~]$ X=0 ; if [ $X -eq 1 ] ; then echo "X equals 1" ; fi
[student@studentvm1 ~]$
```

Let's add an else stanza to this.

```
[student@studentvm1 ~]$ X=1 ; if [ $X -eq 1 ] ; then echo "X equals 1" ; else
echo "X does not equal 1" ; fi
X equals 1
```

```
[student@studentvm1 ~]$ X=0 ; if [ $X -eq 1 ] ; then echo "X equals 1" ; else
echo "X does not equal 1" ; fi
X does not equal 1
[student@studentvm1 ~]$
```

We used "-eq" which is a numeric comparison operator. We could have used "==" for a string comparison and the functional result would be the same.

```
[student@studentvm1 ~]$ X=0 ; if [ $X == 1 ] ; then echo "X equals 1" ; else
echo "X does not equal 1" ; fi
X does not equal 1
[student@studentvm1 ~]$ X=1 ; if [ $X == 1 ] ; then echo "X equals 1" ; else
echo "X does not equal 1" ; fi
X equals 1
[student@studentvm1 ~]$
```

We can also invert the meaning of the comparison using the ! character. We also must change the then and else lists.

```
[student@studentvm1 ~]$ X=0 ; if ! [ $X -eq 1 ] ; then echo "X does not equal
1" ; else echo "X equals 1" ; fi
X does not equal 1
[student@studentvm1 ~]$ X=1 ; if ! [ $X -eq 1 ] ; then echo "X does not equal
1" ; else echo "X equals 1" ; fi
X equals 1
[student@studentvm1 ~]$
```

And we also want to ensure that our logic works for other values of the variable $X as well.

```
[student@studentvm1 ~]$ X=7 ; if ! [ $X -eq 1 ] ; then echo "X does not equal
1" ; else echo "X equals 1" ; fi
X does not equal 1
[student@studentvm1 ~]$
```

## Miscellaneous operators

The miscellaneous operators described in Figure 9-5 allow us to see if a shell option is set or a shell variable has a value, but it does not discover the value of the variable, just whether it has one.

| Operator | Description |
|---|---|
| -o optname | True if the shell option optname is enabled. See the list of options under the description of the -o option to the Bash **set** builtin in the Bash man page. |
| -v varname | True if the shell variable varname is set (has been assigned a value). |
| -R varname | True if the shell variable varname is set and is a name reference. |

***Figure 9-5.*** *Miscellaneous Bash logical operators*

### EXPERIMENT 9-10

In this experiment we look to see if a variable has been set. Notice the unusual syntax; there are no quotes around Var1 and there is no $ to distinguish it as a variable rather than a fixed string. Just the use of -v and the syntax of the comparison tells Bash that Var1 is a variable.

```
[student@studentvm1 ~]$ Var1="Hello World" ; echo $Var1 ; if [ -v Var1 ] ;
then echo "Var1 has been assigned a value." ; else echo "Var1 does not have a
value." ; fi
Hello World
Var1 has been assigned a value.
[student@studentvm1 ~]$ unset Var1 ; echo $Var1 ; if [ -v Var1 ] ; then echo
"Var1 has been assigned a value." ; else echo "Var1 does not have a value." ; fi

Var1 does not have a value.
```

The general rule for using the $ character to specify a variable is that it should be used when accessing – reading – the value of the variable. It should not be used when setting the value or when using logical operators.

You should experiment with all of these logical comparison operators, especially the ones not covered explicitly in any of these experiments. However, it is not necessary to memorize all of them along with all of their options and forms. I always find it most

beneficial to explore specific operators – and Linux commands in general – while working on a project that requires them. I learn and retain more that way. Over time I have also learned which ones I use frequently and which I almost never use. I see little point in memorizing information that I may never use.

# Grouping program statements

Sometimes, to get the results you want, it is necessary to group program statements together. For example, I sometimes want to know how much time running a program takes to run on one host so I know what to expect on the rest of the hosts on which I need to run the same program. The **time** utility shows me the amount of real time, user time, and system time.

Real time is obvious; it is the total amount of clock time used by a program. User time is the amount of time spent by the system to execute the user code that was entered. System (sys) is the amount of time spent running system code and libraries.

**EXPERIMENT 9-11**

Let's look first at the **time** utility to see how that works. It can also illustrate a bit about the time delays introduced by Input/Output (I/O). Do this part as the student user.

```
[student@studentvm1 ~]$ time cat dmesg.txt
<snip>
[40368.982838] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready
[40371.049241] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow
Control: RX
[40371.049584] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready

real    0m0.026s
user    0m0.001s
sys     0m0.002s
```

The last three lines show the results from the **time** utility. The result can be interpreted that a total of 0.003 seconds was used in executing code. The rest of the time, 0.023 seconds, was spent waiting for I/O to occur.

If you run this program several times, the amount of real time usually will be significantly reduced due to caching the results of the disk read in memory where it can be accessed faster. I ended up with something like this.

```
real    0m0.007s
user    0m0.001s
sys     0m0.001s
```

However, if I then run this little program and redirect the output to the /dev/null, I get the following results, and you should see something quite similar.

```
[student@studentvm1 ~]$ time cat dmesg.txt > /dev/null

real    0m0.002s
user    0m0.001s
sys     0m0.000s
```

So we can see that sending data to the display screen (I/O activity) takes a relatively large amount of real time. When we send the data to /dev/null instead, the whole thing takes much less time.

Now let's move on to the real purpose of this experiment. Suppose that we want to run multiple program statements and measure the total time that it takes to do so. That might look like Figure 9-6 in which I want to know how much time is used by the pair of commands I normally use to destroy the content of a storage drive and then test it for errors.

```
[root@david ~]# time ( shred -n 3 -v /dev/sdk ; dd if=/dev/sdk of=/dev/null )

shred: /dev/sdk: pass 1/3 (random)...

shred: /dev/sdk: pass 1/3 (random)...147MiB/466GiB 0%

shred: /dev/sdk: pass 1/3 (random)...322MiB/466GiB 0%


<snip>


7814037167+0 records in
7814037167+0 records out
4000787029504 bytes (4.0 TB, 3.6 TiB) copied, 39041.5 s, 102 MB/s
real    1986m28.783s
user    85m49.873s
sys     127m49.951s
```

***Figure 9-6.*** *Grouping the* ***shred*** *and* ***dd*** *commands so that the* ***time*** *command measures elapsed time for both commands*

If you have been performing all of the experiments in this course, your VM should have some virtual storage devices we can use for this. Perform the rest of this experiment as the root user. The **lsblk** command should show that /dev/sdd is 2GB in size and has one partition, /dev/sdd1, that is configured as a swap partition. We can also see this with these two commands.

If you don't see two swap partitions, you may have turned off the one we created in Chapter 5 of this volume. If you did, turn all swap space on, then redo this command.

```
[root@studentvm1 home]# cat /proc/swaps
Filename                                Type            Size    Used    Priority
/dev/dm-1                               partition       6291452 0       6
/dev/sdd1                               partition       2096124 0       3
[root@studentvm1 home]# swapon -s
Filename                                Type            Size    Used    Priority
/dev/dm-1                               partition       6291452 0       6
/dev/sdd1                               partition       2096124 0       3
```

Where do you think that the swapon -s (-s for summary) command obtains its information?

Let's turn off /dev/sdd1 as swap and verify that it is off.

```
[root@studentvm1 home]# swapoff /dev/sdd1 ; swapon -s
Filename                                Type            Size    Used    Priority
/dev/dm-1                               partition       6291452 0       6
```

And comment out the following line as shown in /etc/fstab.

```
# /dev/sdd1         swap        swap    pri=3,defaults        0 0
```

Now the VM won't try to start /dev/sdd1 as swap space when rebooting.

Let's shred the entire storage device, not just the partition. By doing it this way, we also shred the boot record, partition table, and the entire data area of the device.

```
[root@studentvm1 home]# time shred -vn 3 /dev/sdd ; dd if=/dev/sdd of=/dev/null
shred: /dev/sdd: pass 1/3 (random)...
shred: /dev/sdd: pass 1/3 (random)...564MiB/2.0GiB 27%
shred: /dev/sdd: pass 1/3 (random)...1.0GiB/2.0GiB 50%
<snip>
shred: /dev/sdd: pass 3/3 (random)...1.6GiB/2.0GiB 84%
shred: /dev/sdd: pass 3/3 (random)...2.0GiB/2.0GiB 100%
```

```
real    0m56.186s
user    0m0.831s
sys     0m3.722s
4194304+0 records in
4194304+0 records out
   2147483648 bytes (2.1 GB, 2.0 GiB) copied, 13.3271 s, 161 MB/s
```

Notice where the data from the time command appears – between the output from the **shred** and **dd** commands. So the time shown is only for the **shred** command. What result would we get using the following command?

```
shred -vn 3 /dev/sdd ; time dd if=/dev/sdd of=/dev/null
```

We know that won't give us the time for the entire operation. Try it like this, which gives us the total time for both program statements that are enclosed in the parentheses.

```
[root@studentvm1 home]# time ( shred -vn 3 /dev/sdd ; dd if=/dev/sdd of=/dev/null )
shred: /dev/sdd: pass 1/3 (random)...
shred: /dev/sdd: pass 1/3 (random)...571MiB/2.0GiB 27%
<snip>
shred: /dev/sdd: pass 3/3 (random)...1.6GiB/2.0GiB 81%
shred: /dev/sdd: pass 3/3 (random)...2.0GiB/2.0GiB 100%
4194304+0 records in
4194304+0 records out
2147483648 bytes (2.1 GB, 2.0 GiB) copied, 13.1474 s, 163 MB/s

real    1m6.207s
user    0m1.149s
sys     0m9.872s
```

# Expansions

Bash supports a number of types of expansions and substitutions which can be quite useful. According to the Bash man page, Bash has seven (7) forms of expansions. We will look at tilde (~) expansion, arithmetic expansion, and pathname expansion.

## Brace expansion

Brace expansion is a method to use for generating arbitrary strings. We have already discussed brace expansion in Chapter 15 of Volume 1 so there is no need to explore that any further here.

## Tilde expansion

Arguably the most common expansion we run into is the tilde (~) expansion. When we use this in a command like `cd ~/Documents`, the Bash shell actually expands that shortcut to the full home directory of the user.

**EXPERIMENT 9-12**

As the student user, use these Bash programs to observe the effects of the tilde expansion.

```
[student@studentvm1 ~]$ echo ~
/home/student
[student@studentvm1 ~]$ echo ~/Documents
/home/student/Documents
[student@studentvm1 ~]$ Var1=~/Documents ; echo $Var1 ; cd $Var1
/home/student/Documents
[student@studentvm1 Documents]$
```

## Pathname expansion

Pathname expansion is fancy term for expansion of file globbing patterns using the characters ? and ∗ into the full names of directories which match the pattern. As we discussed in Chapter 15 of Volume 1, globbing means special pattern characters that allow us significant flexibility in matching file names, directories, and other strings when performing various actions. These special pattern characters allow matching single, multiple, or specific characters in a string:

?: Matches only one of any character in the specified location within the string

∗: Zero or more of any character in the specified location within the string

In this case we apply this expansion to matching directory names.

## EXPERIMENT 9-13

As the student user, let's see how this works. Ensure that your home directory, ~, is the PWD and start with a plain listing.

```
[student@studentvm1 ~]$ ls
chapter7         dmesg1.txt    Documents    newfile.txt     testdir     Videos
chapter9         dmesg2.txt    Downloads    Pictures        testdir1
cpuHog           dmesg3.txt    hello.txt    Public          testdir6
Desktop          dmesg4.txt    Music        random.txt      testdir7
diskusage.txt    dmesg.txt     mypipe       Templates       umask.test
[student@studentvm1 ~]$
```

Now list the directories that start with "Do", ~/Documents, and ~/Downloads.

```
[student@studentvm1 ~]$ ls Do*
Documents:
Directory01 file07 file15       test02 test10 test20     testfile13  TextFiles
Directory02 file08 file16       test03 test11 testfile01 testfile14
file01      file09 file17       test04 test12 testfile04 testfile15
file02      file10 file18       test05 test13 testfile05 testfile16
file03      file11 file19       test06 test14 testfile09 testfile17
file04      file12 file20       test07 test15 testfile10 testfile18
file05      file13 Student1.txt test08 test16 testfile11 testfile19
file06      file14 test01       test09 test18 testfile12 testfile20

Downloads:
[student@studentvm1 ~]$
```

Well that did not do what we want. It listed the contents of the directories that begin with Do. To list only the directories and not their contents, we can use the -d option.

```
[student@studentvm1 ~]$ ls -d Do*
Documents  Downloads
[student@studentvm1 ~]$
```

So what happens here – in both cases – is that the Bash shell expands the Do* pattern into the names of the two directories that match the pattern. But what if there are also files that match the pattern, which there currently are not.

```
[student@studentvm1 ~]$ touch Downtown ; ls -d Do*
Documents  Downloads  Downtown
[student@studentvm1 ~]$
```

That shows the file too. So any files that match the pattern are also expanded to their full names.

## Command substitution

Command substitution is a form of expansion. Command substitution is a tool that allows the STDOUT data stream of one command to be used as the argument of another command, for example, as a list of items to be processed in a loop. The Bash man page says, "Command substitution allows the output of a command to replace the command name." I find that to be accurate, if a bit obtuse.

There are two forms of this substitution, `` `command` `` and `$(command)`. In the older form using back tics (`), a backslash (\) used in the command retains its literal meaning. However, when used in the new parenthetical form, the backslash takes on its meaning as a special character. Note also that the parenthetical form uses only single parentheses to open and close the command statement.

I frequently use this capability in command-line programs and scripts where the results of one command can be used as an argument for another command.

### EXPERIMENT 9-14

As the student user, let's start with a very simple example using both forms of this expansion. Ensure that ~ is the PWD.

```
[student@studentvm1 testdir7]$ echo "Todays date is `date`"
Todays date is Sun Apr  7 14:42:46 EDT 2019
[student@studentvm1 testdir7]$ echo "Todays date is $(date)"
Todays date is Sun Apr  7 14:42:59 EDT 2019
[student@studentvm1 testdir7]$
```

We have seen the seq utility previously. It is used to generate a sequence of numbers.

```
[student@studentvm1 ~]$ seq 5
1
2
3
4
5
[student@studentvm1 ~]$ echo `seq 5`
1 2 3 4 5
[student@studentvm1 testdir7]$
```

Notice that by using command substitution we lose the newlines at the end of each number in the sequence. We have already used this when creating new files for testing in previous experiments. Let's look at it again. Make ~/chapter9 the PWD and we will create some new files in that directory. The -w option to the **seq** utility adds leading zeros to the numbers generated to that they are all the same width, that is, number of digits regardless of the value. This makes it easier to sort them in numeric sequence. We have done this before, but this time let's focus on the function of the command substitution.

```
[student@studentvm1 chapter9]$ for I in $(seq -w 5000) ; do touch file-$I ;
done
```

In this usage, the statement **seq -w 5000** generates a list of numbers from 1 to 5000. By using command substitution as part of the **for** statement, the list of numbers is used by the **for** statement to generate the numerical part of the file names.

List the files in the directory to ensure that they were properly created.

```
[student@studentvm1 chapter9]$ ls | column | less
```

We will explore the use of the **for** statement just a little further on in this chapter.

## Arithmetic expansion

Bash does do integer math but it is rather cumbersome to do so, as you will soon see. The syntax for arithmetic expansion is `$((arithmetic-expression))` using double parentheses to open and close the expression.

Arithmetic expansion works like command substitution in a shell program or script – the value calculated from the expression replaces the expression for further evaluation by the shell.

## EXPERIMENT 9-15

Once again we will start with something simple.

```
[student@studentvm1 chapter9]$ echo $((1+1))
2
[student@studentvm1 chapter9]$ Var1=5 ; Var2=7 ; Var3=$((Var1*Var2)) ; echo
"Var 3 = $Var3"
Var 3 = 35
```

The following division results in zero because the result would be a decimal value of less than one.

```
[student@studentvm1 chapter9]$ Var1=5 ; Var2=7 ; Var3=$((Var1/Var2)) ; echo
"Var 3 = $Var3"
Var 3 = 0
```

Here is a simple calculation that I do in a script or CLI program that tells me how much total virtual memory I have in a Linux host. The **free** command does not provide that data.

```
[student@studentvm1 chapter9]$ RAM=`free | grep ^Mem | awk '{print $2}'` ;
Swap=`free | grep ^Swap | awk '{print $2}'` ; echo "RAM = $RAM and Swap =
$Swap" ; echo "Total Virtual memory is $((RAM+Swap))" ;
RAM = 4037080 and Swap = 6291452
Total Virtual memory is 10328532
```

Note that I used the ` character to delimit the sections of code that were used for command substitution.

I use Bash arithmetic expansion mostly for checking system resource amounts in a script and then choosing a program execution path based on the result.

# for loops

Every programming language I have ever used has some version of the **for** command. The Bash implementation of this structure is, in my opinion, a bit more flexible than most because it can handle non-numeric values while the standard C language for loop, for example, can only deal with numeric values.

The basic structure of the Bash version of the **for** command is simple – `for Var in list1 ; do list2 ; done`. This translates to, "For each value in list1, set the $Var to that value and then perform the program statements in list2 using that value; when all of the values in list1 have been used, we are done, so exit the loop." The values in list1 can be a simple explicit string of values or it can be the result of a command substitution as we have seen in Experiment 9-14 and many others throughout this course.

As you can see from previous experiments in this course, I use this construct frequently.

## EXPERIMENT 9-16

As the student user, ensure that ~/chapter9 is still the PWD. Let's clean up and then look at a trivial example of the **for** loop starting with an explicit list of values. This list is a mix of alphanumeric values but do not forget that all variables are strings and can be treated as such.

```
[student@studentvm1 chapter9]$ rm -rf *
[student@studentvm1 chapter9]$ for I in a b c d 1 2 3 4 ; do echo $I ; done
a
b
c
d
1
2
3
4
```

A bit more useful version along with a more meaningful variable name.

And even more useful…

```
[student@studentvm1 chapter9]$ for Dept in "Human Resources" Sales Finance
"Information Technology" Engineering Administration Research ; do echo
"Department $Dept" ; done
Department Human Resources
Department Sales
Department Finance
Department Information Technology
Department Engineering
Department Administration
Department Research
```

Let's make some directories.

```
[student@studentvm1 chapter9]$ for Dept in "Human Resources" Sales Finance
"Information Technology" Engineering Administration Research ; do echo
"Working on Department $Dept" ; mkdir "$Dept"  ; done
Working on Department Human Resources
Working on Department Sales
Working on Department Finance
Working on Department Information Technology
Working on Department Engineering
Working on Department Administration
Working on Department Research
[student@studentvm1 chapter9]$ ll
total 28
drwxrwxr-x 2 student student 4096 Apr  8 15:45  Administration
drwxrwxr-x 2 student student 4096 Apr  8 15:45  Engineering
drwxrwxr-x 2 student student 4096 Apr  8 15:45  Finance
drwxrwxr-x 2 student student 4096 Apr  8 15:45 'Human Resources'
drwxrwxr-x 2 student student 4096 Apr  8 15:45 'Information Technology'
drwxrwxr-x 2 student student 4096 Apr  8 15:45  Research
drwxrwxr-x 2 student student 4096 Apr  8 15:45  Sales
```

Note that it is necessary to enclose $Dept in quotes in the mkdir statement, or the two-part department names such as "Information Technology" will be treated as two separate departments. That highlights a best practice that I like to follow and that is all file and directory names should be a single word. Although most modern operating systems can deal with spaces in those names, it takes extra work for SysAdmins to ensure that those special cases

are considered in scripts and CLI programs. But they almost certainly should be considered, even if they're annoying, because you never know what files you're actually going to have.

So delete everything in ~/chapter9 – again – and let's do this one more time.

```
[student@studentvm1 chapter9]$ rm -rf * ; ll
total 0
[student@studentvm1 chapter9]$ for Dept in Human-Resources Sales Finance
Information-Technology Engineering Administration Research ; do echo "Working
on Department $Dept" ; mkdir "$Dept"  ; done
Working on Department Human-Resources
Working on Department Sales
Working on Department Finance
Working on Department Information-Technology
Working on Department Engineering
Working on Department Administration
Working on Department Research
[student@studentvm1 chapter9]$ ll
total 28
drwxrwxr-x 2 student student 4096 Apr  8 15:52 Administration
drwxrwxr-x 2 student student 4096 Apr  8 15:52 Engineering
drwxrwxr-x 2 student student 4096 Apr  8 15:52 Finance
drwxrwxr-x 2 student student 4096 Apr  8 15:52 Human-Resources
drwxrwxr-x 2 student student 4096 Apr  8 15:52 Information-Technology
drwxrwxr-x 2 student student 4096 Apr  8 15:52 Research
drwxrwxr-x 2 student student 4096 Apr  8 15:52 Sales
```

Suppose someone asks for a list of all RPMs on a particular Linux computer and a short description of each. This happened to me while I worked at the State of North Carolina. Since Open Source was not "approved" for use by state agencies at that time and I only used Linux on my desktop computer, the pointy haired bosses (PHBs) needed a list of each piece of software that was installed on my computer so that they could "approve" an exception.

How would you approach that? Here is one way, starting with the knowledge that the **rpm -qi** command provides a complete description of an RPM including the two items we want, the name and a brief summary.

## EXPERIMENT 9-17

We will build up to the final result one step at a time. This experiment should be performed as the student user. First we list all RPMs.

```
[student@studentvm1 chapter9]$ rpm -qa
perl-HTTP-Message-6.18-3.fc29.noarch
perl-IO-1.39-427.fc29.x86_64
perl-Math-Complex-1.59-429.fc29.noarch
lua-5.3.5-2.fc29.x86_64
java-11-openjdk-headless-11.0.ea.28-2.fc29.x86_64
util-linux-2.32.1-1.fc29.x86_64
libreport-fedora-2.9.7-1.fc29.x86_64
rpcbind-1.2.5-0.fc29.x86_64
libsss_sudo-2.0.0-5.fc29.x86_64
libfontenc-1.1.3-9.fc29.x86_64
<snip>
```

Adding the **sort** and **uniq** commands sorts the list and then prints only the unique ones. There is a slight possibility that some RPMs with identical names are installed.

```
[student@studentvm1 chapter9]$ rpm -qa | sort | uniq
a2ps-4.14-39.fc29.x86_64
aajohan-comfortaa-fonts-3.001-3.fc29.noarch
abattis-cantarell-fonts-0.111-1.fc29.noarch
abiword-3.0.2-13.fc29.x86_64
abrt-2.11.0-1.fc29.x86_64
abrt-addon-ccpp-2.11.0-1.fc29.x86_64
abrt-addon-coredump-helper-2.11.0-1.fc29.x86_64
abrt-addon-kerneloops-2.11.0-1.fc29.x86_64
abrt-addon-pstoreoops-2.11.0-1.fc29.x86_64
abrt-addon-vmcore-2.11.0-1.fc29.x86_64
<snip>
```

Since this gives the correct list of RPMs you want to look at, we can use this as the input list to a loop that will print all of the details of each RPM.

```
[student@studentvm1 chapter9]$ for RPM in `rpm -qa | sort | uniq` ; do rpm
-qi $RPM ; done
```

This code produces way more data than was desired. Note that our loop is actually complete. The next step is to extract only the information that was requested. Now we add an **egrep** command which is used to select ^Name or ^Summary. Thus, any line with Name or Summary at the beginning of the line (the carat ^ specifies the beginning of the line) is displayed.

```
[student@studentvm1 chapter9]$ for RPM in `rpm -qa | sort | uniq` ; do rpm
-qi $RPM ; done | egrep -i "^Name|^Summary"
Name        : a2ps
Summary     : Converts text and other types of files to PostScript
Name        : aajohan-comfortaa-fonts
Summary     : Modern style true type font
Name        : abattis-cantarell-fonts
Summary     : Humanist sans serif font
Name        : abiword
Summary     : Word processing program
Name        : abrt
Summary     : Automatic bug detection and reporting tool
<snip>
```

You can try grep instead of egrep in the preceding command but it does not work. You could also pipe the output of this command through the less filter so you can explore these results.

The final command sequence looks like this.

```
[student@studentvm1 chapter9]$ for RPM in `rpm -qa | sort | uniq` ; do rpm
-qi $RPM ; done | egrep -i "^Name|^Summary" > RPM-summary.txt
```

It uses pipelines, redirection, and a for loop – all on a single line. It redirects the output of our little CLI program to a file that can be used in an email or as input for other purposes.

This process of building up the program one step at a time allows you to see the results of each step and to ensure that it is working as you expect and provides the desired results.

---

Note that the PHBs received a list of over 1900 separate RPM packages. I seriously doubt that anyone actually read that list. But I gave them exactly what they asked for and I never heard another word from them about this.

# Other loops

There are two more types of loop structures available in Bash: the **`while`** and **`until`** structures, which are very similar to each other in both syntax and function. The basic syntax of these loop structures is simple.

```
while [ expression ] ; do list ; done
```

and

```
until [ expression ] ; do list ; done
```

The logic of these reads as follows: "While the expression evaluates as true, execute the list of program statements. When the expression evaluates as false, exit from the loop." And "Until the expression evaluates as true, execute the list of program statements. When the expression evaluates as true, exit from the loop."

## while

The while loop is used to execute a series of program statements, while (so long as) the logical expression evaluates to true. We used this as part of the cpuHog programs we wrote in Chapter 4. Let's look at the while loop again in more detail.

**EXPERIMENT 9-18**

As the student user, make ~ the PWD.

The simplest form of the while loop is one that runs forever. In the following form, we use the **`true`** statement to always generate a "true" return code. We could use a simple "1" as we did in the original cpuHog and that would work just the same, but this illustrates the use of the **`true`** statement.

Let's pipe the results through the head statement to limit output to the first ten lines of the data stream.

```
[student@studentvm1 ~]$
0
1
2
3
```

```
4
5
<snip>
```

This CLI program should make more sense now that we have studied its parts. First we set $X to zero just in case it had some leftover value from a previous program or CLI command. Then, since the logical expression **[ true ]** always evaluates to 1, which is true, the list of program instructions between do and done is executed forever – or until we press Ctrl-C or otherwise send a signal 2 to the program. Those instructions are an arithmetic expansion that prints the current value of $X and then increments it by one.

One of the tenets of *The Linux Philosophy for SysAdmins* is to strive for elegance and that one way to achieve elegance is simplicity. We can simplify this program by using the variable increment operator, ++. In this first instance, the current value of the variable is printed and then the variable is incremented. This is indicated by placing the ++ operator after the variable.

```
[student@studentvm1 ~]$ X=0 ; while [ true ] ; do echo $((X++)) ; done | head
0
1
2
3
4
5
6
```

Now delete **| head** from the end of the program and run it again.

In this next version, the variable is incremented before its value is printed. This is specified by placing the ++ operator before the variable. Can you see the difference?

```
[student@studentvm1 ~]$ X=0 ; while [ true ] ; do echo $((++X)) ; done | head
1
2
3
4
5
6
```

We have reduced two statements into a single one that both print the value of the variable and increment that value. There is also a decrement operator, --.

We need a method for stopping the loop at a specific number. To accomplish that, we can change the **true** expression to an actual numeric evaluation expression. So let's have our program loop to 5 and stop. In Figure 9-4 you can see that -le is the logical numeric operator for "less than or equal to." This means that so long as $X is less than or equal to 5, the loop will continue. When $X increments to 6, the loop terminates.

```
[student@studentvm1 ~]$ X=0 ; while [ $X -le 5 ] ; do echo $((X++)) ; done
0
1
2
3
4
5
[student@studentvm1 ~]$
```

## until

The **until** command is very much like the **while** command. The difference is that it will continue to loop until the logical expression evaluates to "true."

### EXPERIMENT 9-19

As the student user, make ~ the PWD. As in Experiment 9-18, let's look at the simplest form of this construct.

```
[student@studentvm1 ~]$ X=0 ; until false  ; do echo $((X++)) ; done | head
0
1
2
3
4
5
6
7
8
9
[student@studentvm1 ~]$
```

Now we use a logical comparison to count to a specific value.

```
[student@studentvm1 ~]$ X=0 ; until [ $X -eq 5 ]  ; do echo $((X++)) ; done
0
1
2
3
4
[student@studentvm1 ~]$ X=0 ; until [ $X -eq 5 ]  ; do echo $((++X)) ; done
1
2
3
4
5
[student@studentvm1 ~]$
```

# Chapter summary

We have explored the use of many powerful tools that we can use to build command-line programs and Bash shell scripts. Despite the interesting things we have done in this chapter, Bash command-line programs and shell scripts can do so much more. We have barely scratched the surface.

All we have done here is to inform you of the many possibilities of Bash command-line programming. The rest is up to you. I have discovered over time that the best way to learn Bash programming is to actually do it. Find a simple project that requires multiple Bash commands and make a CLI program out of them. SysAdmins do many tasks that lend themselves to CLI programming this way so I am sure that you will easily find tasks to automate.

Despite being familiar with other shell languages and Perl, many years ago, I made the decision to use Bash for all of my SysAdmin automation tasks. I have discovered that – perhaps with a bit of searching – I have been able to accomplish everything I need.

# Exercises

Perform the following exercises to complete this chapter:

1. Write a short command-line program to count from 0 to 5000 by increments of 5, and print the resulting data in two columns.
2. What happens when quotes are not used when assigning a value to a variable?
3. How do the variables $PATH and $Path differ?
4. Devise and run an experiment to determine whether the -r file operator returns true if any of User, Group, or Other, ownership for a given file has the read bit set, as opposed to specifically the read bit for User.
5. Create two versions of an "if" compound command that tests if two variables are equal and print "The variables are equal" if they are and "The variables are not equal" if they are not. One version should use the == operator and the other should use the != operator. Use test cases for both versions where the variables are equal and where they are not.
6. Is the CLI program **`Var1="7" ; Var2="9" ; echo "Result = $Var1*$Var2"`** valid? What about **`Var1="7" ; Var2="9" ; echo "Result = $Var1*$Var2"`**? Why?
7. What happens when you use a decimal such as 5.5 in an arithmetic expansion?
8. Which of these works and which does not? Why?

   ```
   RAM=`free | grep ^Mem | awk '{print $2}'` ; echo $RAM
   RAM=$((free | grep ^Mem | awk '{print $2}')) ; echo $RAM
   ```

9. Create a CLI program to count down from 10 to 0 and print the resulting numbers to the screen.

CHAPTER 10

# Automation with Bash Scripts

## Objectives

In this chapter you will learn

- The advantages of automation with Bash shell scripts
- Why using shell scripts is a better choice for SysAdmins than compiled languages like C or C++
- To create a set of requirements for new scripts
- To create simple Bash shell scripts from CLI programs
- To use the file ownership and permissions to impose a layer of security on who can run the script
- To further enhance security through the use of the UID of the user running the script
- To use logical comparison tools to provide execution flow control for both command-line programs and scripts
- To use command-line options to control various script functionality
- To create Bash functions that can be called from one or more locations within a script
- Why and how to license your code as open source
- To create a simple test plan
- To test early and test often




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_10

# Introduction

My question to you is "What is the function of computers?" In my opinion, the right answer is "to automate mundane tasks in order to allow us humans to concentrate on the tasks that the computers cannot – yet – do." For SysAdmins, those of us who run and manage the computers most closely, we have direct access to the tools that can help us work more efficiently. We should use those tools to maximum benefit.

In this chapter we explore using automation in the form of Bash shell scripts to make our own lives as SysAdmins easier. This chapter is only partly about creating the scripts and making them work. It is also about some of the philosophy of automation and shell scripting.

# Why I use shell scripts

In Chapter 9 of my book, *The Linux Philosophy for SysAdmins*,[1] I state:

"A SysAdmin is most productive when thinking – thinking about how to solve existing problems and about how to avoid future problems; thinking about how to monitor Linux computers in order to find clues that anticipate and foreshadow those future problems; thinking about how to make her job more efficient; thinking about how to automate all of those tasks that need to be performed whether every day or once a year."

"SysAdmins are next most productive when creating the shell programs that automate the solutions that they have conceived while appearing to be unproductive. The more automation we have in place the more time we have available to fix real problems when they occur and to contemplate how to automate even more than we already have."

Have you ever performed a long and complex task at the command line thinking "Glad that's done – I never have to worry about it again"? I have – very frequently. I ultimately figured out that almost everything that I ever need to do on a computer, whether mine, one that belongs to an employer, or a consulting customer, will need to be done again sometime in the future.

Of course I always think that I will remember how I did the task in question. But the next time I need to do it is far enough out into the future that I sometimes even forget that I have ever done it at all, let alone how to do it. For some tasks I started writing down

[1]Both, David, *The Linux Philosophy for SysAdmins*, Apress, 2018, 165

the steps required on a bit of paper. I thought, "How stupid of me!" So I then transferred those scribbles to a simple notepad-type application on my computer. Suddenly one day I thought again, "How stupid of me!" If I am going to store this data on my computer, I might as well create a shell script and store it in a standard location, /usr/local/bin or ~/bin, so that I can just type the name of the shell program and it does all of the tasks I used to do manually.

For me automation also means that I don't have to remember or recreate the details of how I performed that task in order to do it again. It takes time to remember how to do things and time to type in all of the commands. This can become a significant time sink for tasks that require typing large numbers of long commands. Automating tasks by creating shell scripts reduces the typing necessary to perform my routine tasks.

## Shell scripts

Shell scripts can also be an important aid to newer SysAdmins to enable them to keep things working while the senior SysAdmin – or whoever knows more or has more experience than those who are left, which is usually us – is out on vacation or ill. Figuring out how to do things takes time, even if it's a faster process for the more experienced. Because shell programs are inherently open to view and change, they can be an important tool for less experienced SysAdmins to learn the details of how to perform these tasks when they need to be responsible for them.

Writing shell programs – also known as scripts – provides the best strategy for leveraging my time. Once having written a shell program, it can be rerun as many times as needed. I can update my shell scripts as needed to compensate for changes from one release of Linux to the next. Other factors that might require making these changes are the installation of new hardware and software, changes in what I want or need to accomplish with the script, adding new functions, removing functions that are no longer needed, and fixing the not-so-rare bugs in my scripts. These kinds of changes are just part of the maintenance cycle for any type of code.

Every task performed via the keyboard in a terminal session by entering and executing shell commands can and should be automated. SysAdmins should automate everything we are asked to do or that we decide on our own needs to be done. Many times I have found that doing the automation up front saves time the first time.

One bash script can contain anywhere from a few commands to many thousands. In fact, I have written bash scripts that have only one or two commands in them. Another script I have written contains over 2700 lines, more than half of which are comments.

## Scripts vs. compiled programs

When writing programs to automate – well, everything – always use shell scripts. Because shell scripts are stored in ASCII text format, they can be easily viewed and modified by humans just as easily as they can by computers. You can examine a shell program and see exactly what it does and whether there are any obvious errors in the syntax or logic. This is a powerful example of what it means to be open.

I know some developers tend to consider shell scripts something less than true programming. This marginalization of shell scripts and those who write them seems to be predicated on the idea that the only true programming language is one that must be compiled from source code to produce executable code. I can tell you from experience this is categorically untrue.

I have used many languages including BASIC, C, C++, Pascal, Perl, Tcl/Expect, REXX, and some of its variations including Object REXX; many shell languages including Korn, csh, and Bash; and even some assembly language. Every computer language ever devised has had one purpose – to allow humans to tell computers what to do. When you write a program, regardless of the language you choose, you are giving the computer instructions to perform specific tasks in a specific sequence.

Scripts can be written and tested far more quickly than compiled languages. Programs usually must be written quickly to meet time constraints imposed by circumstances or the PHB. Most of the scripts we write are to fix a problem, to clean up the aftermath of a problem, or to deliver a program that must be operational long before a compiled program could be written and tested.

Writing a program quickly requires shell programming because it allows quick response to the needs of the customer whether that be ourselves or someone else. If there are problems with the logic or bugs in the code, they can be corrected and retested almost immediately. If the original set of requirements was flawed or incomplete, shell scripts can be altered very quickly to meet the new requirements. So, in general, we can say that the need for speed of development in the SysAdmin's job overrides the need to make the program run as fast as possible or to use as little as possible in the way of system resources like RAM.

Most things we do as SysAdmins take longer to figure out how to do than they do to execute. Thus, it might seem counter-productive to create shell scripts for everything we do. Writing the scripts and making them into tools that produce reproducible results and which can be used as many times as necessary takes some time. The time savings come every time we can run the script without having to figure out again how to perform the task.

You may encounter a situation when the execution of a script takes an excessive amount of time or when the problem is so general that it will be done thousands or even millions of times, in which case compiled languages make more sense. But those are extremely rare cases.

# Updates

Most of the time my scripts start with short CLI programs that I use multiple times daily and that are morphing into more complex forms. So let's take a CLI program we have already used and turn it into a script.

One task I do frequently is to install updates on all of my computers. In fact, I have been doing updates this morning. This is a task that requires only a couple decisions and can be easily automated. "But that is so simple, why automate a task that requires only a command or two?" It turns out that updates are not so simple. Let's think about this for a minute.

## About updates

There are two important things to understand about updates as we enter into this programming task. First, installing updates does not preclude using your computer at the same time. With Linux, we can install updates while also performing the other tasks necessary to get our regular work done.

Second, there are times when rebooting after installing updates is a good idea. Linux does not automatically reboot for us – it leaves us to decide when to perform that reboot. Although I usually reboot right after the updates are completed, it is not necessary to do so. But if certain packages are updated, it is a very good idea to reboot soon. The point is that Linux lets us make these choices.

## Create a list of requirements

So what are the requirements for this script? You should always create a set of requirements for every project. I always create a set of requirements for my scripts, even if it is a simple list with only two or three items on it.

First I must determine whether any updates are available. Then I need to determine whether a package that requires a reboot is being updated, such as the kernel, glibc, or systemd. At this point I can install the update. Before I do a reboot, assuming one is required, I run the `mandb` utility to update the man pages; if this is not done, new and replacement man pages won't be accessible and old ones that have been removed will appear to be there even though they are not. Then, if the kernel has been updated, I rebuild the grub boot loader configuration file so that it includes recovery options for each installed kernel. Finally, if a reboot is needed, I do that.

That is a non-trivial set of individual tasks and commands that require some decisions. Doing those tasks manually requires paying attention and intervention to enter new commands when the previous ones complete. Because of the need to babysit while waiting to enter the next command, this would take a great deal of my time to monitor each computer as it went through the procedures. There was room for error as I was reminded occasionally when I would enter the wrong command on a host.

Using the statement of requirements I created earlier, because that is what that paragraph really is, it was easy to automate this to eliminate all of those issues. I wrote a little script that I call doUpdates. It provides options like help, verbose mode, printing the current version number, and an option to reboot only if the kernel, systemd, or glibc has been updated.

Over half of the lines in this program are comments so I can remember how the program works the next time I need to work on it to fix a bug or add a little more function. I arbitrarily chose this as the program to illustrate creating scripts because it offers many opportunities to expand and implement fun and useful features. It is also illustrative of the process I went through as it grew and became more than a CLI program.

As we work through the series of experiments in this chapter, we will start out very simply. To begin, we will only do the check for updates and some items such as a help facility. Because it may be a few days after doing an actual update that another is needed, we will wait until near the end to actually perform the update. This is, in fact, how I develop my scripts anyway – so that they start out harmless.

The program we will create together in this chapter will be a shorter and more efficient version of the one I have created for myself.

## The CLI program

There are four steps required to actually do the updates in the CLI program. We first do the update, update the man page database, rebuild the grub config files, and reboot the host. Refer to Chapter 16 of Volume 1 to review grub configuration and why we need to rebuild the grub configuration.

Let's make some assumptions for our initial command-line program. We will assume that the man database will always be rebuilt, that a reboot is always required – although we won't always do that in our testing – and that the grub configuration file needs to be updated.

**EXPERIMENT 10-1**

As the root user, start with this little CLI program. Remember that we are only checking for updates and not yet doing them. This will leave something for the script to do when it is complete. Enter and run the following CLI program and observe the results.

```
[root@studentvm1 ~]# dnf check-update ; mandb ; grub2-mkconfig > /boot/grub2/
grub.cfg ; reboot
```

It gives us a list of the RPM packages that need to be updated, rebuilds the man database, regenerates the grub configuration file, and then reboots the VM.

## Convert the CLI program to a script

We now understand the basic steps required to do the job. That does not seem like a lot, but if you have tens or hundreds of computers to update, it would amount to a lot of typing. So let's create a very minimal script to do these steps.

## EXPERIMENT 10-2

As the root user, ensure that root's home directory is the PWD. Then create a new file named doUpdates and make it executable for root and the root group but with no permissions of any kind for other users.

```
[root@studentvm1 ~]# cd ; touch doUpdates ; chmod 770 doUpdates ; ll
total 8
-rw-------. 1 root root 2118 Dec 22 11:07 anaconda-ks.cfg
-rwxrwx---  1 root root    0 Apr 10 16:15 doUpdates
-rw-r--r--. 1 root root 2196 Dec 22 12:47 initial-setup-ks.cfg
[root@studentvm1 ~]#
```

Use Vim to edit the new file and add the code shown in Figure 10-1 to the file.

```
dnf check-update
mandb
grub2-mkconfig > /boot/grub2/grub.cfg
# reboot
```

***Figure 10-1.*** *The first version of our doUpdates script*

We have commented the reboot command so that the program will not reboot every time it is run. This will save time, and it serves as a reminder that we will eventually need to deal with the code that determines whether a reboot is required.

Without exiting Vim, open another root terminal session and run the program.

```
[root@studentvm1 ~]# ./doUpdates
```

In this initial version of our program, we have not started with the shebang ( #! ) which defines which shell to use to run this program if Bash is not the default shell. So let's add the shebang that defines Bash as the first line of the script. This just ensures that the program is always run in a Bash shell.

Figure 10-2 shows our script now. Run this program again but the results should be exactly the same.

```
#!/usr/bin/bash
#
dnf check-update
mandb
grub2-mkconfig > /boot/grub2/grub.cfg
# reboot
```

***Figure 10-2.*** *Adding the shebang ensures that the script always runs in a Bash shell*

## Add some logic

The first thing I added to my script was some basic logic that allowed me to skip around certain tasks. For now the actual updates are not actually performed. We will also do the same with rebooting. This will make further testing easier.

**EXPERIMENT 10-3**

First we need to define variables we shall call $Check and $doReboot and then add some logic around the **`dnf`** and **`reboot`** commands. We will initially set these variables so that the we do the check but not the actual updates and that the reboot is not performed. We should also start adding some comments.

I have also added a message that will print if the reboot is skipped. This helps test the one branch of the logic with some positive feedback. It will also be a nice verification that the logic is working when in actual use.

After adding the new code so that it looks like Figure 10-3, test the program and fix any errors that you might encounter.

```
#!/usr/bin/bash
#

################################################################################
# Initialize variables
################################################################################
Check=1
doReboot=0

################################################################################
# Main body of the program
################################################################################
# First we decide whether to do the updates or just check whether any are available

if [ $Check == 1 ]
then
   # Check for updates
   dnf check-update
fi

# Update the man database
mandb

# update the grub configuration
grub2-mkconfig > /boot/grub2/grub.cfg

# Reboot if necessary
if [ $doReboot == 1 ]
then
   reboot
else
   echo "Not rebooting."
fi
```

***Figure 10-3.*** *We have added some simple logic to control which tasks we want to perform*

For now these settings are built-in, but we will next look at ways to control program flow from the command line. This also does not look at both factors that would initiate a reboot. It only looks at the CLI option -r, but it is fine for now. It doesn't process any command-line options at this point; it only checks the variable.

## Limit to root

This program should be limited to usage by the root user. We can do this partially through ownership and permission settings, but we should also add some code to check for this. Remember that the root user ID (UID) is zero (0) and all other users have UIDs greater than zero.

**EXPERIMENT 10-4**

Add the code in Figure 10-4 to our program just below the variable initialization and before the main body of the program. This code checks the UID of the user account attempting to run the program. It only allows the root user to continue and dumps everyone else out with an error.

```
################################################################################
# Check for root
################################################################################
if [ `id -u` != 0 ]
then
   echo "You must be root user to run this program"
   exit
fi
```

***Figure 10-4.*** *Checking for root as the only authorized user*

Now test the program as root to ensure that it still works for root, then make a copy of the program in /tmp, and try to run it as the student user. You should first get a permissions error.

```
[student@studentvm1 tmp]$ ./doUpdates
-bash: ./doUpdates: Permission denied
```

As root, set the permissions for the copy in /tmp to 777 – which is never a good thing in reality. Then try to run it again as the student user.

```
[student@studentvm1 tmp]$ ./doUpdates
You must be root user to run this program
[student@studentvm1 tmp]$
```

This result is exactly what we want.

Of course a knowledgeable non-root user could modify the code if it is in /tmp with permissions of 777, but every bit of security we can build into our scripts helps deter the casual user from wreaking unexpected havoc.

## Add command-line options

We now have some logic in our program but no way to control it other than editing the variable settings in the code itself. That is not very practical so we need a way to set options at the command line. We also want to determine whether the kernel or glibc[2] is to be updated. It is always a good idea to reboot after one or both of those are updated.

Fortunately, Bash as a couple tools that we can use for this purpose. The **getops** command gets options from the command line and, along with **while** and the **case** structure, allows us to set variables or perform other tasks based on the options read from the command line.

**EXPERIMENT 10-5**

First we need to add some new variables. The revised variables section of our code now looks like Figure 10-5.

The original $doReboot variable will be set to true to cause a reboot if the user enters -r on the command line. The $NeedsReboot variable will be set to true if either the kernel or glibc is to be updated. The system will be rebooted only if both of these variables are true. The $UpdatesAvailable variable will be set to true if one or more updates are available from the Fedora repositories.

[2]The glibc package contains the general C libraries that are needed by almost every program that runs as part of the Linux operating system and application programs.

```
################################################################################
# Initialize variables
################################################################################
Check=1
doReboot=0
NeedsReboot=0
UpdatesAvailable=0
```

***Figure 10-5.*** *We added three new variables to enable better control of the doUpdates program*

Now we can add the code that allows us to capture the command options that are input at the CLI, evaluate them, and act accordingly. Figure 10-6 shows a very basic version of this. We will add more to this as we proceed.

The getops command gets the list of options the user entered on the command line such as **`doUpdates -c`**. It creates a list of options for the **`while`** command which loops until there are no more options left in the list. The **`case`** structure evaluates each possible option and executes a list of program statements for each valid option.

The two options we are adding now, -c and -r, are used to set variables in the case structure. If any invalid option is entered at the command line, the last case in the case structure executes. In this case the **`exit`** command exits from the program.

Notice that each case ends with a double semicolon. The esac statement ends the case structure, and the done statement closes out the while structure.

Enter the code in Figure 10-6 just below our test for the root user.

```
################################################################################
# Process the input options                                                    #
################################################################################
# Get the options
while getopts ":cr" option; do
   case $option in
      c) # Check option
           Check=1;;
      r) # Reboot option
           doReboot=1;;
     \?) # incorrect option
           echo "Error: Invalid option."
           exit;;
   esac
done
```

***Figure 10-6.*** *Getting the command-line options*

Before we proceed any further, some testing is needed. Let's first test for an invalid option. The -x is not a valid option so we should get the error message and the program should exit.

```
[root@studentvm1 ~]# ./doUpdates -x
Error: Invalid option.
[root@studentvm1 ~]#
```

Because we don't have real logic around the -r option, using it will cause your VM to reboot after doing the check for updates, updating the man pages, and generating the new grub configuration file. At the moment, this is the expected result.

```
[root@studentvm1 ~]# ./doUpdates -r
```

We now have the ability to control program flow using options on the command line. We will add more options and more logic around these existing options.

## Check for updates

We need to do a real check to see if updates are available and then determine whether a reboot is needed.

## EXPERIMENT 10-6

We need to add some code and make some logic changes to check for any available updates. Add the new variable, UpdatesFile, to the initialization section as shown in Figure 10-7.

```
################################################################################
# Initialize variables
################################################################################
Check=1
doReboot=0
NeedsReboot=0
UpdatesAvailable=0
UpdatesFile="/tmp/updates.list"
```

***Figure 10-7.*** *Add the new variable $UpdatesFile*

Delete the following comment line which is now obsolete.

```
# First we decide whether to do the updates or just check whether any are
available
```

We will also move our logic for the -c option into the "else" branch of this new if structure. So we can remove the code fragment in Figure 10-8 from our program.

```
if [ $Check == 1 ]
then
   # Check for updates
   dnf check-update
fi
```

***Figure 10-8.*** *Remove this code from the program*

Add the code in Figure 10-9 immediately after the option processing code. It checks whether updates are available at all while saving a file containing the list of updates that can be parsed for specific packages that require a reboot. This new code will exit the program if no updates are available.

```
################################################################################
# Are updates available? Just quit with message if not.
# RC from dnf check-update = 100 if available and 0 if none available.
# One side effect is to create list of updates that can be searched for
# items that trigger a reboot.
################################################################################
dnf check-update > $UpdatesFile
UpdatesAvailable=$?
if [ $UpdatesAvailable == 0 ]
then
   echo "Updates are NOT available for host $HOSTNAME at this time."
   exit
else
   echo "Updates ARE available for host $HOSTNAME."
   if [ $Check == 1 ]
   then
      exit
   fi
fi

# Temporary exit
exit
```

***Figure 10-9.*** *Testing to see if any updates are available and exit if not*

Notice that I have also added a temporary exit command so that we do not need to run any of the code beyond this new section. This saves time by not running code that has not yet been completed with all of the logic necessary. Also note the use of the Bash environment variable, $HOSTNAME, which always contains the name of the Linux host.

Testing this new bit of code results in the following. We will not be able to test the "then" branch of the new code until we have installed all of the current updates.

```
[root@studentvm1 ~]# ./doUpdates
Updates ARE available for host studentvm1.
[root@studentvm1 ~]#
```

## Is a reboot required?

Now that we know that updates are available, we can use the data in the file we created to determine whether any of the packages we have specified as making it a good idea to reboot are in the list. This is easily done.

However, even though a reboot might be a good thing to do, Linux is far more flexible than other operating systems which force one or more reboots during each update. We can put that Linux reboot off until it is more convenient, such as 02:00 AM or

over a weekend. To do that, we look at two variables, $NeedsReboot, which is determined by looking for the trigger packages that are being updated, and $doReboot, which is set from the command line with the -r option. The -r option is our way of maintaining control over what happens after the update itself is complete.

## EXPERIMENT 10-7

In this experiment we add a series of if statements to determine whether any of the packages that typically need a reboot are being updated. Add the code in Figure 10-10 below the code we added in Experiment 10-6 and just above the temporary exit code.

```
# Does the update include a new kernel
if grep ^kernel $UpdatesFile > /dev/null
then
   NeedsReboot=1
   echo "Kernel update for $HOSTNAME."
fi
# Or is there a new glibc
if grep ^glibc $UpdatesFile > /dev/null
then
   NeedsReboot=1
   echo "glibc update for $HOSTNAME."
fi
# Or is there a new systemd
if grep ^systemd $UpdatesFile > /dev/null
then
   NeedsReboot=1
   echo "systemd update for $HOSTNAME."
fi

# Temporary exit
exit
```

***Figure 10-10.*** *Checking for updates to packages that indicate the need for a reboot*

We also need to change the default entry on the -c (check) option from 1 to zero in the variable initialization settings.

After adding the code in Figure 10-10 and changing the initial value of the $Check variable to 0, run some tests to verify that it is correct and working as expected.

```
[root@studentvm1 ~]# ./doUpdates -c
Updates ARE available for host studentvm1.
[root@studentvm1 ~]# ./doUpdates
Updates ARE available for host studentvm1.
```

```
Kernel update for studentvm1.
glibc update for studentvm1.
systemd update for studentvm1.
[root@studentvm1 ~]#
```

Change the reboot logic at the bottom of the program to that in Figure 10-11. Note that we have made this fairly verbose, especially in the event of a failure. The "else" entry in the if structure is there in case none of the other expected logical combinations are met.

```
# Reboot if necessary
if [ $NeedsReboot == 0 ]
then
   echo
   echo "##########################################################"
   echo "A reboot is not required."
   echo "##########################################################"
elif [ $doReboot == 1 ] && [ $NeedsReboot == 1 ]
then
   reboot
elif [ $doReboot == 0 ] && [ $NeedsReboot == 1 ]
then
   echo
   echo "##########################################################"
   echo "A reboot is needed."
   echo "Be sure to reboot at the earliest opportunity."
   echo "##########################################################"
   echo
else
   echo
   echo "##########################################################"
   echo "An error has occurred and I cannot determine whether"
   echo "to reboot or not. Intervention is required."
   echo "##########################################################"
   echo
fi
```

***Figure 10-11.*** *Change the reboot logic code to this*

Now reboot after updates are installed only if the $NeedsReboot and $DoReboot variables are set to "true," that is, one (1).

We will test this code in later experiments after we remove the temporary exit.

## Adding a Help function

Shell functions are lists of Bash program statements that are stored in the shell environment and which can be executed like any other command by typing its name at the command line. Shell functions may also be known as procedures or subroutines depending upon which other programming language you might be using.

Functions are called in our scripts or from the CLI by using their names, just as we would for any other command. In a CLI program or a script, the commands in the function are executed when called and then the sequence of program flow returns to the calling entity and the next series of program statements in that entity is executed.

The syntax of a function is

```
FunctionName(){list}
```

Before we add our help function, let's explore how functions work.

### EXPERIMENT 10-8

Perform this experiment as the student user. Start by creating a simple function at the CLI. The function is stored in the shell environment for the shell instance in which it is created. We are going to create a function called "hw" which stands for hello world.

Enter the following code at the CLI and press **Enter**. Then enter hw as you would any other shell command.

```
[student@studentvm1 ~]$ hw(){ echo "Hi there kiddo"; }
[student@studentvm1 ~]$ hw
Hi there kiddo
[student@studentvm1 ~]$
```

OK, so I am a little tired of the standard "Hello world" we usually start with.

Now let's list all of the currently defined functions. There are a lot of them so I have shown just the new hw function.

```
[student@studentvm1 ~]$ declare -f | less
<snip>
hw ()
{
    echo "Hi there kiddo"
}
<snip>
```

Now let's remove that function because we do not need it any more. We can do that with the **unset** command.

```
[student@studentvm1 ~]$ unset -f hw ; hw
bash: hw: command not found
[student@studentvm1 ~]$
```

Verify that the function has been removed from the environment.

Now that we know a little about how functions work we can add our help facility.

## EXPERIMENT 10-9

As root again, add the function in Figure 10-12 to the doUpdates script. Place it after the shebang line and before the variable initialization section.

```
################################################################################
# Help function
################################################################################
Help()
{
   echo "doUpdates"
   echo ""
   echo "Installs all available updates from Fedora repositories. Can reboot"
   echo "after updates if certain packages are updated. Those packages are:"
   echo ""
   echo "1. The kernel"
   echo "2. glibc"
   echo "3. systemd"
   echo ""
   echo "Syntax: doUpdates [-c|h|r]"
   echo "Options:"
   echo "-c   Check whether updates are available and exit."
   echo "-h   Print this Help and exit."
   echo "-r   Reboot if specific trigger packages are updated"
   echo ""
} # end of Help()
```

***Figure 10-12.*** *The Help() function*

Now add an option for help to the **case** statement. *Be sure to add the "h" to the* ***getops*** *option string*. Figure 10-13 shows the revised option processing code that includes the new "h" option.

```
# Get the options
while getopts ":hcr" option; do
   case $option in
      c) # Check option
           Check=1;;
      h) # Help function
           Help
           exit;;
      r) # Reboot option
           doReboot=1;;
     \?) # incorrect option
           echo "Error: Invalid option."
           exit;;
   esac
done
```

***Figure 10-13.*** *Add the Help function to the option processing code*

We now test again and fix any errors we find. I neglected to add the double semicolon (;;) at the end of the help function processing so I received the following error.

```
[root@studentvm1 ~]# ./doUpdates -h
./doUpdates: line 55: syntax error near unexpected token `)'
./doUpdates: line 55: `      r) # Reboot option'
```

After fixing the problem, I reran the test and the Help function worked as expected.

```
[root@studentvm1 ~]# ./doUpdates -h
doUpdates

Installs all available updates from Fedora repositories. Can reboot
after updates if certain packages are updated. Those packages are:

1. The kernel
2. glibc
3. systemd

Syntax: doUpdates [-c|h|r]
Options:
-c   Check whether updates are available and exit.
-h   Print this Help and exit.
-r   Reboot if specific trigger packages are updated
```

Be sure to test using the -c option to ensure that nothing else is broken. For now we will skip testing the -r (reboot) option for expediency.

## Finishing the script

So there is now only one thing we need to do in order to finish this script – at least for now. We need to add the code that actually performs the update and remove the temporary exit. We also need to add some logic to the reboot at the end of the program.

**EXPERIMENT 10-10**

We need to do three things before our program is ready. First, remove the two lines in Figure 10-14.

```
# Temporary exit
exit
```

***Figure 10-14.*** *Remove the temporary exit code*

Then add the code in Figure 10-15 to replace what we just deleted.

```
# Perform the update
dnf -y update
```

***Figure 10-15.*** *...and add this code in its place*

So now we are ready to test version 0.0.1 of our program. But before we do that, let's discuss testing in more detail.

# About testing

*There is always one more bug.*

—Lubarsky's Law of Cybernetic Entomology

Lubarsky – whoever that might be – is correct. We can never find all of the bugs in our code. For every bug I find, there always seems to be another that crops up usually at a very inopportune time.

Testing is not just about programs. It is also about verification that problems – whether caused by hardware, software, or the seemingly endless ways that users can find to break things – that we are supposed to have resolved actually have been. These problems can be with application or utility software we wrote, system software, applications, and hardware. Just as importantly, testing is also about ensuring that the code is easy to use and the interface makes sense to the user.

Following a well-defined procedure when writing and testing shell scripts can contribute to consistent and high-quality results. My procedures are simple:

1. Create a simple test plan.
2. Start testing right at the beginning of development.
3. Perform a final test when the code is complete.
4. Move to production and test more.

You have undoubtedly noticed that we have run multiple tests at every step of creating this program. One of the tenets of *The Linux Philosophy for SysAdmins* is to "Test Early, Test Often."[3]

## Testing in production

Huh – what?

> *Not until a program has been in production for at least six months will the most harmful error be discovered.*
>
> —Troutman's Programming Postulates

Yes, testing in production is now considered normal and desirable. Having been a tester myself, this actually does seem reasonable. "But wait! That's dangerous," you say. My experience is that it is no more dangerous than extensive and rigorous testing in a dedicated test environment. In some cases, there is no choice because there *is* no test environment – only production.

SysAdmins are no strangers to the need to test new or revised scripts in production. Any time a script is moved into production that becomes the ultimate test. The production environment itself constitutes the most critical part of that test. Nothing that can be dreamed up by testers in a test environment can fully replicate the true production environment.

The allegedly new practice of testing in production is just the recognition of what we SysAdmins have known all along. The best test is production – so long as it is not the only test.

[3]op cit, Chapter 11

## Fuzzy testing

This is another of those buzzwords that caused me to roll my eyes when I first heard it. I learned that its essential meaning is simple – have someone bang on the keys until something happens and see how well the program handles it. But there really is more to it than that.

Fuzzy testing is a bit like the time my son broke the code for a game in less than a minute with his random input. That pretty much ended my attempts to write games for him.

Most test plans utilize very specific input that generates a specific result or output. Regardless of whether the test is for a positive or negative outcome as success, it is still controlled and the inputs and results are specified and expected, such as a specific error message for a specific failure mode.

Fuzzy testing is about dealing with randomness in all aspects of the test such as starting conditions, very random and unexpected input, random combinations of options selected, low memory, high levels of CPU contention with other programs, multiple instances of the program under test, and any other random conditions that you can think of to be applied to the tests.

I try to do some fuzzy testing right from the beginning. If the Bash script cannot deal with significant randomness in its very early stages, then it is unlikely to get better as we add more code. This is also a good time to catch these problems and fix them while the code is relatively simple. A bit of fuzzy testing at each stage of completion is also useful in locating problems before they get masked by even more code.

After the code is completed, I like to do some more extensive fuzzy testing. Always do some fuzzy testing. I have certainly been surprised by some of the results I have encountered. It is easy to test for the expected things, but users do not usually do the expected things with a script.

## Testing the script

**EXPERIMENT 10-11**

Before we start this final test, let's take a snapshot of the StudentVM1 host so that we can return to a known working state in which there are definitely updates to be performed and some that require a reboot. This will give us a good bit of flexibility if we have some fixes to be made to our code. We can boot the snapshot, make the changes, and test again, as many times as necessary to get our script working correctly.

Oh, yes, there are always improvements and functional changes we might want to make that would benefit from having this capability. Hint, hint.

Save the doUpdates program, power off StudentVM1, and make the snapshot. You can refer to Volume 1, Chapter 5, for details of creating the snapshot. I added the following text to the snapshot description. "Near end of Chapter 10. Can be used to roll back to a state where updates are available." Save the snapshot.

---

**Tip** From this point on, when you make changes to the doUpdates script, make a copy of it on an external USB thumb drive or other device. Then you can restart the latest snapshot in which updates are available, copy the latest version of the script into /root, and rerun the test. Using this procedure, you can make changes and rerun the test as many times as necessary.

---

Now reboot and proceed with the rest of this experiment.

Now we test, but we will not immediately test the update portion of this code. We will start our test using the options that do not lead down that execution path. That way we know those options have not been broken by our latest additions.

```
[root@studentvm1 ~]# ./doUpdates -c
Updates ARE available for host studentvm1.
[root@studentvm1 ~]# ./doUpdates -h
doUpdates

Installs all available updates from Fedora repositories. Can reboot
after updates if certain packages are updated. Those packages are:

1. The kernel
2. glibc
3. systemd

Syntax: doUpdates [-c|h|r]
Options:
-c   Check whether updates are available and exit.
-h   Print this Help and exit.
-r   Reboot if specific trigger packages are updated

[root@studentvm1 ~]#
```

It is now time to test the primary function of our script. We will do this first with a manual reboot. We will then reboot StudentVM1 to the last snapshot and run the script again and do a programmatic reboot using the -r option at the command line.

First run the script with no options.

```
[root@studentvm1 ~]# time ./doUpdates
```

Depending upon how many packages need to be updated, the speed of your VM, and the speed of your Internet connection, this process may take a long time. On my VM there were 622 packages to update, and it took just over 47 minutes. Jason, my technical reviewer for this volume, let me know that it only took 10 minutes for his updates to complete. It will depend upon how recently you last updated and how many new updates there are.

Do a manual reboot to verify that the VM can at least do that, and then power it off.

So we have done one test of our Bash script. We now need to roll back to the last snapshot and retest using the -r option. Let's first do the rollback.

## EXPERIMENT 10-12

This experiment should be performed in the GUI of the host system for the VM. These instructions guide you through reverting to the most recent snapshot.

Open the VirtualBox window and select StudentVM1. Click the menu icon on the right side of the StudentVM1 bar, then select **Snapshots**. Select the most recent snapshot which should have been taken earlier in this chapter. Right-click it and then click **Restore**. Uncheck the box "Create a snapshot of the current machine state," then click Restore.

Hover over the "Current state" line. A little text box opens with the message "The current state is identical to the state stored in the current snapshot." This is exactly right. Be sure that "Current state" is selected and reboot the VM.

You can check to verify that you have successfully reverted by running **`dnf check-update`** to list all of the available updates. You could also run **`uname -a`** both before and after rollbacks and compare the release levels.

We can now finish our testing.

**EXPERIMENT 10-13**

As the root user, run the doUpdates program again, this time using the -r option.

```
[root@studentvm1 ~]# ./doUpdates -r
```

This should install all of the available updates, rebuild the man database, generate a new GRUB configuration file, and reboot the VM. After the reboot, log in and run some basic tests to verify that the VM is working and responding to simple commands.

Power off, revert to the most recent snapshot, and reboot the VM.

# Making it better

Guess what! Our script is not really finished.

Yes, it does the job we specified for it so, technically, it is complete in that sense, but there are some ways in which it can be improved. As in real life, additional requirements may be discovered.

When doUpdates is run with the -c option, it simply indicates that there are updates available but not whether a reboot would be required; it might also be nice to know which packages are being updated that prompt the reboot.

**EXPERIMENT 10-14**

Add or modify the code of the doUpdates Bash script to meet the following additional requirement:
When run with the -c option, the program should also list which packages require the reboot and should state clearly that a reboot will be required.

Test extensively.

# Licensing

One of the best ways I know to give back to the open source community that provides us with all of these incredible programs like the GNU Utilities, the Linux kernel, LibreOffice, WordPress, and thousands more is to open source our own programs and scripts with an appropriate license.

Just because we write a program and we believe in open source and agree that our programs should be open source code does not make it so. As SysAdmins we do write a lot of code, but how many of us ever consider the issue of licensing our own code? We must make the choice and explicitly state that the code is open source and under which license it is being distributed. Without this critical step, the code we create is subject to becoming fettered with proprietary licenses so that the community cannot take advantage of our work.

We should include the GPLv2 (or your other preferred) license header statement as a command-line option that would print the license header on the terminal. When distributing code, I also recommend that we make it practice to include a text copy of the entire license with the code – which is a requirement of some licenses.

I find it very interesting that in all of the books I have read and all of the classes I have attended, not once did any of them tell me to be sure to license any code I wrote in my tasks as a SysAdmin. All of these sources completely ignored the fact that SysAdmins write code too. Even in the conference sessions on licensing that I have attended, the focus was on application code, kernel code, or even GNU-type utilities. None of the presentations even so much as hinted at the fact that we SysAdmins write huge amounts of code to automate our work or that we should even consider licensing it in any way. Perhaps you have had a different experience, but this has been mine. At the very least, this frustrates me; at the most it angers me.

We devalue our code when we neglect to license it. Most of us SysAdmins don't even think about licensing, but it is important if we want our code to be available to the entire community. This is neither about credit nor is it about money. This is about ensuring that our code is now and always will be available to others in the best sense of free and open source.

Eric Raymond, author of 2003 book *The Art of Unix Programming*, writes that in the early days of computer programming and especially in the early life of Unix, sharing code was a way of life.[4] In the beginning this was simply reusing existing code. With the advent of Linux and the open source licensing, this became much easier. It feeds the needs of system administrators to be able to legally share and reuse open source code.

[4]Raymond, Eric S., *The Art of Unix Programming*, Addison-Wesley (2004), 380, ISBN 0-13-13-142901-9

Raymond states, "Software developers want their code to be transparent. Furthermore, they don't want to lose their toolkits and their expertise when they change jobs. They get tired of being victims, fed up with being frustrated by blunt tools and intellectual property fences and having to repeatedly reinvent the wheel."[5] This statement also applies to SysAdmins.

I read an interesting article recently, "The source code is the license,[6]" that helps explain the reasoning behind this.

So let's add a licensing statement to our code that can be displayed with a new option.

## EXPERIMENT 10-15

As root, edit the doUpdates program. First let's add the function shown in Figure 10-16 immediately after the Help function.

```
################################################################################
# Print the GPL license header                                                 #
################################################################################
gpl()
{
   echo
   echo "################################################################################"
   echo "#  Copyright (C) 2019  David Both                                             #"
   echo "#  http://www.both.org                                                        #"
   echo "#                                                                             #"
   echo "#  This program is free software; you can redistribute it and/or modify       #"
   echo "#  it under the terms of the GNU General Public License as published by       #"
   echo "#  the Free Software Foundation; either version 2 of the License, or          #"
   echo "#  (at your option) any later version.                                        #"
   echo "#                                                                             #"
   echo "#  This program is distributed in the hope that it will be useful,            #"
   echo "#  but WITHOUT ANY WARRANTY; without even the implied warranty of             #"
   echo "#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the              #"
   echo "#  GNU General Public License for more details.                               #"
   echo "#                                                                             #"
   echo "#  You should have received a copy of the GNU General Public License          #"
   echo "#  along with this program; if not, write to the Free Software                #"
   echo "#  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA  #"
   echo "################################################################################"
   echo
} # End of gpl()
```

***Figure 10-16.*** *Add a function that prints the GPL V2 license header*

[5]Ibid.

[6]Scott K Peterson, *The source code is the license*, Opensource.com, https://opensource.com/article/17/12/source-code-license

Now we need to add an option to the option processing code. Since this is the GPL, I chose "g." Figure 10-17 shows the revised option processing code. I like to place the new case sections in alphabetical order to make them a bit easier to find when performing maintenance.

```
################################################################################
# Process the input options                                                    #
################################################################################
# Get the options
while getopts ":ghcr" option; do
   case $option in
      c) # Check option
           Check=1;;
      g) # display the GPL header
           gpl
           exit;;
      h) # Help function
           Help
           exit;;
      r) # Reboot option
           doReboot=1;;
     \?) # incorrect option
           echo "Error: Invalid option."
           exit;;
   esac
done
```

***Figure 10-17.*** *Add the g option to the case structure*

Finally, we need to add a line to the Help function. Add the line shown in Figure 10-18 in the options section. I like to try and keep these in alphabetical order too, but you can place them in any order that makes sense to you.

```
  echo "-g   Print the GPL license notification."
```

***Figure 10-18.*** *Add a line to the Help function that describes the -g option*

Now test this new option and make sure nothing else has been broken.

# Automated testing

Testing is important and, if your Bash programs are more than just a few lines, you may want to explore some automated testing tools. Although I have worked as a tester in one of my jobs and used Tcl/Expect to automate our application testing, that type of tool is way overkill for the SysAdmin. We work on tight schedules with little or no time to use complex, and possibly expensive, tools, whether open source or not.

I have found one interesting tool that you might wish to explore. BATS[7] is a tool specifically designed for testing Bash programs. There are undoubtedly other tools that can be useful to testing Bash program. Most of the time manual testing works just fine for the Bash programs I write.

# Security

This program should only be run by root and will fail if any other user tries to run it. We have looked at security a bit already and seen the effect of using permissions of 750 and 777. We have also looked at using a bit of code to exit from the program if the UID of the account attempting to run the program is not zero, that is, the root user.

Root is the only user that needs access to this program so it can be placed in /root/bin. This also makes it available at all times even if other filesystems are not mounted. Placing it in root's own ~/bin directory makes the program inaccessible to non-root users.

**EXPERIMENT 10-16**

Perform this experiment as the root user.

If the doUpdates script is currently open for editing, close it. Ensure that /root is the PWD. Create the /root/bin directory because it is not created by default. Move the doUpdates file to /root/bin. Run the program to test its new location.

```
[root@studentvm1 ~]# cd ; mkdir bin ; mv doUpdates bin ; doUpdates -h
```

[7]Opensource.com, *Testing Bash with BATS*, https://opensource.com/article/19/2/testing-bash-bats

# Additional levels of automation

Now I have this incredibly wonderful and useful script. I have copied it to /root/bin on all of my computers. All I have to do now is run it at appropriate times on each of my Linux hosts to do the updates. I can do this by using SSH to log in to each host and run the program.

But wait! There's more! Have I told you yet how absolutely cool SSH is?

The **ssh** command is a secure terminal emulator that allows one to log in to a remote computer to access a remote shell session and run commands. So I can log in to a remote computer and run the `doUpdates` command on the remote computer as shown in Figure 10-19. The results are displayed in the ssh terminal emulator window on my localhost. The Standard Output (STDOUT) from the command is displayed on my terminal window.

For this course we do not have a second VM to work with so we cannot do lab projects for this. I will describe the steps and the CLI programs and scripts will be shown as figures. We will look at SSH in more detail in the next course in this series, *Advanced Linux System and Server Administration.*

```
ssh hostname doUpdates -r
```

***Figure 10-19.*** *Using SSH to perform remote updates with the doUpdates script*

That part is trivial and everyone does that. But the next step is a bit more interesting. Rather than maintain a terminal session on the remote computer, I can simply use a command on my local computer such as that in Figure 10-19 to run the command on the remote computer with the results being displayed on the localhost. This assumes that SSH public/private keypairs[8] (PPKP) are in use and I do not have to enter a password each time I issue a command to the remote host.

So now I run a single command on my localhost that sends a command through the SSH tunnel to the remote host. OK, that is good, but what does it mean?

It means that what I can do for a single computer, I can also do for several – or several hundred. The Bash command-line program in Figure 10-20 illustrates the power I now have.

[8]How to Forge, www.howtoforge.com/linux-basics-how-to-install-ssh-keys-on-the-shell

```
for I in host1 host2 host3 ; do ssh $I doUpdates -r ; done
```

***Figure 10-20.*** *Using a simple CLI program to perform remote updates on multiple computers*

This little command-line program is now doing the type of function that advanced tools like Ansible[9] can do. It is important to understand automation with Bash in order to fully understand and appreciate the role and capabilities of tools like Ansible.

Think we're done? No, we are not! The next step is to create a short Bash script of this CLI program so we don't have to retype it every time we want to install updates on our hosts. This does not have to be fancy; the script can be as simple as the one in Figure 10-21.

```
for I in host1 host2 host3 ; do ssh $I doUpdates -r ; done
```

***Figure 10-21.*** *This Bash script contains the command-line program that runs the doUpdates program on three remote hosts*

This script could be named "updates" or something else depending on how you like to name scripts and what you see as its ultimate function. I think we should call this script "`doit`". Now we can just type a single command and run a smart update program on as many hosts as we have in the list of the `for` statement. Our script should be located in the /usr/local/bin directory so it can be easily run from the command line.

Our little **`doit`** script looks like it could be the basis for more general application. We could add more code to **`doit`** that would enable it to take arguments or options such as the name of a command to run on all of the hosts in the list. This enables us to run any command we want on a list of hosts, and our command to install updates might be **`doit doUpdates -r`** or **`doit myprogram`** to run "myprogram" on each host.

The next step might be to take the list of hosts out of the program itself and place them in a doit.conf file located in /usr/local/etc – again in compliance with the Linux FHS. That command would look like Figure 10-22 for our simple `doit` script. Notice the back tics (`) that create the list used by the `for` structure from the results of the `cat` command.

[9]Jonathan Lozada De La Matta, *A sysadmin's guide to Ansible: How to simplify tasks*, https://opensource.com/article/18/7/sysadmin-tasks-ansible, Opensource.com

```
#!/bin/bash
for I in `cat /usr/local/etc/doit.conf` ; do ssh $I doUpdates ; done
```

***Figure 10-22.*** *We have now added a simple external list that contains the hostnames on which the script will run the specified command*

By keeping the list of hosts separate, we can allow non-root users to modify the list of hosts while protecting the program itself against modification. It would also be easy to add an -f option to the `doit` program so that the users could specify the name of a file containing their own list of hosts on which to run the specified program.

Finally, we might want to set this up as a cron job so that we don't have to remember to run it on whatever schedule we want. Setting up cron jobs is worthy of its own section in this chapter so that is coming up next.

# Chapter summary

Computers are designed to automate various mundane tasks, and why should that not also be applied to the SysAdmin's work? We lazy SysAdmins use the capabilities of the computers on which we work to make our jobs easier. Automating everything that we possibly can means that the time we free up by creating that automation can now be used to respond to some real or perceived emergency by others, especially by the PHB. It can also provide us with time to automate even more.

If you reflect on what we have done in this chapter, you can see that automation is not merely about creating a program to perform every task. It can be about making those programs flexible so that they can be used in multiple ways such as the ability to be called from other scripts and to be called as a cron job.

My programs almost always use options to provide flexibility. The `doit` program used in this chapter could easily be expanded to be more general than it is while still remaining quite simple. It could still do one thing well if its objective were to run a specified program on a list of hosts.

My shell scripts did not just spring into existence with hundreds or thousands of lines. In most cases they start as a single ad hoc command-line program. I create a shell script from the ad hoc program. Then another command-line program is added to the short script, then another. As the short script becomes longer, I add comments, options, and a help feature.

Then, sometimes, it makes sense to make a script more general so that it can handle more cases. In this way the `doit` script becomes capable of "doing it" for more than just a single program that does updates.

As far as this chapter is concerned, this script is complete and it can be used in production. But as you use it, you will undoubtedly find more refinements you might want to make. Clearly you should feel free to do so.

# Exercises

Complete the following exercises to finish this chapter:

1. List at least three advantages to creating and using scripts.
2. What happens when you change the string comparison operator (==) to the numeric operator (-eq)?
3. Running **`doUpdates`** displays a list of the packages being updated for which a reboot is required. It does not list those packages if the -c option is used. Revise the script so that these three packages are also listed with the -c option – assuming that they are in the list of updates.
4. There is at least one set of conditions under which we have not tested our doUpdates shell program. Devise and implement a way to test the doUpdates program in that circumstance.
5. Add one line of code to the "Incorrect option" case stanza that will display the help text before exiting.

CHAPTER 11

# Time and Automation

## Objectives

In this chapter you will learn

- To use chrony to maintain accurate system time
- To create a crontab file for root using the crontab command
- To add necessary statements to the crontab file that set up the environment for cron jobs
- To interpret the time specifications for cron and configure cron jobs to run at different recurring intervals
- To create cron jobs using crontab
- To create cron jobs for hourly, daily, weekly, and monthly periods
- To use the at command to run scripts or commands once at a specific time in the future

## Introduction

In previous chapters we have looked at some examples of using command-line programs and Bash scripts to automate tasks we perform as SysAdmins. All of that is well and good, but what happens when tasks need to be performed at times that are not convenient for us as humans? For example, if we do backups at 01:01 AM every day or run a maintenance script at 03:00 AM every Sunday, I do not want to get out of bed to perform those tasks.

Linux provides multiple tools and ways in which we can use those tools to run those tasks at specified times in the future, repeating as needed or just for a one-time occurrence. However, keeping accurate time is critical to ensuring that scheduled jobs run at the correct times.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_11

# Keeping time with chrony

*Does anybody really know what time it is? Does anybody really care?*

—Chicago, 1969

Perhaps that rock group didn't care what time it was, but our computers really need to know the exact time. Timekeeping is very important to computer networks. In banking, stock markets, and other financial businesses, transactions must be maintained in the proper order and exact time sequences are critical for that. For SysAdmins and DevOps following the trail of email through a series of servers or determining the exact sequence of events using log files on geographically dispersed hosts can be much easier when exact times are kept on the computers in question.

I used to work at one organization that received over 20 million emails per day and which had four servers just to accept and do a basic filter on the incoming flood of email. From there, emails were sent to one of four more servers to perform more complex anti-spam assessments and then deliver the email to one of several additional servers where the messages were placed in the correct inboxes. At each layer, the emails would be sent to one of the servers at the next level selected only by the randomness of round-robin DNS. Sometimes we needed to trace the arrival of a new message through the system until we could determine where it "got lost," according to the pointy haired bosses. We had to do this with frightening regularity.

Most of that email turned out to be spam. Some people actually complained that their [Joke, cat pic, recipe, inspirational saying, and a few even more strange emails]-of-the-day was missing and asked us to find it. We did reject those opportunities.

Our email searches, as well as other transactional searches, were all aided by log entries with timestamps that – today – can resolve down to the nanosecond in even the slowest of modern Linux computers. In very high-volume transaction environments, the difference of a few microseconds of difference in the system clocks can mean thousands of transactions to be sorted through in order to find the correct ones.

## The NTP server hierarchy

NTP is the Network Time Protocol, and it is used by computers worldwide to synchronize their times with Internet standard reference clocks via a hierarchy of NTP servers. The NTP server hierarchy is built in layers called strata. Each stratum is a layer

of NTP servers. The primary servers are at stratum 1, and they are connected directly to various national time services at stratum 0 via satellite, radio, or even modems over phone lines in some cases. Those time services at stratum 0 may be an atomic clock, a radio receiver that is tuned to the signals broadcast by an atomic clock, or a GPS receiver using the highly accurate clock signals broadcast by GPS satellites.

To prevent time requests from time servers lower in the hierarchy, that is with a higher stratum number, from overwhelming the primary reference servers, there are several thousand public NTP stratum 2 servers that are open and available for all to use. Many users and organizations, myself included, with large numbers of their own hosts that need an NTP server, set up their own time servers so that only one localhost actually accesses the stratum 2 time servers. The remaining hosts in our networks are all configured to use the local time server which, in my case, is a stratum 3 server.

## NTP choices

The original NTP daemon, ntpd, has been joined by a newer one, chronyd. Both perform the task of keeping the time of the localhost synchronized with the time server. Both services are available and I have seen nothing to indicate that this will change any time soon.

Chrony has some features which make it the better choice for most environments. Some of the primary advantages of using chrony are shown in this list:

- Chrony can synchronize to the time server much faster than ntp. This is good for laptops or desktops that do not run constantly.
- It can compensate for fluctuating clock frequencies such as when a host hibernates or enters a sleep mode or when the clock speed varies due to frequency stepping that slows clock speeds when loads are low.
- It handles intermittent network connections and bandwidth saturation.
- It adjusts for network delays and latency.

- After the initial time sync is accomplished, Chrony never steps the clock. This ensures stable and consistent time intervals for many system services and applications that require that.
- Chrony can work even without a network connection of any type. In this case the localhost or server could be updated manually.

Both the ntp and chrony RPM packages are available from standard Fedora repositories. It is possible to install both and switch between them, but modern releases of Fedora, CentOS, and RHEL have moved from NTP to Chrony as the default timekeeping implementation. I have found that Chrony works well, provides a better interface for the SysAdmin, and presents much more information and increases control. I see no reason to use the old NTP service when Chrony is so much better.

So just to make it clear, NTP is a protocol that is implemented with either NTP or Chrony. We will explore only Chrony for both client and server configuration on a Fedora host. Configuration for current releases of CentOS and RHEL is the same.

## Chrony structure

The Chrony daemon, chronyd, runs in the background and monitors the time and status of the time server specified in the chrony.conf file. If the local time needs to be adjusted, chronyd does so smoothly without the programmatic trauma that would occur if the clock were to be instantly reset to a new time.

Chrony also provides the chronyc tool that allows us to monitor the current status of Chrony and to make changes if necessary. The chronyc utility can be used as a command that accepts sub-commands, or it can be used as an interactive text-mode program. We will use it both ways in this article.

## Client configuration

The NTP client configuration is simple and requires little or no change. The NTP server can be defined by the SysAdmin during the Linux installation, or it can be provided by the DHCP server at boot time. The default /etc/chrony.conf file shown in its entirety in Figure 11-1 requires no alterations to work properly as a client. For Fedora, Chrony uses the Fedora NTP pool. CentOS and RHEL also have their own NTP server pools. Like many Red Hat-based distributions, the configuration file is well commented.

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
pool 2.fedora.pool.ntp.org iburst

# Record the rate at which the system clock gains/losses time.
driftfile /var/lib/chrony/drift

# Allow the system clock to be stepped in the first three updates
# if its offset is larger than 1 second.
makestep 1.0 3

# Enable kernel synchronization of the real-time clock (RTC).

# Enable hardware timestamping on all interfaces that support it.
#hwtimestamp *

# Increase the minimum number of selectable sources required to adjust
# the system clock.
#minsources 2

# Allow NTP client access from local network.
#allow 192.168.0.0/16

# Serve time even if not synchronized to a time source.
#local stratum 10

# Specify file containing keys for NTP authentication.
keyfile /etc/chrony.keys

# Get TAI-UTC offset and leap seconds from the system tz database.
leapsectz right/UTC

# Specify directory for log files.
logdir /var/log/chrony

# Select which information is logged.
#log measurements statistics tracking
```

***Figure 11-1.*** *The default chrony.conf configuration file*

Let's look at the current status of NTP on our student virtual machines.

## EXPERIMENT 11-1

Perform this experiment as the root user.

The **chronyc** command when used with the tracking sub-command provides statistics that tell us how far off the local system is from the reference server.

```
[root@studentvm1 ~]# chronyc tracking
Reference ID    : C0A80034 (yorktown.both.org)
Stratum         : 5
Ref time (UTC)  : Fri May 17 19:36:00 2019
System time     : 0.000000017 seconds fast of NTP time
Last offset     : -0.000012977 seconds
RMS offset      : 0.000012977 seconds
Frequency       : 0.035 ppm fast
Residual freq   : -2.894 ppm
Skew            : 0.046 ppm
Root delay      : 0.031905096 seconds
Root dispersion : 0.008672087 seconds
Update interval : 2.0 seconds
Leap status     : Normal
[root@studentvm1 ~]#
```

The Reference ID in the first line of the result is the server to which our host is synchronized. That server is a server in my own network which was last contacted by our host at May 17, 19:36:00 2019. The rest of these lines are described in the chronyc(1) man page. The stratum line indicates which stratum our local VM is at so the yorktown host is at stratum 4.

The other sub-command I find interesting and useful is sources which provides information about the time sources configured in chrony.conf.

```
[root@studentvm1 ~]# chronyc sources
210 Number of sources = 5
```

```
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* yorktown.both.org           4   6    17     4   -775us[-1085us] +/-   25ms
^- 192.138.210.214             2   6    17     3  +1573us[+1573us] +/-   30ms
^- ntp1.wiktel.com             1   6    17     3   -415us[ -415us] +/-   36ms
^- dev.smatwebdesign.com       3   6    17     4  -4245us[-4245us] +/-  101ms
^+ time.tritn.com              2   6    17     5  +1227us[ +918us] +/-   38ms
[root@studentvm1 ~]#
```

The first source in the list is the time server that I set up for my personal network. The rest were provided by the pool. Even though my NTP server does not appear in the Chrony configuration file shown earlier, the DHCP server provides this IP address for the NTP server. Note that the "S" column – Source State – indicates that the server with an asterisk (*) in that line is the one to which our host is currently synchronized. This is consistent with the data from the tracking sub-command.

Note that the -v option provides a nice description of the fields in this output.

```
[root@studentvm1 ~]# chronyc sources -v
210 Number of sources = 5

  .-- Source mode  '^' = server, '=' = peer, '#' = local clock.
 / .- Source state '*' = current synced, '+' = combined , '-' = not combined,
| /   '?' = unreachable, 'x' = time may be in error, '~' = time too variable.
||                                                 .- xxxx [ yyyy ] +/- zzzz
||      Reachability register (octal) -.           |  xxxx = adjusted offset,
||      Log2(Polling interval) --.      |          |  yyyy = measured offset,
||                                \     |          |  zzzz = estimated error.
||                                 |    |           \
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* yorktown.both.org           4   6   177    45   +905us[ +160us] +/-   25ms
^+ 192.138.210.214             2   6   177    44  -3770us[-3770us] +/-   35ms
^? ntp1.wiktel.com             0   6     0     -     +0ns[   +0ns] +/-    0ns
^- dev.smatwebdesign.com       3   6   177    46  +1955us[+1955us] +/-   93ms
^+ time.tritn.com              2   6   177    48  -2290us[-3034us] +/-   43ms
```

If I wanted my own server to be the preferred reference time source for this host, I would add the following line to the /etc/chrony.conf file. I usually place this line just above the first pool server statement near the top of the file. There is no special reason for this except that I like to keep the server statements together. It would work just as well at the bottom of the file and I have done that on several hosts. This configuration file is not sequence sensitive.

```
server 192.168.0.51 iburst prefer
```

The "prefer" option marks this as the preferred reference source. As such, this host will always be synchronized with this reference source so long as it is available. You could also use the fully qualified hostname for a remote reference server or the hostname only without the domain name for a local reference time source so long as the search statement is set in the /etc/resolv.conf file. I prefer the IP address to ensure that the time source is accessible even if DNS is not working. In most environments the server name is probably the better option because NTP will continue to work even if the IP address of the server is changed.

You may not have a specific reference source with which you want to synchronize so it is fine to use the defaults.

## chronyc as an interactive tool

I mentioned near the beginning of this section that chronyc can be used as an interactive command tool. Let's explore that.

**EXPERIMENT 11-2**

Perform this experiment as root. Let's look at the **chronyc** command in more detail. Simply run the command without a sub-command and you get a chronyc command prompt.

```
[root@studentvm1 ~]# chronyc
chrony version 3.4
Copyright (C) 1997-2003, 2007, 2009-2018 Richard P. Curnow and others
chrony comes with ABSOLUTELY NO WARRANTY.  This is free software, and
you are welcome to redistribute it under certain conditions.  See the
GNU General Public License version 2 for details.

chronyc>
```

Now you can enter just the sub-commands. Try using the tracking, ntpdata, and sources sub-commands. The chronyc command line allows command recall and editing for chronyc sub-commands. You can use the help sub-command to get a list of possible commands and their syntax.

One thing I like to do after my client computers have synchronized with the NTP server is to set the system hardware clock from the system (OS) time using the following system command. Note that it is not a chronyc command.

```
[root@studentvm1 ~]# /sbin/hwclock --systohc
```

This command can be added as a cron job, as a script in cron.daily, or as a systemd timer to keep the hardware clock synced with the system time.

Chrony is a powerful tool for synchronizing the times of client hosts whether they are all on the local network or scattered around the globe. It is easy to configure because, despite the large number of configuration options available, only a few are required in most circumstances.

Chrony and NTP (the old service) both use the same configuration, and the files' contents are interchangeable. The man pages for chronyd, chronyc, and chrony.conf contain an amazing amount of information that can help you get started or learn about some esoteric configuration option.

## Using cron for timely automation

There are many tasks that need to be performed off-hours when no one is expected to be using the computer or, even more importantly, on a regular basis at specific times. In this chapter we explore the cron service and how to use it.

I use the cron service to schedule obvious things like regular backups that occur every day at 01:01 AM. I also do a couple less obvious things. All of my many computers have their system times, that is, the operating system time, set using NTP – the Network Time Protocol. NTP sets the system time; it does not set the hardware time which can drift and become inaccurate. I use cron to run a command that sets the hardware time using the system time. I also have a bash program I run early every morning that creates a new "message of the day" (MOTD) on each computer that contains information such as disk usage that should be current in order to be useful. Many system processes use cron to schedule tasks as well. Services like logwatch and rkhunter all use the cron service to run programs every day.

## The crond daemon

The crond daemon is the background service that enables cron functionality. The cron service checks for files in the /var/spool/cron and /etc/cron.d directories and the /etc/anacrontab file. The contents of these files define cron jobs that are to be run at various intervals.

The individual user cron files are located in /var/spool/cron, and system services and applications generally add cron job files in the /etc/cron.d directory. The /etc/anacrontab file is a special case that will be covered a bit further on in this chapter.

## crontab

Each user, including root, can have a crontab file. The term crontab derives from the Greek word chronos, for time, and the term table, because the file is a table of tasks set to perform at specific times and days.

---

**Note** The terms cron file and crontab file are sometimes used interchangeably.

---

By default, no file exists, but using the crontab -e command as shown in Figure 11-2 to edit a crontab file creates them in the /var/spool/cron directory. I strongly recommend that you not use a standard editor such as vi, vim, emacs, nano, or any of the many other editors that are available. Using the `crontab` command not only allows you to edit the command, it also restarts the crond daemon when you save and exit from the editor. The crontab command uses vi as its underlying editor because vi is always present on even the most basic of installations.

All cron files are empty the first time you edit it so you must create it from scratch. I always add the job definition example in Figure 11-2 to my own cron files just as a quick reference. This help and initial setup section is comprised of the top part of the crontab file down to the line of "#" characters. The rest of the file consists of the cron jobs I have set up.

---

**Tip** The crontab job definition help is located in the /etc/crontab file so you can copy that to your own crontab.

---

In Figure 11-2 the first three lines set up a default environment. Setting the environment to that necessary for a given user is required because cron does not provide an environment of any kind. The SHELL variable specifies the shell to use when commands are executed. In this case it specifies the bash shell. The MAILTO variable sets the email address to which cron job results will be sent. These emails can provide the status of backups, updates, or whatever, and consist of the output from the programs that you would see if you ran them manually from the command line. The last of these three lines sets up the PATH for this environment. Regardless of the path set here, however, I always like to prepend the fully qualified path to each executable file name.

```
# crontab -e

SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed
###############################################################################################
# backup using the rsbu program to the internal HDD then the external USB HDD
01 01 * * * /usr/local/bin/rsbu -vbd1 ; /usr/local/bin/rsbu -vbd2
# Set the hardware clock to keep it in sync with the more accurate system clock
03 05 * * * /sbin/hwclock --systohc
# Perform monthly updates on the first of the month
25 04 1 * * /usr/local/bin/doit
```

***Figure 11-2.*** *The crontab command is used to edit the cron files*

There are several comment lines that detail the syntax required to define a cron job. The entries in the crontab files are checked every minute by the crond daemon. Figure 11-3 defines each of the fields and the values allowed in each.

| Interval field | Values allowed | Comments |
|---|---|---|
| Minute | 0-59 | |
| Hour | 0-23 | |
| Day of month | 1-31 | |
| Month | 0-12 or month abbreviations | jan, feb, mar, etc, in lowercase. |
| Day of the week | 0-7 or day abbreviations | Zero (0) and 7 are both Sunday. sun, mon, tue, in lowercase. |

***Figure 11-3.*** *The crontab time fields*

## cron examples

Each interval field of the crontab entries also supports lists and ranges of values as well as calculated values. For example, suppose we want to run a task on the 1st and 15th of each month at 01:00 AM. That cron entry would look like this.

```
00 01 * 1,15 * /usr/local/bin/mycronjob.sh
```

We can get more creative using a little math. For example, if we want to run a task every 5 minutes, the minute field could be set as */5 to denote that. The way this works is that for each minute, the value is divided by 5. If the remainder of the division is zero, that is, the minute is evenly divisible by 5, that is considered a match. Of course the other time intervals must match too. These are called step values.

```
*/5 * * * * /usr/local/bin/mycronjob.sh
```

Suppose that we want to mycronjob.sh once every 3 hours on the hour during the day. This specification would run the task at 03:00 AM, 06:00 AM, 09:00 AM, and so on throughout the day.

```
00 */3 * * * /usr/local/bin/mycronjob.sh
```

In a more complex scheduling task, the next example shows one way to run a task on the first Sunday of each month. The logic here is that one and only one Sunday must always fall in the first 7 days of the month

```
00 01 1-7 * sun /usr/local/bin/mycronjob.sh
```

or

```
00 01 1-7 * 0 /usr/local/bin/mycronjob.sh
```

or because Sunday can be either 0 or 7

```
00 01 1-7 * 7 /usr/local/bin/mycronjob.sh
```

Now let's complicate this even more and imagine that we want this cron job to run only in the months of summer which we designate as June through September

```
00 01 1-7 jun,jul,aug,sept sun /usr/local/bin/mycronjob.sh
```

or

```
00 01 1-7 6,7,8,9 0 /usr/local/bin/mycronjob.sh
```

or

```
00 01 1-7 6-9 0 /usr/local/bin/mycronjob.sh
```

The crontab(5)[1] man page has a good description of how this all works. It also has some additional examples.

## crontab entries

There are three sample crontab entries in Figure 11-2 so we will explore those now that we know how to interpret them.

The following line shown runs one of my bash shell scripts, `rsbu`, to perform backups of all my systems. This job is kicked off at 1 minute after 01:00 AM every day. The splat/star/asterisks (*) in positions 3, 4, and 5 of the time specification are like file globs for those time divisions; they match every day of the month, every month, and every day of the week. This line runs my backups twice, once to backup onto an internal dedicated backup hard drive and once to backup onto an external USB hard drive that I can take to the safe deposit box.

```
01 01 * * * /usr/local/bin/rsbu -vbd1 ; /usr/local/bin/rsbu -vbd2
```

[1]Use the command form `man 5 crontab`.

This next cron entry sets the hardware clock on the computer using the system clock as the source of an accurate time. This line is set to run at 3 minutes after 05:00 AM every day.

```
03 05 * * * /sbin/hwclock --systohc
```

The last cron job is the one we are especially interested in. It is used to install Fedora updates at 04:25 AM on the first day of each month. The cron service has no option for "The last day of the month," so we use the first day of the following month.

```
25 04 1 * * /usr/local/bin/doit
```

Let's try a couple things to get a feel for using cron for task scheduling using the cron service.

## EXPERIMENT 11-3

Perform this experiment as root. Let's look at the crontab file for root using the crontab command. The -l option just prints the current crontab file while -e edits the cron file.

```
[root@studentvm1 ~]# crontab -l
no crontab for root
```

We can see from this result that there is no default crontab file for root. This is also true of all users because there are no crontab files at all. So let's create our own crontab file.

```
[root@studentvm1 ~]# crontab -e
```

This opens Vim with an empty file. We can start by importing the help file using the following vim command. The "r" stands for read, and the file name follows immediately with no space between the command and the file name. Vim must be in command mode for this, and it is so when it is first launched so we do not need to press Esc to switch to command mode.

```
:r/etc/crontab
```

So now we have the beginnings of a crontab file with built-in help. Now we add a simple command to illustrate how cron works with repetitive tasks. I like to use a trivial example for this and repeat it every minute. Add the following two lines to the bottom of the crontab file. This cron job runs the free utility once each minute and stores it in the /tmp/freemem.log file.

```
# Run the free program and store the results in /tmp/freemem.log
*  *  *  *  * /usr/bin/free >> /tmp/freemem.log
```

Use the command **Esc :wq** to write the data to the disk and exit from Vim to activate the changes. You should receive the message shown next.

```
"/tmp/crontab.YOLBoe" 19L, 566C written
crontab: installing new crontab
```

Open a root terminal session and make /tmp the PWD. Use **ls** to check that the file is present, but it will not be until a second or so after the first minute changes to the next. So if you saved the crontab file at 13:54:32, the first entry will be made at about 13:55.01. You can use the stat command to get the exact time precisely.

We can use the **tail -f** command to follow the file. That is, using this command shows the current content of the file, and whenever new lines are added to the file, they are displayed immediately. This makes it unnecessary to use commands like **cat** every minute or so to see the file as it changes.

```
[root@studentvm1 tmp]# tail -f freemem.log
              total        used        free      shared  buff/cache   available
Mem:        4036976      222640     3101688        3012      712648     3577336
Swap:       6291452           0     6291452
              total        used        free      shared  buff/cache   available
Mem:        4036976      222276     3101948        3016      712752     3577696
Swap:       6291452           0     6291452
              total        used        free      shared  buff/cache   available
Mem:        4036976      222096     3101956        3012      712924     3577828
Swap:       6291452           0     6291452
<snip>
```

This does not tell us the date or time that the entries were made. We can add another statement to our existing cron job, as shown in the following, to make that happen.

```
# Run the free program and store the results in /tmp/freemem.log
*  *  *  *  * /usr/bin/date >> /tmp/freemem.log ; /usr/bin/free >> /tmp/
freemem.log
```

Save the revised crontab and tail the freemem.log file for a few minutes to observe the revised results.

```
[root@studentvm1 tmp]# tail -f freemem.log
              total        used        free      shared  buff/cache   available
Mem:        4036976      223200     3099652        3012      714124     3576724
Swap:       6291452           0     6291452
              total        used        free      shared  buff/cache   available
Mem:        4036976      222944     3099904        3012      714128     3576996
Swap:       6291452           0     6291452
Thu Apr 18 15:14:01 EDT 2019
              total        used        free      shared  buff/cache   available
Mem:        4036976      223600     3094012        3012      719364     3576220
Swap:       6291452           0     6291452
```

# Other scheduling options

There are some other options provided by the cron service that we can also use to schedule programs to run on a regular basis.

## /etc/cron.d

The directory /etc/cron.d is where some applications install cron files when there are no users under which the programs would run, these programs need a place to locate cron files so they are placed in /etc/cron.d. These cron files have the same format as a user cron file. The crontab files located in this directory are each. The root user can also place crontab files in this directory.

The /etc/cron.d directory should contain three files. We are particularly interested in the 0hourly crontab file.

**EXPERIMENT 11-4**

Perform this experiment as root. Make /etc/cron.d the PWD. Then list the contents of the directory and view the contents of 0hourly.

```
[root@studentvm1 ~]# cd /etc/cron.d ; ll ; cat 0hourly
total 12
-rw-r--r--. 1 root root 128 Mar 18 06:56 0hourly
-rw-r--r--. 1 root root  78 Feb  9  2018 atop
-rw-r--r--. 1 root root 108 Sep 13  2018 raid-check
```

```
# Run the hourly jobs
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
01 * * * * root run-parts /etc/cron.hourly
[root@studentvm1 cron.d]#
```

The run-parts command in the 0hourly crontab file runs all of the files in the /etc/cron.hourly directory in alphanumeric sorted sequence beginning at 1 minute after each hour. We explore the reason for this in the next section.

## anacron

The crond service assumes that the host computer runs all the time. What that means is that if the computer is turned off for a period of time and cron jobs were scheduled for that time, they will be ignored and will not run until the next time they are scheduled. This might cause problems if the cron jobs that did not run were critical. So there is another option for running jobs at regular intervals when the computer is not expected to be on all the time.

The anacron program performs the same function as crond, but it adds the ability to run jobs that were skipped if the computer was off or otherwise unable to run the job for one or more cycles. This is very useful for laptops and other computers that get turned off or put in sleep mode.

As soon as the computer is turned on and booted, anacron checks to see whether configured jobs have missed their last scheduled run. If they have, those jobs are run almost immediately, but only once no matter how many cycles have been missed. For example, if a weekly job was not run for 3 weeks because the system was shut down while you were away on vacation, it would be run soon after you turn the computer on, but it would be run once, not three times.

The anacron program provides some easy options for running regularly scheduled tasks. Just install your scripts in the /etc/cron.[hourly|daily|weekly|monthly] directories, depending on how frequently they need to be run.

How does this work? The sequence is simpler than it first appears. The crond service runs the cron job specified in /etc/cron.d/0hourly as seen in Figure 11-4.

```
# Run the hourly jobs
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
01 * * * * root run-parts /etc/cron.hourly
```

***Figure 11-4.*** *The contents of /etc/cron.d/0hourly cause the shell scripts located in /etc/cron.hourly to run*

The cron job specified in /etc/cron.d/0hourly runs the run-parts program once per hour. The run-parts program runs all of the scripts located in the /etc/cron.hourly directory. The /etc/cron.hourly directory contains the 0anacron script which runs the anacron program using the /etdc/anacrontab configuration file shown in Figure 11-5.

```
# /etc/anacrontab: configuration file for anacron
# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
# the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=45
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22

#period in days   delay in minutes   job-identifier   command
1       5       cron.daily              nice run-parts /etc/cron.daily
7       25      cron.weekly             nice run-parts /etc/cron.weekly
@monthly 45     cron.monthly            nice run-parts /etc/cron.monthly
```

***Figure 11-5.*** *The contents of /etc/anacrontab file run the executable files in the cron.[daily|weekly|monthly] directories at the appropriate times*

The anacron program runs the programs located in /etc/cron.daily once per day; it runs the jobs located in /etc/cron.weekly once per week and the jobs in cron.monthly once per month. Note the specified delay times in each line that helps prevent these jobs from overlapping themselves and other cron jobs.

Files that are located in the /etc/cron.X directories are not executable from the command line unless used with the full path. So instead of placing complete bash programs in the cron.X directories, I install them in the /usr/local/bin directory which allows me to run them easily from the command line. Then I add a symlink in the appropriate cron directory, such as /etc/cron.daily.

The anacron program is not designed to run programs at specific times. Rather, it is intended to run programs at intervals that begin at the specified times such as 03:00 AM (see the START_HOURS_RANGE in Figure 11-5) of each day, on Sunday to begin the week, and the first day of the month. If any one or more cycles are missed, then anacron will run the missed jobs one time as soon as possible.

# Thoughts about cron

I use most of these methods for scheduling various tasks to run on my computers. All of those tasks are ones that need to run with root privileges. I have seen only a few times when non-root users had a real need for any type of cron job, one of those being for a developer to kick off a daily compile in a development lab.

It is important to restrict access to cron functions by non-root users. However, there are circumstances when it may be necessary for a user to set tasks to run at specified times and cron can allow users to do that when necessary. SysAdmins realize that many users do not understand how to properly configure these tasks using cron and the users make mistakes in the configuration. Some of those mistakes may be harmless, but others can cause problems for themselves and other users. By setting procedural policies that cause users to interact with the SysAdmin, those individual cron jobs are much less likely to interfere with other users and other system functions.

## Scheduling tips

Some of the times I have set in the crontab files for my various systems seem rather random and to some extent they are. Trying to schedule cron jobs can be challenging especially as the number of jobs increases. I usually only have a few tasks to schedule on each of my own computers so it is a bit easier than some of the production and lab environments I have worked.

One system for which I was the SysAdmin usually had around a dozen cron jobs that needed to run every night and an additional three or four that had to run on weekends or the first of the month. That was a challenge because if too many jobs ran at the same time, especially the backups and compiles, the system would run out of RAM and then nearly fill the swap file which resulted in system thrashing while performance tanked so that nothing got done. We added more memory and were able to do a better job of scheduling tasks. Adjusting the task list included removing one of the tasks which was very poorly written and which used large amounts of memory.

## Security

Security is always important and that is no different for cron jobs. I recommend that non-root users be prohibited from creating cron jobs. A badly designed script or program that is launched unintentionally multiple times by a cron job can bring a host to a very quick and unexpected stop.

It is considered a best practice to deny use of the cron system to non-root users in order to help eliminate rogue cron jobs. This can be accomplished using the files cron.allow and cron.deny. The logic of these two files is summarized in Figure 11-6. The root user can always use cron.

| cron.allow | cron.deny | Effect |
|---|---|---|
| Not present | Not present | Only root can use cron. |
| Present but empty | Not present | Only root can use cron. |
| Present | Not present | User ID must be listed in cron.allow to use cron. |
| Not present | Present but empty | All non-root users can use cron. |
| Not present | Present | Users listed in cron.deny are denied access to cron. |

***Figure 11-6.*** *Using cron.allow and cron.deny to control access to using cron*

All cron jobs should be thoroughly inspected and tested before being added by the root user.

## cron resources

The man pages for cron, crontab, anacron, anacrontab, and run-parts all have excellent information and descriptions of how the cron system works.

# at

The tasks we have discussed so far are repetitive and need to occur on a repeating schedule of some type. There are times, however, when the need is to run a job only once at some time in the future. The **at** command can be used for this.

For example, I have had the need to install updates during a 02:00 AM maintenance window for which I did not want to be awake or even on site. Actually, to be a bit more specific, the updates could be performed during the day, but the reboot required by a new kernel would need to be performed during the maintenance period. This is possible because a Linux system can be updated without an immediate reboot, and that is the default. Additionally, performing the update does not normally affect the other tasks so users would not even know that the updates are being installed. The only time a side effect such as slowed response time might be noted is if the host is already running at nearly 100% of CPU capacity.

In my case it was totally unnoticeable so I did the updates during the day. I then set an **at** job to run during the maintenance window and perform the reboot.

## Syntax

The syntax of the **at** command is simple.

Type **at <time specification>** and press the Enter key. We look at time specifications immediately in the following data. This starts a new line indicated by the prompt, **at>**. Type in a series of commands to be executed at the specified time, and then press the Ctrl-D key combination to exit and activate the job. As far as I know, this is the only Linux command that uses Ctrl-D to perform a save and exit sequence.

The **atq** command lists the jobs in the queue, and the **atrm <job number>** command allows removal of jobs from the queue.

## Time specifications

The **at** command provides some interesting ways to specify times and dates, both explicit and fuzzy. Some examples of these methods are listed in Figure 11-7. Note that if a time of day (TOD) is specified, but not a specific day, the job will run today if that time is still in today's future. If that time for today has already passed, the job will run the next instance of that time, which would be tomorrow.

We can generalize that any **at** job time specification will execute the first time the specification matches a future time. If the specification matches a time that exists only in the past, it will run as soon as possible, usually within a few minutes.

| Time specification | Description |
|---|---|
| at 05:00 | A specific time with no day or date supplied. At 5:00am. Today if the current time is before 5am and tomorrow if the current time is after 5am. 5am and 5:00am also work. |
| at 5pm | At 5pm today if the current time is before 5pm and tomorrow if the current time is after 5pm. |
| at 11am tuesday | A time and day of the week is specified. If the current day is Tuesday but the time is after 11am, the job will run on 11am Tuesday in 7 days from now. If today is Monday, the job will run at 11am tomorrow. |
| at 3pm + 5 days | This job will run at 3pm five days from now regardless of which day today is. |
| at now + 10 minutes | If the time this job is added to the queue is 11:27am, it will run at 11:37am. |
| at tomorrow | At this time tomorrow. If the job is entered into the queue at 09:48 today, it will run at 09:48 tomorrow. |
| at 21:05 January 15 | This job will run on a specific month and day at 9pm. |
| at noon | This sets a job to run at the next 12pm (noon), today if the current time is before noon and tomorrow if it is already after noon. |
| at midnight | This job will run at the next occurrence of midnight, 12:00AM. |
| At teatime | A job with this specification will run at 4pm. |
| at 15:35 05/21/2019 | This job will be run at the date and time specified of 15:35 (3:35pm) on May 21 of 2019. Acceptable date specification formats are MMDD[CC]YY, MM/DD/[CC]YY, DD.MM.[CC]YY, and [CC]YY-MM-DD. |

***Figure 11-7.*** *Examples of time/date specifications for the* ***at*** *command.*

By now you get the idea and should be able to specify time and date for the **at** command in any number of different ways. I like the flexibility this command provides in the time and date specifications. This is an excellent example of the Linux Philosophy tenet that programs should do one thing and do it well.

## EXPERIMENT 11-5

Let's start with something simple and see how it works. Start this experiment as the student user. EOT means End of Text, which is issued with the Ctrl-D key combination.

```
[student@studentvm1 ~]$ at now +2 minutes
warning: commands will be executed using /bin/sh
at> free
at> <EOT>
job 1 at Thu May  2 15:06:00 2019
```

Note that the job number is displayed and the date and time the job will be run. We can also use the **atq** command to see this. The **atq** command also shows the username that the job belongs to. There is only a single queue for **at** jobs and all jobs go into that queue. The **atq** command shows all at jobs for all users.

```
[student@studentvm1 ~]$ atq
1       Thu May  2 15:06:00 2019 a student
[student@studentvm1 ~]$
```

Now wait until after the runtime shown in the **atq** results. What happens? Why do we not see anything? Does the job even run? Use the **atq** command to verify that the queue is now empty.

We can verify the job ran by looking in the cron log. As root, do the following.

```
[root@studentvm1 ~]# tail /var/log/cron
May  2 14:01:01 studentvm1 CROND[22490]: (root) CMD (run-parts /etc/cron.hourly)
May  2 14:01:01 studentvm1 run-parts[22490]: (/etc/cron.hourly) starting 0anacron
May  2 14:01:01 studentvm1 run-parts[22490]: (/etc/cron.hourly) finished 0anacron
May  2 15:01:01 studentvm1 CROND[23351]: (root) CMD (run-parts /etc/cron.hourly)
May  2 15:01:01 studentvm1 run-parts[23351]: (/etc/cron.hourly) starting 0anacron
May  2 15:01:01 studentvm1 run-parts[23351]: (/etc/cron.hourly) finished 0anacron
May  2 15:06:00 studentvm1 atd[26634]: Starting job 3 (a00003018bebba) for user
                                       'student' (1000)
```

```
May  2 15:08:24 studentvm1 atd[26691]: Starting job 4 (a00004018bebbc) for user
                                       'student' (1000)
May  2 15:10:00 studentvm1 atd[26763]: Starting job 5 (a00005018bebbe) for user
                                       'student' (1000)
May  2 15:15:00 studentvm1 atd[26883]: Starting job 6 (a00006018bebc3) for user
                                       'student' (1000)
[root@studentvm1 log]#
```

You should see at least one entry indicating that the at job has started. This verifies the job ran, but what happened to the output? The answer is nowhere. We do not have all of the parts in place to receive the emails that it would normally send to the user as its default way of communicating the results. We need to install sendmail, which is a mail handling and transfer agent, and mailx, which is a text-mode email client that can be used from the command line.

As root, install these two tools. This may also install a few dependencies on your VM.

```
[root@studentvm1 log]# dnf -y install mailx sendmail
```

Activate SendMail. We use the systemctl command here to manage the SendMail server. The start sub-command obviously starts the server, while the enable sub-command configures it to start at every system boot. The **systemctl** command is used to manage system services and background processes (daemons) and is part of systemd. We will explore systemd and the systemctl command in Chapter 13.

```
[root@studentvm1 log]# systemctl status sendmail
● sendmail.service - Sendmail Mail Transport Agent
   Loaded: loaded (/usr/lib/systemd/system/sendmail.service; disabled; vendor
preset: disabled)
   Active: inactive (dead)
[root@studentvm1 log]# systemctl start sendmail ; systemctl enable sendmail
Created symlink /etc/systemd/system/multi-user.target.wants/sendmail.service
→ /usr/lib/systemd/system/sendmail.service.
Created symlink /etc/systemd/system/multi-user.target.wants/sm-client.service
→ /usr/lib/systemd/system/sm-client.service.
[root@studentvm1 log]#
```

Now let's do the same at job as before.

```
[student@studentvm1 ~]$ at now + 2 minutes
warning: commands will be executed using /bin/sh
at> free
```

```
at> <EOT>
job 7 at Thu May  2 16:23:00 2019
[student@studentvm1 ~]$ atq
7       Thu May  2 16:23:00 2019 a student
```

After 2 minutes have passed, use **atq** to verify the job queue is now empty. Then type mailx to view the email in your mailq.

```
[student@studentvm1 ~]$ mailx
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/student": 1 message 1 new
>N  1 Student User          Thu May  2 16:23  19/937   "Output from your
job        7"
&
```

The ampersand (&) is the command prompt for mailx. We can see that there is one message in the inbox. It has the number 1, that we can use to manage it. Type **1** at the prompt and press **Enter**. The message should look very similar to the following one. It shows the standard email message headers, a subject, and the output from our job.

```
& 1
Message  1:
From student@studentvm1.both.org  Thu May  2 16:23:01 2019
Return-Path: <student@studentvm1.both.org>
Date: Thu, 2 May 2019 16:23:00 -0400
From: Student User <student@studentvm1.both.org>
Subject: Output from your job        7
To: student@studentvm1.both.org
Status: R

              total        used        free      shared  buff/cache   available
Mem:        4036976      252396     2866344        3116      918236     3537948
Swap:       6291452           0     6291452

&
```

Type **q** and **Enter** to exit from mailx.

```
& q
Held 1 message in /var/spool/mail/student
You have mail in /var/spool/mail/student
[student@studentvm1 ~]$
```

This is a trivial example, but it does show you how the **at** command works and how to deal with the data stream from the jobs.

There are two more fun things that we can do with the **at** command. The **wall** command writes a message to all users. We can use that to send a message to all users at a specified time. Be sure you have more than one terminal session open for the student user.

```
[student@studentvm1 ~]$ at now + 2 minutes
warning: commands will be executed using /bin/sh
at> wall "Hello World."
at> <EOT>
job 8 at Thu May  2 21:42:00 2019
[student@studentvm1 ~]$
```

And the results of the preceding code at job look like this. This message will appear in every terminal session of the student user. If this is done as root, it will appear in all terminal sessions for all users.

```
[student@studentvm1 ~]$

Broadcast message from student@studentvm1 (somewhere) (Thu May  2
21:42:00 2019

Hello World.

Broadcast message from student@studentvm1 (somewhere) (Thu May  2 21:42:00 2019

Hello World.

You have new mail in /var/spool/mail/student
[student@studentvm1 ~]$
```

Notice that we also get a message indicating that the student user has received the email notification.

We can also send a message to a specific terminal. First let's determine which pseudo-terminal we are using as the student user.

```
[student@studentvm1 ~]$ who am i
student  pts/4        2019-05-02 13:08 (:pts/2:S.1)
[student@studentvm1 ~]$
```

Now let's create an at job that sends a message only to that terminal. You might normally set something like this to ensure you leave work on time at the end of the day.

```
[student@studentvm1 ~]$ at now + 5 minutes
warning: commands will be executed using /bin/sh
at> echo "It is time to go home." > /dev/pts/4
at> <EOT>
job 9 at Thu May  2 21:53:00 2019
[student@studentvm1 ~]$
```

The result looks like this, but it only appears on the specified terminal session, not all of them.

```
[student@studentvm1 ~]$ It is time to go home.

[student@studentvm1 ~]$
```

Sending messages like this to all or to a single terminal session can also be done using cron for repeating tasks.

## Security

The **at** command uses the files at.allow and at.deny to specify which users have access to the at command. The logic is the same as with cron.allow and cron.deny as discussed earlier.

## Cleanup

If you have any cron jobs still active, delete them.

## Chapter summary

This chapter has explored methods for running tasks at specific times using various repetitive time periods or only a single future time. The ability to run tasks at specified times can make the SysAdmin's job easier by removing the need to be present – or even awake – when the tasks need to be run.

Note that systemd has its own tools called timers that are designed to be used in the same manner as cron and at. Because these timers are so closely integrated with and managed by systemd, we will cover them in Chapter 13 of this volume.

# Exercises

Perform the following exercises to complete this chapter:

1. Create a crontab entry that runs /usr/local/bin/mycronjob.sh at 09:00 AM and 05:00 PM on the 7th and 21st of each month.
2. Where are cron files – those created by the crontab command – stored?
3. Describe the difference between cron and anacron.
4. Why might some cron files be stored in /etc/cron.d?
5. When is the first time after a host is booted that the files managed by anacron are executed?
6. Create a script that generates a listing of the filesystems, their sizes, and how much space is used and/or available. The result should be appended to a file in /etc. This does not need to be fancy, just a single command can do this. Place the script in /usr/local/bin. Use at least three different cron methods to run this job once per hour. After testing that, use three different ways to run it once per day.
7. When entering **at** jobs and using a specification like now + 5 minutes, assume the time is 09:04 AM when the command **at now + 5 minutes** is issued and the time is 09:06 AM when **Ctrl-D** is entered to add the job to the queue. At what time does the **at** job execute?

CHAPTER 12

# Networking

## Objectives

In this chapter you will learn

- Some basic networking concepts
- To define and describe the TCP/IP five-layer network model
- The structure of IPv4 and IPv6 addresses
- To define and use Classless Inter-Domain Routing (CIDR) network notation
- How to use simple tools to explore network ranges and subnetting
- Basic client-side DHCP network configuration
- How to use Linux CLI tools to explore and understand the host network configuration
- The basics of routing
- To manage firewalls using IPtables and ufw

## Introduction

Today, in 2019, nearly every electronic device on the planet is connected to the Internet. Devices such as computers, smart phones, and tablets are more or less obvious. But it goes even further because we have thermostats, refrigerators, and security systems, and even our cars are connected.

Linux-based computers are no exception. At the beginning of this course, you downloaded VirtualBox and the Xfce version of Fedora and you had to be connected to the Internet to do so. You also configured a virtual network for the virtual machine




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_12

you created and tested it to ensure that your VM had connectivity to the outside world through that virtual network.

In this chapter we discuss client-side networking and explore how our VM is configured automatically at boot time. We will look at the functional aspects of routing and firewalls on the client side. We will discuss servers and services such as DHCP, DNS, and routing in more detail from the server side in Volume 3 of this course.

## About IPv6

We will concentrate on IPv4 (IP version 4) in this course because it is still far more common than IPv6 (IP version 6) at the end user side. Many ISPs use IPv6 for their backbone networks and are slowly progressing with IPv6 to the end user. Very slowly. We will discuss some of the IPv6 concepts but will base our experiments on IPv4.

# Basic networking concepts

Although this course is intended to provide a practical approach to Linux and networking, some important concepts are a necessary foundation on which to build further understanding.

## Definitions

Let's start with some important definitions. We will encounter other terms and their definitions as we proceed through this chapter:

- A **node** is any device connected to and accessible on a network including computers, routers, printers, and any other network-attached device.
- A **host** is a node that is specifically a computer attached to the network.
- **IP** stands for Internet Protocol. It is a group of network layer protocols that allow computers to communicate with each other. IP is a best-effort packet-switching protocol used to transmit data packets from one network node to another. It is not considered reliable because data packets can be lost or intentionally dropped for any of a number of reasons.

- **TCP** is Transmission Control Protocol that sits on top of IP. TCP provides reliable communications and flow control including full duplex which means that communication can go in both directions over the transmission medium at the same time.
- A **network** (of any kind) is a web- or net-like structure of communications systems that allow connected nodes to communicate with each other. There is a good physical model for this. It is inexact but a good starting point for further understanding.

This model is the system of roadways for automobiles and trucks (vehicles) to carry passengers and freight from one location to another. The main end points are the cities and specific homes or businesses in the cities. Each vehicle is analogous to a data packet in a computer network, and the passengers and freight are like the data contained in a packet.

These vehicles start from a location such as a house and travel the roads through surface intersections and large highway interchanges, switching from one route to another as required to get to the final destination. The intersections and interchanges along the way are similar to the routers used to switch data packets from one path on the Internet to another. However, in a computer network, the router makes the decisions, and for these vehicles, the driver makes the decisions about where to turn onto another route.

In this model each packet or vehicle is transported independently of surrounding packets from source to destination:

- A **NIC** is a network interface controller[1] that is either built in to a computer motherboard or which can be added as a pluggable device card. A NIC provides the computer with a hardware connection to a network.
- A network **node** is any device connected to a network and which is addressable by other devices such that a connection can be made.

[1]Wikipedia, *Network interface controller*, http://en.wikipedia.org/wiki/Network_Interface_Controller

- A **switch** is a hardware device that is used to connect multiple nodes together on a logical network segment. Two or more nodes or hosts have Ethernet cable connecting them to the switch so that they can communicate with each other. It is possible to connect two computers together using a special crossover cable, but that is unusual and limiting because only two computers can be connected that way. Switches are not visible to the TCP/IP protocols and operate only on the physical layer.
- A **router** is a device that routes data packets between two or more networks based on the destination IP address contained in the data packets. Routers have IP addresses for each network to which they connect and are visible to other devices on those networks.
- At least one router on the network is the **default gateway** to other networks or the rest of the Internet. If a data packet is sent by a host and there are no other routes defined, the default gateway sends the packet to the next router on the way to its final destination.
- A **connection** is a logical link between two nodes on a network. Connections exist at each layer of the TCP/IP stack.
- The term **stack** refers to the stacked layers of the TCP/IP network model. These layers form a stack of hardware and the software protocols that create links between network nodes. We explore the TCP/IP network model in some detail a bit further on in this chapter.

## MAC address

A **MAC address**[2] is a unique hardware address assigned to each network interface card (NIC) that provides the hardware a means of identification. The MAC address is configured permanently in the hardware by the device vendor and cannot be changed. This is called a universally administered address (UAA) and is sometimes referred to as the "burned-in address" or the "hardware address."

[2]Wikipedia, *MAC address*, https://en.wikipedia.org/wiki/MAC_address

There are, however, software methods that can be used to assign different MAC addresses to an interface, but the reasons for doing so are beyond the scope of this book and I strongly recommend against doing so in any event for wired devices.[3] This is a locally administered address (LAA).

Xerox Network Systems created the original 48-bit Ethernet addressing scheme. The 48-bit address space contains 281,474,976,710,656 (2^48) possible MAC addresses.

MAC addresses consist of six two-digit Hexadecimal (Hex) numbers separated by colons, such as 08:00:27:81:ec:cc, which is the MAC address of my VM. The first three pairs are the Organizational Unique Identifier (OUI) and can be used to identify the vendor of the NIC.[4] The last three pairs are the hardware ID for that specific NIC. The OUI numbers are assigned to vendors by the Institute of Electrical and Electronic Engineers[5] (IEEE).

## EXPERIMENT 12-1

This experiment should be performed as the student user; root access is not required.

Identify the installed network interface cards and their MAC addresses. This also shows the IP addresses, but we will ignore that for now.

```
[student@studentvm1 ~]$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
   default qlen 1000
    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 fq_codel state UP
   group default qlen 1000
    link/ether 08:00:27:e1:0c:10 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 1057sec preferred_lft 1057sec
```

[3]It is probably a good security practice to alter the MAC address of wireless devices if you want to prevent your device from being tracked.

[4]AJ Arul's Utilities, https://aruljohn.com/mac.pl

[5]IEEE, https://www.ieee.org/

```
    inet6 fe80::b7f2:97cf:36d2:b13e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

The first entry is the local (lo) interface which is used by many Linux kernel tasks and applications to communicate within the host. Every Linux computer has a local interface even if it is not connected to a network or even if it does not have a NIC installed. This interface is an absolute requirement for any Linux (or Unix) computer to function properly.

The second entry, enp0s3 (0 = zero), is how Linux sees the first virtual network adapter on a VirtualBox VM. The second NIC, if configured in the VirtualBox manager for this VM, would be enp0s8. The link/ether line in this second entry shows 08:00:27:e1:0c:10 as the MAC address.

All networked devices have a MAC address, and the ip command enables us to see the MAC and IP addresses of the "neighbor" hosts with which our host has communicated.

```
[student@studentvm1 ~]$ ip neigh
10.0.2.1 dev enp0s3 lladdr 52:54:00:12:35:00 REACHABLE
```

You should see one line as the result of this command which should contain the IP address 10.0.2.1. If you do not see this line, ensure that the StudentVM1 host has recently communicated with the router/gateway. Then retry the **ip neigh** command.

```
[root@studentvm1 ~]# ping -c2 _gateway ; ip neigh
PING _gateway (10.0.2.1) 56(84) bytes of data.
64 bytes from _gateway (10.0.2.1): icmp_seq=1 ttl=255 time=0.152 ms
64 bytes from _gateway (10.0.2.1): icmp_seq=2 ttl=255 time=0.219 ms

--- _gateway ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 56ms
rtt min/avg/max/mdev = 0.152/0.185/0.219/0.036 ms
```

This is the virtual router in our virtual network. REACHABLE means that the NIC on that device is reachable and has been accessed recently. Another thing you might see as the last field in the line is STALE which means the connection has not been used for some time. The ip man page does not specify what that time might be. You might also encounter DELAY as this last field. That means that there was a delay, probably small, the last time that connection was used.

We can also use the **arp** command to view the MAC addresses that are known to our host. Using **arp** without options displays the DNS names of any nodes that have an entry in DNS. Using the -n option displays the IP addresses.

```
[student@studentvm1 ~]$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
_gateway                 ether   52:54:00:12:35:00   C                     enp0s3
[student@studentvm1 ~]$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
```

MAC addresses are limited in scope to the physical network segment on which they reside. They are not routable and cannot be accessed outside the local network segment.

The **ip** command is designed to replace the **ifconfig**, **arp**, and some other network-related commands, so Red Hat has published a very nice IP command cheat sheet[6] that I use frequently. The man page for the **arp** command contains the following note, "This program is obsolete. For replacement check **ip neigh**" and man page for the **ifconfig** command has a similar one.

For now, those commands are still available, and it may be years before they disappear completely.

## IP address

The IPv4 address[7] is composed of four sets of Hexadecimal pairs, called octets because they each contain eight (8) binary bits, for a total of 32 bits. The octets are separated by periods, such as 192.168.25.36. Each octet can have a maximum value of 2^8-1, or 255. Computers and network routing and management equipment deal with the binary forms of IP addresses, but our devices are smart enough to display them to us in a human-readable form.

Any computer or other device that needs to be accessible on a network or the Internet must have an IP address assigned to it. IP addresses are routable and can be accessed from other network segments through a router. Some IP address ranges are reserved for internal, private use by organizations. These private IP address ranges are well defined, and we will explore this and more about IP addresses in some detail later in this chapter.

[6]Red Hat, *IP Command Cheat Sheet*, https://access.redhat.com/sites/default/files/attachments/rh_ip_command_cheatsheet_1214_jcs_print.pdf

[7]Wikipedia, *IP Address*, https://en.wikipedia.org/wiki/IP_address

Because IPv4 was running out of assignable addresses and as a way to implement more efficient routing through the Internet, IPv6 was developed. IPv6 uses 128 bits for addressing, divided into eight sections of four Hexadecimal digits. A typical IPv6 address looks like this, `2001:0db8:0000:0000:0000:ff00:0042:8329`, which looks rather daunting, but which can be shortened by omitting leading zeros and eliminating consecutive sections of zeros. The result looks like this, `2001:db8::ff00:42:8329`.

## EXPERIMENT 12-2

Perform this experiment as the student user. Let's use the ip command again, but this time look at the IP addresses it reveals.

```
[root@studentvm1 ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    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 fq_codel state UP
group default qlen 1000
    link/ether 08:00:27:e1:0c:10 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 1099sec preferred_lft 1099sec
    inet6 fe80::b7f2:97cf:36d2:b13e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

The loopback IPv4 address is 127.0.0.1 and, as previously mentioned, this network interface and its IP address is used by many Linux kernel processes and other applications. The IPv6 address is ::1/128. We can ping both addresses. The ping command sends a special ICMP[8] data packet to the target IP address that simply says, "Hi – please respond," and is a way to determine whether another host is on the network and active.

```
[root@studentvm1 ~]# ping -c2 ::1
PING ::1(::1) 56 data bytes
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from ::1: icmp_seq=2 ttl=64 time=0.111 ms

--- ::1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 73ms
rtt min/avg/max/mdev = 0.067/0.089/0.111/0.022 ms
[root@studentvm1 ~]# ping -c2 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.069 ms

--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 103ms
rtt min/avg/max/mdev = 0.063/0.066/0.069/0.003 ms
```

We can also ping remote hosts. We use the example.com domain which is a legitimate domain set up specifically for testing. DNS (Domain Name System) converts the human-readable domain name into an IP address which is then used as the destination address in the **ping** command.

```
[root@studentvm1 ~]# ping -c2 www.example.com
PING www.example.com (93.184.216.34) 56(84) bytes of data.
64 bytes from 93.184.216.34 (93.184.216.34): icmp_seq=1 ttl=54 time=37.10 ms
64 bytes from 93.184.216.34 (93.184.216.34): icmp_seq=2 ttl=54 time=151 ms

--- www.example.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 148ms
rtt min/avg/max/mdev = 37.968/94.268/150.568/56.300 ms
```

[8]Wikipedia, *Internet Control Message Protocol*, https://en.wikipedia.org/wiki/internet_Control_Message_Protocol

## IP address assignments

The Internet Assigned Numbers Authority (IANA)[9] is responsible for global coordination and management of IP address and autonomous system (AS) number assignments. This organization coordinates the assignments of IP addresses to large geographic-political entities. Registries within those divisions are responsible for assigning addresses to customers such as ISPs. The IANA web site has a great deal of information that you may find useful.

# TCP/IP

Before we get into networking any further, it helps understand a little bit about how packets find their way to the correct host on a network. The TCP/IP network model defines a five-layer stack that describes the mechanisms necessary to move data packets from one host to another, whether that host is on the local network or someplace else on the planet.

---

**Tip** Some versions of this model use a four-layer stack in which the bottom two layers, datalink and physical, are combined into a single layer. I prefer the five-layer version because I think it provides more clarity.

---

## The TCP/IP network model

Each of the five layers in the following description of this model is numbered and also contains the names of the data units that are handled by that layer. The diagram in Figure 12-1 shows each layer and the protocols typically encountered at that layer. This list describes the layers of the stack in vertical order from top to bottom:

1. **Application layer – Message:** This layer consists of the connection protocols required for various network applications to communicate, such as HTTP, DHCP, SSH, FTP, SMTP, IMAP, and others. When you request a web page from a remote web site, a

[9]Internet Assigned Numbers Authority (IANA), http://www.iana.org/

connection request is sent to the web server and the response is sent back to your host at this layer and then your browser displays the web page in its window.

2. **Transport layer – TCP segment:** The transport layer provides end-to-end data transport and flow management services that are independent of the data and types of protocols being transported. It uses ports such as 80 for HTTP and 25 for SMTP to make connections between the sending host and the remote host.

3. **Internet layer – Packet:** Packet routing is performed on the Internet layer. This layer is responsible for routing packets across two or more different networks in order to reach their final destination. This layer uses IP addresses and the routing table to determine which device to send the packets to next. If sent to a router, each router is responsible for sending the data packets only to the next router in the series and not for mapping out the entire route from the localhost to the target host. The Internet layer is mostly about routers talking to routers in order to determine the next router in the chain.

4. **Data link layer – Frame:** The link layer manages the direct connections between hardware hosts on a single, local, logical, physical network. This layer uses the media access control (MAC) addresses embedded in the network interface cards (NICs) to identify the physical devices attached to the local network. This layer cannot access hosts that are not on the local network.

5. **Physical layer – Bits:** This is the hardware layer and consists of the NICs and the physical network (usually Ethernet) cable as well as the hardware-level protocols used to transmit individual bits that make up the data between any two hosts or other network nodes that are locally connected.

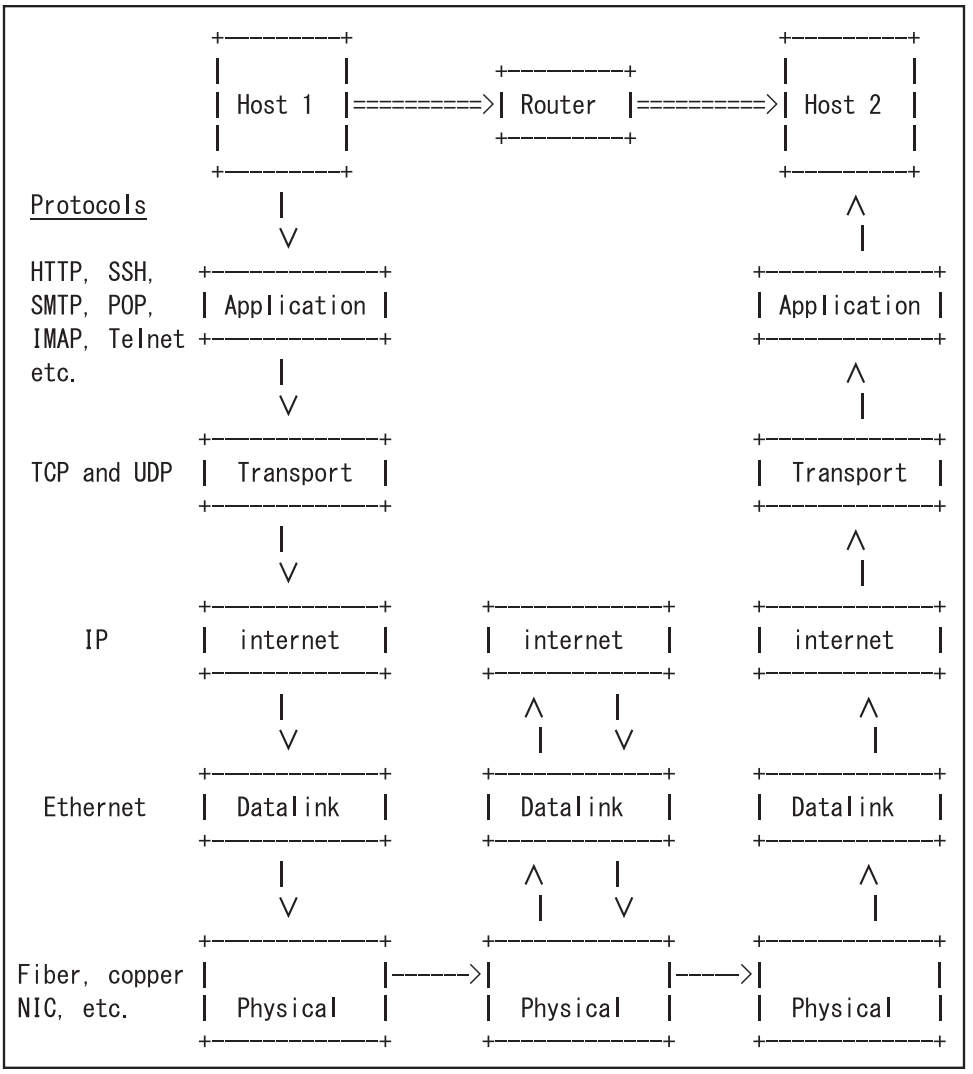


***Figure 12-1.*** *The TCP/IP network model*

## A simple example

So what does that look like when a host is actually sending data on the network using the TCP/IP network model? Here is my own made-up description of how data are moved from one network to another. In this example my computer is sending a request to a remote server for a web page.

Figure 12-1 can be used to follow the flow of data through the various layers of the TCP/IP model as you proceed through this example:

1. On the application layer, the browser, for example, initiates an HTTP or HTTPS connection request message to the remote host, www.example.com, to send back the data comprising the contents of a web page. This is the message, and it includes only the IP address of the remote web server.
2. The transport layer encapsulates the message containing the web page request in a TCP datagram with the IP address of the remote web server as the destination. Along with the original request packet, this packet now includes the source port from which the request will originate, usually a very high number random port, so that the return data knows which port the browser is listening on, and the destination port on the remote host, port 80 in this case.
3. The Internet layer encapsulates the TCP datagram in a packet that also contains both the source and destination IP addresses.
4. The data link layer uses the Address Resolution Protocol (ARP) to identify the physical MAC address of the default router and encapsulates the Internet packet in a frame that includes both the source and destination MAC addresses.
5. The frame is send over the wire – usually CAT5[10] or CAT6[11] – from the NIC on the localhost to the NIC on the default router. In a wireless environment, the wireless NIC sends the frame over the air to the receiver in the wireless router. In this case the wireless router is the default, that is, the gateway router.

[10]Wikipedia, *Category 5 cable*, https://en.wikipedia.org/wiki/Category_5_cable
[11]Wikipedia, *Category 6 cable*, https://en.wikipedia.org/wiki/Category_6_cable

6. The default router opens the datagram and determines the destination IP address. The router uses its own routing table to identify the IP address of the next router that will take the frame on the next step of its journey. The router then re-encapsulates the frame in a new datagram that contains its own MAC as the source and the MAC address of the next router and then sends it on through the appropriate interface. The router performs its routing task at layer 3, the Internet layer.

Switches are invisible to all protocols at layers 2 and above, so they do not affect the transmission of data in any logical manner. The function of switches is merely to provide a simple means to connect multiple hosts into a single physical network via lengths of network cable.

There is also an OSI network model, but that is more complex than the TCP/IP model and that additional complexity adds nothing to our understanding. The OSI model is irrelevant for our needs.

# CIDR – Network notation and configuration

CIDR stands for Classless Inter-Domain Routing.[12] It defines a notation methodology for network addressing that is used to specify the network portion of an IP address.

## Network classes

Before examining how CIDR actually works, let's first look at the classful network notation that CIDR replaces. Introduced in 1981, the classful methodology defined five network classes to be used for identification and addressing devices on the Internet. The network class is defined by the four leading bits of the address. Figure 12-2 shows the five network classes defined by classful network addressing, including both the subnet mask and CIDR notation for each class.

[12]Wikipedia, *Classless Inter-Domain Routing*, https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

| Class | Start | End | Subnet Mask | CIDR | No of Networks | IP Addresses / Network |
|---|---|---|---|---|---|---|
| A | 0.0.0.0 | 127.255.255.255 | 255.0.0.0 | /8 | 128 | 16,177,216 ($2^{24}$) |
| B | 128.0.0.0 | 191.255.255.255 | 255.255.0.0 | /16 | 16,384 | 65,536 ($2^{20}$) |
| C | 192.0.0.0 | 223.255.255.255 | 255.255.255.0 | /24 | 2,097,152 | 256 ($2^{8}$) |
| D | 224.0.0.0 | 239.255.255.255 | | | Undefined | Undefined |
| E | 240.0.0.0 | 255.255.255.255 | | | Undefined | Undefined |

***Figure 12-2.*** *Classful Internet addressing defines five classes*

Classes A, B, and C are the commonly used unicast address ranges that were assigned to organizations. Unicast means that the data packets are sent to a single target host. Class D was the so-called multicast range of addresses. In this range, data packets would be sent to all hosts on a defined network. This range of IP addresses was essentially unused. The class E address range was reserved for future expansion but was also never used.

Note that there are only three possible subnet masks that match each class of the classful networks, 255.0.0.0 (8 bits), 255.255.0.0 (16 bits), and 255.255.255.0 (24 bits), divided on the octet boundaries. This is one of the limiting factors in public address allocation due to the relatively limited number of networks that the classes define.

Unfortunately, classful networking assignments led to major waste. Organizations would apply for a number of addresses, but if they needed more than the number of addresses in a class C network, for example, they would be applied for and be assigned an entire class B network whether they needed all of the addresses in that network or not. The same is true for class B networks; a few large organizations needed more than a class B network, so they were assigned class A networks. Thus, a few large organizations became assigned very large numbers of IP addresses.[13] See RFC790, "Historic allocation of class A networks,"[14] for the complete list of the assigns of the current /8 blocks and historical class A networks.

[13]Wikipedia, *List of assigned /8 blocks networks*, https://en.wikipedia.org/wiki/List_of_assigned_/8_IPv4_address_blocks

[14]The Internet Engineering Task Force (IETF®): RFC790 Historic allocation of class A networks

The four leading (leftmost) bits of the address define the class of the network, not the subnet mask or the CIDR equivalent of the subnet mask. In practical terms, this meant that large networks could not be broken down into smaller subnets at the Internet level because the Internet routers could only have a single route to each assigned classful network. Further, although the large, classful networks could be divided into subnets by the organizations that owned them, routing packets to other geographical locations on the same network then required the organization to use private internal networks or public VPNs at a very high cost premium.

For a simple example, imagine that a company that has six departments and requires about 400 IP addresses for each. This requires more than a single class C network of 256 IP addresses, a total of 2400 addresses. The company has a class B network of 65,536 addresses assigned to it. As a result, the remaining 63,136 IP addresses would be wasted because they could not be assigned to other organizations.

---

**Note** For the purposes of these experiments, it is necessary to use a portion of the current private 10.0.0.0/8 CIDR block of addresses as if it were a public class B address. This is to protect public addresses that may belong to some real organization.

---

The **sipcalc** command provides a great deal of information about an IP address or address ranges. As you will see later, it also has the capability to generate a list of subnets in a given address range given a subnet mask. You may have to install the **sipcalc** program; it was not installed by default on my Fedora system.

## EXPERIMENT 12-3

First, as the root user, install the **sipcalc** RPM package.

```
[root@studentvm1 ~]# dnf -y install sipcalc
```

Now as the student user, we can use **sipcalc** to explore IP addressing.

Use the **sipcalc** CLI program to provide the network data for this randomly selected class B network from the pseudo-public address range.

```
[student@studentvm1 ~]$ sipcalc 10.125.0.0/16
-[IPv4 : 10.125.0.0/16] - 0
```

```
[CIDR]
Host address            - 10.125.0.0
Host address (decimal)  - 175964160
Host address (hex)      - A7D0000
Network address         - 10.125.0.0
Network mask            - 255.255.0.0
Network mask (bits)     - 16
Network mask (hex)      - FFFF0000
Broadcast address       - 10.125.255.255
Cisco wildcard          - 0.0.255.255
Addresses in network    - 65536
Network range           - 10.125.0.0 - 10.125.255.255
Usable range            - 10.125.0.1 - 10.125.255.254
```

The output from the **sipcalc** command shows, among other things, the network address, the netmask, the network address range, as well as the available addresses in that range. The address 10.125.0.0 is the network address, and 10.125.255.254 is the broadcast address for this network. Those two addresses cannot be used for hosts.

Another option would have been to assign multiple class C networks to the company. That would significantly reduce the number of wasted IP addresses, but configuring the routing for this organization would be more complex than it would otherwise need to be with a single network. This option would also reduce the number of class C address blocks available for other organizations.

## Along came a CIDR

Classless Inter-Domain Routing[15] (CIDR) notation was introduced in 1993 as a means of extending the lifetime of IPv4[16] which was running out of assignable addresses. It accomplishes this by making it possible for organizations to more efficiently utilize the public IPv4 address ranges assigned to them and by opening up some previously reserved address ranges.

[15]Wikipedia, *Classless Inter-Domain Routing*, https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

[16]Bandel, David A., *Linux Journal, CIDR: A Prescription for Shortness of Address Space*, www.linuxjournal.com/article/3017. Linux Journal has ceased publication. This article is no longer available as their web site is no longer on line. The article can be viewed at this location: http://www.ipst-info.net/LinuxJournal/LJ/056/3017.html

In 1996, RFC1918 enhanced CIDR with the assignments of reserved, externally non-routable networks in each of the old A, B, and C class ranges. These private networks,[17] shown in Figure 12-3, can be used freely by any organization for their internal networks; no longer is it necessary for every computer to have an assigned public IP address. This feature provides a significant portion of the solution to multiple problems.

| CIDR Block | Address Range | No. of IP Addresses |
|---|---|---|
| 10.0.0.0/8 | 10.0.0.0 - 10.255.255.255 | 16,777,216 |
| 172.16.0.0/12 | 172.16.0.0 - 172.31.255.255 | 1,048,576 |
| 192.168.0.0/16 | 192.168.0.0 - 192.168.255.255 | 65,536 |

***Figure 12-3.*** *IPv4 address ranges reserved for use as private internal networks*

The use of these private internal networks allows organizations to be assigned one or possibly a few public IP addresses for access to the outside Internet while providing large private address spaces for internal networks. To be absolutely clear, each of these address ranges can be used by many different organizations because these private network addresses are not routable through the Internet; of course organizations can route internally between private networks.

Returning to our example company, let's make the assumption that it only requires a single public IP address to connect it to the outside world. The company's Internet provider only assigns minimum blocks of four addresses, two of which are reserved for the network address and the broadcast address, thus leaving two usable addresses. This provides a balance between unusable addresses due to excessive subnetting, wasted addresses, and cost to the customer.

[17]The Internet Engineering Task Force (IETF®), *RFC1918 Address Allocation for Private Internets*, https://tools.ietf.org/html/rfc1918

## EXPERIMENT 12-4

As the student user, we will now use **sipcalc** to explore using the smallest assignable IP address space to our fictional organization. Assume that the ISP assigns the company a public network address, 10.125.16.32/30. Remember that, for this example, we are using part of the 10.0.0.0/8 private network as if it were public. This assignment provides the company with the following public network which uses a network mask of 30 bits.

```
[student@studentvm1 ~]$ sipcalc 10.125.16.32/30
-[IPv4 : 10.125.16.32/30] - 0

[CIDR]
Host address            - 10.125.16.32
Host address (decimal)  - 175968288
Host address (hex)      - A7D1020
Network address         - 10.125.16.32
Network mask            - 255.255.255.252
Network mask (bits)     - 30
Network mask (hex)      - FFFFFFFC
Broadcast address       - 10.125.16.35
Cisco wildcard          - 0.0.0.3
Addresses in network    - 4
Network range           - 10.125.16.32 - 10.125.16.35
Usable range            - 10.125.16.33 - 10.125.16.34
```

This provides our organization with four IP addresses. Two of these addresses are used by the network address and the broadcast address, which leaves two IP addresses for use by our network, 10.125.16.33 and 10.125.16.34.

Our example company is free to choose to use any of the private network ranges for their internal networks. They can use Network Address Translation (NAT) to access the outside world from their internal, private network.

At first glance, the straightforward thing to do might be to choose a network from the private 172.16.0.0/12 range to provide a large enough range for a single internal network. For our example, they could choose the 172.16.0.0/12 network which would provide the following internal network space for them.

```
[student@studentvm1 ~]$ sipcalc 172.16.0.0/12
-[IPv4 : 172.16.0.0/12] - 0

[CIDR]
Host address            - 172.16.0.0
Host address (decimal)  - 2886729728
Host address (hex)      - AC100000
Network address         - 172.16.0.0
Network mask            - 255.240.0.0
Network mask (bits)     - 12
Network mask (hex)      - FFF00000
Broadcast address       - 172.31.255.255
Cisco wildcard          - 0.15.255.255
Addresses in network    - 1048576
Network range           - 172.16.0.0 - 172.31.255.255
Usable range            - 172.16.0.1 - 172.31.255.254
```

Note that this network does not conform to the old class B network as it has fewer network bits in the netmask, thus providing more space for host address bits. The 12 network bits leaves 20 bits for hosts or 1,048,576 hosts. That is far more hosts available than an old class B network would provide for a network. It is also more space than the organization actually needs for its network.

## Variable Length Subnet Masking

CIDR also brings with it a new approach to the old netmask, called Variable Length Subnet Masking, or VLSM. The use of a 12-bit netmask for the private address range defined by the CIDR block in Experiment 12-4 hints at this.

VLSM allows our example company to easily create more manageable subnets from the large private address space available to them by adding bits to the netmask. Using the 12-bit netmask encompasses this entire available private address range, so in order to be more conservative about the address space that the company actually needs, they decide to increase the number of bits in the netmask they will use.

The **`sipcalc -s xx`** command, where **xx** is the number of bits in the subnet mask, can be used to calculate the subnets in this private address range.

## EXPERIMENT 12-5

Perform this experiment as the student user. Calculate the 16 subnets of 172.16.0.0/12 that have a 16-bit subnet mask.

```
[student@studentvm1 ~]$ sipcalc 172.16.0.0/12 -s 16
-[IPv4 : 172.16.0.0/12] - 0

[Split network]
Network                  - 172.16.0.0      - 172.16.255.255
Network                  - 172.17.0.0      - 172.17.255.255
Network                  - 172.18.0.0      - 172.18.255.255
Network                  - 172.19.0.0      - 172.19.255.255
Network                  - 172.20.0.0      - 172.20.255.255
Network                  - 172.21.0.0      - 172.21.255.255
Network                  - 172.22.0.0      - 172.22.255.255
Network                  - 172.23.0.0      - 172.23.255.255
Network                  - 172.24.0.0      - 172.24.255.255
Network                  - 172.25.0.0      - 172.25.255.255
Network                  - 172.26.0.0      - 172.26.255.255
Network                  - 172.27.0.0      - 172.27.255.255
Network                  - 172.28.0.0      - 172.28.255.255
Network                  - 172.29.0.0      - 172.29.255.255
Network                  - 172.30.0.0      - 172.30.255.255
Network                  - 172.31.0.0      - 172.31.255.255
```

Use **sipcalc** to calculate the number of addresses provided by various numbers of bits in the netmask of the 172.16.0.0/12 network. You should be able to determine and verify the data shown in Figure 12-4.

| Bits in netmask | Number of addresses |
|---|---|
| 12 | 1,048,576 |
| 16 | 65,536 |
| 17 | 32,768 |
| 18 | 16,384 |
| 19 | 8,192 |
| 20 | 4,096 |

***Figure 12-4.*** *Number of addresses in various subnet ranges for network 172.16.0.0/12*

As mentioned before, the company currently needs about 2400 IP addresses. To allow plenty of room for growth while reducing the total number of addresses to a manageable level, the company chooses to use a 19-bit netmask that provides 8192 addresses. They calculate the available 19-bit subnets using **sipcalc**.

```
[student@studentvm1 ~]$ sipcalc 172.16.0.0/12 -s 19
-[IPv4 : 172.16.0.0/12] - 0

[Split network]
Network                - 172.16.0.0      - 172.16.31.255
Network                - 172.16.32.0     - 172.16.63.255
Network                - 172.16.64.0     - 172.16.95.255
Network                - 172.16.96.0     - 172.16.127.255
Network                - 172.16.128.0    - 172.16.159.255
Network                - 172.16.160.0    - 172.16.191.255
Network                - 172.16.192.0    - 172.16.223.255
Network                - 172.16.224.0    - 172.16.255.255
<snip>
Network                - 172.31.96.0     - 172.31.127.255
Network                - 172.31.128.0    - 172.31.159.255
Network                - 172.31.160.0    - 172.31.191.255
Network                - 172.31.192.0    - 172.31.223.255
Network                - 172.31.224.0    - 172.31.255.255
```

The company randomly decides to use the 172.30.64.0/19 subnet. So their network specification can be calculated.

```
[student@studentvm1 ~]$ sipcalc 172.30.64.0/19
-[IPv4 : 172.30.64.0/19] - 0

[CIDR]
Host address            - 172.30.64.0
Host address (decimal)  - 2887663616
Host address (hex)      - AC1E4000
Network address         - 172.30.64.0
Network mask            - 255.255.224.0
Network mask (bits)     - 19
Network mask (hex)      - FFFFE000
Broadcast address       - 172.30.95.255
Cisco wildcard          - 0.0.31.255
Addresses in network    - 8192
Network range           - 172.30.64.0 - 172.30.95.255
Usable range            - 172.30.64.1 - 172.30.95.254
```

Of course this is only one possible 19-bit subnet out of 128 in the private address range. The company could have chosen any of the 19-bit subnets calculated in Figure 12-5, any of which would work equally well.

Another option would be to use the 192.168.0.0/16 private address range and select one of the 19-bit subnets available in that range. I leave the task of determining how many and which subnets would be available in that range as an exercise for the reader.

---

Using CIDR notation along with the reorganization of previously allocated addresses by CIDR block, as well as the use of VLSM, provides more usable public IP addresses and increased flexibility in the assignment of public addresses. The design of CIDR notation with VLSM respects the old classful networking scheme while providing significantly more flexibility and IP address availability for private internal use by organizations of all sizes. Private address spaces as well as assigned public address spaces can be easily split into subnets by adding bits to the netmask without consideration for network classes.

CIDR notation can be used when referring to classful networks but only as a notational shorthand.

# DHCP client configuration

Each network interface controller (NIC) on your computer provides a physical connection to your network. Most computers have only one NIC, while others may have several. Laptops usually have a NIC for a wired connection and a NIC for a wireless connection. Some laptops may also have a NIC for a cellular network connection. Some Linux desktop or tower computers have multiple wired NIC cards and are used as inexpensive routers for internal networks; such is the case with a couple of my own systems.

In most instances, and with current releases of Fedora, 29, 30, and later, the default is to use Dynamic Host Configuration Protocol[18] (DHCP) configuration for all network interfaces. This requires a DHCP server to be located on the local network. The virtual router on our virtual network also provides DHCP services.

The DHCP server can provide a large number of network configuration data components, but those in the following list are the only ones that are absolutely required. These are the minimum data required for a host to access the network:

- An IP address
- The IP address of the router/gateway device
- The IP address of at least one name server

Some other configuration data that the DHCP server might provide are listed as follows but not limited to

- Up to two additional name server IP addresses
- The domain name of the local network so that using a command like ping does not require typing the complete domain name
- The subnet mask

[18]Wikipedia, *Dynamic Host Configuration Protocol*, https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

# NIC naming conventions

The naming conventions for network interface controllers used to be simple, uncomplicated, and, I thought, easy. Using ethX made sense to me and was easy to type. It also did not require extra steps to figure out what long and obscure name belonged to a NIC. Unfortunately, adding a new NIC could force the renaming of existing ones, causing issues with startup configuration of all the NICs.

That has all changed – more than once. After a short stint with some very long and unintelligible NIC names that apparently made some sense to a small group of programmers, we now have a third set of naming conventions which seemed only marginally better until I came to understand it better.

## How it works – sort of

The udev device manager detects when a new device has been added to the system, such as a new NIC, and creates a rule to identify and name it if one does not already exist. The details of how this works have changed in more recent versions of Fedora, CentOS, and RHEL.

During the early part of the startup phase, the Linux kernel via udev identifies connected devices including network interface controllers. At this stage the devices are still known by their traditional names of ethX. A very short time after that, systemd renames the devices according to a series of hierarchical naming schemes.

**EXPERIMENT 12-6**

Perform this experiment as root.

```
[root@studentvm1 ~]# dmesg | grep eth
[    4.791454] e1000 0000:00:03.0 eth0: (PCI:33MHz:32-bit) 08:00:27:e1:0c:10
[    4.791520] e1000 0000:00:03.0 eth0: Intel(R) PRO/1000 Network Connection
[    5.294678] e1000 0000:00:03.0 enp0s3: renamed from eth0
```

So this shows that at a little over 4 seconds into the Linux startup sequence, the eth0 network device is located and less than a second later eth0 is renamed to enp0s3.

Chapter 11 of the RHEL 7 "Networking Guide" describes how this renaming works. The following is excerpted from that document:

- Scheme 1: Names incorporating Firmware or BIOS provided index numbers for on-board devices (example: eno1), are applied if that information from the firmware or BIOS is applicable and available, else falling back to scheme 2.
- Scheme 2: Names incorporating Firmware or BIOS provided PCI Express hot-plug slot index numbers (example: ens1) are applied if that information from the firmware or BIOS is applicable and available, else falling back to scheme 3.
- Scheme 3: Names incorporating physical location of the connector of the hardware (example: enp2s0), are applied if applicable, else falling directly back to scheme 5 in all other cases.
- Scheme 4: Names incorporating interface's MAC address (example: enx78e7d1ea46da), is not used by default, but is available if the user chooses.
- Scheme 5: The traditional unpredictable kernel naming scheme, is used if all other methods fail (example: eth0).

In scheme 1, eno is used in which the letter "o" means on-board, that is, an integral part of the motherboard. In scheme 2, ens is used and the letter "s" indicates that the device is plugged into a PCI Express slot. Back in Experiment 12-1, we looked at the VM's installed NIC and found that it was named enp0s3. This is consistent with naming scheme 3 which is based on the physical location of the connectors, whether virtual or physical, on the hardware.

The primary function of the revised naming schemes is to provide a consistent set of NIC names so that installing a new NIC or even just a reboot would not cause the NIC names to change. This by itself is well worth the changes. I have had plenty of opportunity to fight with apparently random renaming of several ethX devices on a single host. That was much less fun than learning the revised naming schemes.

The newest NIC naming conventions are used by RHEL 7 and 8, CentOS 7 and probably CentOS 8 when it is released, and the current releases of Fedora. The NIC naming conventions for these distributions are described in detail in the RHEL 7

document "Networking Guide[19]" along with a description of how the names are derived. Using the NetworkManager tools to manage networking is covered in the RHEL 8 document, "Configuring and Managing Networking.[20]"

## NIC configuration files

By default, all current releases of Fedora default to DHCP configuration. No options are provided during installation to configure any aspect of the network interface. Starting with Fedora 29, Linux hosts using DHCP for network configuration no longer require interface configuration files if all of the DHCP default configurations are sufficient.

However, non-standard configuration of the NICs for each network connection is still accomplished with ifcfg-X files in the /etc/sysconfig/network-scripts directory. Each NIC can have an interface configuration file named ifcfg-enp0s3, or something similar, where enp0s3 is the interface name assigned by the udev daemon. Each interface configuration file is bound to a specific physical NIC. Using the **`nmcli`** tool (network manager command-line interface) to configure an interface creates the interface configuration file for that interface.

The current strategy is to use the contents of the interface configuration files to generate the rules. However, if an interface configuration file does not exist, plugging in a new device or connecting with a new wireless network causes udev to notify NetworkManager of the new device or wireless connection. Then, in Fedora up through release 28, NetworkManager creates the new interface configuration file. As of Fedora 29 and higher, the Network Manager only creates the connection but does not create an interface configuration file.

The udev daemon creates an entry for each NIC installed in the system in the network rules file. The Network Manager uses these entries, along with information in the interface configuration files in the /etc/sysconfig/network-scripts/ directory to initialize each NIC.

[19]Red Hat, *Networking Guide*, https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/ch-consistent_network_device_naming

[20]Red Hat, *Configuring and Managing Networking*, https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/Configuring-Networking-with-nmcli_configuring-and-managing-networking

Other distributions may keep their network configuration files in the /etc/NetworkManager/system-connections directory, with the name of the network as the file name. For example, my System76 laptop uses POP!_OS which is based on Ubuntu. The /etc/NetworkManager/system-connections directory on that laptop contains files for the wired network as well as each of the wireless networks I have connected with. The structure of these files is different from the ifcfg files we will explore later in this chapter, but they are in ASCII plain text format and are readable and easily understandable.

## Create an interface configuration file

Let's do a quick experiment to see why creating an interface configuration file might be a good idea.

**EXPERIMENT 12-7**

Perform this experiment as the root user.

Suppose that our VM host is being attacked in some way via the network. One way to ensure that a network attack is thwarted is to turn off the host's network connection. If we were local to a host and it was a physical device, then we could unplug the network cable. But when remote we need to turn down the network interface. We can use the **ifdown** or the **ip** commands to do that.

First verify that enp0s3 currently has an IP address assigned.

```
[root@studentvm1 ~]# ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 08:00:27:e1:0c:10 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 1143sec preferred_lft 1143sec
    inet6 fe80::c33:30e7:314e:e83e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Now turn enp0s3 off.

```
[root@studentvm1 ~]# ip link set enp0s3 down
```

And verify.

```
[root@studentvm1 ~]# ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel state DOWN group
default qlen 1000
    link/ether 08:00:27:e1:0c:10 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 1118sec preferred_lft 1118sec
[root@studentvm1 ~]#
```

The link is still up and active. And that is the problem.

After some experimentation of my own, I have learned that it is not possible to turn off an active link or to turn on an inactive one unless there is an interface configuration file for it. So let's create one.

## EXPERIMENT 12-8

This experiment must be performed as the root user.

We could actually create an interface configuration file, ifcfg-enp0s3, for this interface using an editor like Vim, but that is actually the hard way. Instead we will use the nmcli (NetworkManager command-line interface) to do this.

Open a terminal session and make /etc/sysconfig/network-scripts the PWD. Then list the contents. The directory should be empty.

```
[root@studentvm1 ~]# cd /etc/sysconfig/network-scripts/ ; ll
total 4
[root@studentvm1 network-scripts]#
```

The following command adds the new connection and saves the interface configuration file in /etc/sysconfig/network-scripts/. Note that it is not necessary for this to be the PWD, it just makes it easier to see the before and after, with and without the file.

```
[root@studentvm1 network-scripts]# nmcli connection add save yes type
ethernet ifname enp0s3 con-name enp0s3
Connection 'enp0s3' (9e08333c-4458-4c7e-9632-16e3afe41f93) successfully
added.
[root@studentvm1 network-scripts]# ll
```

```
total 8
-rw-r--r--  1 root root 282 May  8 16:09 ifcfg-enp0s3
[root@studentvm1 network-scripts]#
```

Now stop the enp0s3 network connection and verify that it is now down. The lack of an IP address indicates that the interface is down.

```
[root@studentvm1 network-scripts]# ip link set enp0s3 down
```

It can take a few moments for the link to go down, so if the next command still shows it as up, wait a few seconds and try it again.

```
[root@studentvm1 network-scripts]# ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel state DOWN group
default qlen 1000
    link/ether 08:00:27:e1:0c:10 brd ff:ff:ff:ff:ff:ff
[root@studentvm1 network-scripts]#
```

Now bring interface enp0s3 back up again.

I find that remembering these commands is somewhat difficult because I don't use them very often. Using tab completion will always give me a hint as to what the next possible entry might be.

# The interface configuration file

Now let's look at the contents of the ifcfg-enp0s3.

**EXPERIMENT 12-9**

As root, cat the /etc/sysconfig/network-scripts/ifcfg-enp0s3 file.

```
[root@studentvm1 network-scripts]# cat ifcfg-enp0s3
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
```

```
DEFROUTE=yes
IPv4_FAILURE_FATAL=no
IPv6INIT=yes
IPv6_AUTOCONF=yes
IPv6_DEFROUTE=yes
IPv6_FAILURE_FATAL=no
IPv6_ADDR_GEN_MODE=stable-privacy
NAME=enp0s3
UUID=4a527023-daa4-4dfb-9775-dbe9fb00fb0b
DEVICE=enp0s3
ONBOOT=yes
```

This is a typical interface configuration file as created by the **`nmcli`** command. It contains the bare minimum necessary for such a file. The primary purpose for creating this file is that it allows the SysAdmin to control the interface.

Figure 12-5 lists the configuration options shown previously and some common ones that aren't in that file we just created, along with some brief explanations for each. Many of the IPv6 options are similar to those of the similarly named IPv4 ones. Note that local configuration variable settings override those provided by a DHCP server.

| Configuration variable | Description |
|---|---|
| TYPE | Type of network such as Ethernet or token ring. |
| PROXY_METHOD | Proxy configuration method. “none” means no proxy is in use. |
| BROWSER_ONLY | Whether a proxy configuration is for browsers only. |
| BOOTPROTO | Options are dhcp, bootp, none, and static. |
| DEFROUTE | This interface is the default route for this host to the outside world. |
| IPv4_FAILURE_FATAL | If this is set to “no” failure to obtain an IPv4 connection will not affect any attempt to make an IPv6 connection. |
| IPv6INIT | Whether to initialize IPv6 or not. The default is yes. |
| IPv6_AUTOCONF | Yes means use DHCP for configuration of IPv6 on this interface. |
| IPv6_DEFROUTE | This interface is the IPv6 default route for this host to the outside world. |
| IPv6_FAILURE_FATAL | If this is set to “no” failure to obtain an IPv6 connection will not affect any attempt to make an IPv4 connection. |
| IPv6_ADDR_GEN_MODE | Configure IPv6 Stable Privacy addressing. |
| NAME | The interface name, such as enp0s3. |
| UUID | A Universally Unique Identifier for the interface. It is created with a hash of the interface name. |
| DEVICE | The name of the interface to which this configuration file bound. |
| ONBOOT | If yes, this starts the interface at boot (really startup time. If no, the interface is not started until a user logs in at the GUI or manually starts the inteface. I always set this to yes if it is not already. |
| | |
| HWADDR | The MAC address of the interface. |
| DNS1, DNS2 | Up to two name servers may be specified. |
| USERCTL | Specifies whether non-privileged users may start and stop this interface. Options are yes/no. |
| IPADDR | The IP Address assigned to this NIC |
| BROADCAST | The broadcast address for this network such as 10.0.2.255 |
| NETMASK | The netmask for this subnet such as 255.255.255.0 |
| NETWORK | The network ID for this subnet such as 10.0.2.0 |
| SEARCH | The DNS domain name to search when doing lookups on unqualified hostnames such as using studentvm1 instead of studentvm1.example.com. |

***Figure 12-5.*** *Some of the more common configuration items found in network interface configuration files*

| GATEWAY | The network router or default gateway for this subnet, such as 10.0.2.1. |
|---|---|
| PEERDNS | The yes option indicates that /etc/resolv.conf is to be modified by inserting the DNS server entries specified by DNS1 and DNS2 options in this file. No means do not alter the resolv.conf file. Yes is the default when DHCP is specified in the BOOTPROTO line. |

***Figure 12-5.*** *(continued)*

The lines in the interface configuration files are not sequence sensitive and work just fine in any order. By convention the option names are in uppercase and the values are in lowercase. Option values can be enclosed in quotes, but that is not necessary unless the value is more than a single word or number.

Let's make some minor changes to the interface configuration file. We can do this in three ways at the command line, by editing the files directly, by using the **nmtui** program, and by using the **nmcli** command. We will use the command line for this.

## EXPERIMENT 12-10

Perform this experiment as the root user. Use the already open root terminal session with /etc/sysconfig/ as the PWD.

We will add an external DNS server as DNS2. It is necessary to provide DNS1 in the interface configuration file due to the fact that the **nmcli** program does not allow for adding only DNS2. The IP address of the virtual router which is also our virtual DHCP server is 10.0.2.1, and the IP address of one of the Google public name servers is 8.8.8.8.

```
[root@studentvm1 network-scripts]# nmcli connection modify enp0s3 IPv4.dns
"10.0.2.1 8.8.8.8"
```

Now view the contents of the ifcfg-enp0s3 file.

```
[root@studentvm1 network-scripts]# cat ifcfg-enp0s3
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
DEFROUTE=yes
IPv4_FAILURE_FATAL=no
IPv6INIT=yes
```

```
IPv6_AUTOCONF=yes
IPv6_DEFROUTE=yes
IPv6_FAILURE_FATAL=no
IPv6_ADDR_GEN_MODE=stable-privacy
NAME=enp0s3
UUID=4a527023-daa4-4dfb-9775-dbe9fb00fb0b
DEVICE=enp0s3
ONBOOT=yes
DNS1=10.0.2.1
DNS2=8.8.8.8
```

The two new DNS entries are at the bottom of the ifcfg file. However, the VirtualBox DHCP server settings use the physical host's DNS settings. Those are configured first, and then these entries are added at the bottom of /etc/resolv.conf making this change irrelevant in the VM. However, it would work fine when working on a physical host and not a VM. I did try to find a way to make VirtualBox ignore the host's settings but was unable. Of course this does make sense because the DNS servers of the host computer are generally the ones that would also be available to the virtual network.

We cannot test this in our VM – at the moment – so edit the /etc/sysconfig/network-scripts/ifcfg-enp0s3 file and remove the two line items that were just added.

We do have an alternate, brute force method for setting and keeping the desired name servers listed in /etc/resolv.conf.

First edit /etc/resolv.conf, and whatever was there to start, change it to look like this.

```
# Generated by NetworkManager
nameserver 10.0.2.1
nameserver 8.8.8.8
```

Now we can use a new tool to make this file immutable so that it cannot be changed by any process or any user including root. If we do not do this, it will change at the next boot. The **chattr** (change attributes) command can set some file attributes that are not accessible using the normal file mode settings. Most of these attributes are rarely used, but I do use the -i (immutable) attribute for times like this.

With /etc as the PWD, enter the following commands.

```
[root@studentvm1 etc]# chattr +i resolv.conf ; lsattr resolv.conf
----i---------e---- resolv.conf
```

Notice the **i** in the attribute list that tells us the file is now immutable. Leave it that way for now.

In Volume 3 of this course, we will set up DHCP and name servers that will perform as we want and which will enable us to bypass the DHCP and name services of the virtual network. The limitations I discovered while creating this experiment are a good reason to consider using our own servers – to ensure that we have total control over how our network environment works.

For more information about configuration files, the file /usr/share/doc/initscripts/sysconfig.txt contains a list of all the files that can be found in the /etc/sysconfig directory and its subdirectories. This includes the network ifcfg-<interface> files. The descriptions of each file list all of the possible configuration variables and their possible values along with terse explanations.

## The network file

There is one old and now deprecated file you might encounter. The network file usually contains only a single comment line for current releases of Fedora, RHEL, and CentOS. It is located in /etc/sysconfig and was used in the past to enable or disable networking. It was also used to set the networking hostname as shown in the following example.

```
NETWORKING=yes
HOSTNAME=host.example.com
```

This file has been present but unused in Fedora since release 19 and is still present in Fedora 30. It is still used in RHEL/CentOS 6.x but no longer in RHEL/CentOS 7.x. The network hostname is now set in the /etc/hostname file.

## The route-<interface> file

The only other network configuration file you might find in /etc/sysconfig/network-scripts is the route-<interface> file. If you want to set up static routes on a multi-homed[21] system, you would create a route file for each interface. For example, a file might be named route-enp0s3 and would contain information defining routes to entire networks or specific hosts for that interface. Each interface would have its own route file.

[21]A system with multiple NICs, typically used as a router

The use of this file is uncommon in Linux clients. However, if you are using the host as a router or have some special routing needs, you would use this file to configure those complex routes. Therefore, the details of this routing configuration file are beyond the scope of this course. It will be covered in the next course in this series, *Advanced Linux System and Server Administration.*

## Other network files

The /etc/sysconfig/network-scripts directory used to contain many other files, all of which were usually executable BASH scripts rather than configuration files. This was, to me at least, one of the bothersome exceptions to the Linux Filesystem Hierarchical Standard[22] (FHS) which explicitly states that only configuration files and not executable files are to be located in the /etc tree. The use of NetworkManager has finally resulted in removal of all the executable files from this directory and, as of Fedora 29, used solely for configuration files.

# Network startup

With the advent of wireless networks and mobile devices, reconfiguring the network interfaces for each new wireless network became very complicated and time-consuming, which is not a good thing for people who are less technical. It could also be a problem when adding a new NIC to a server or other host with multiple network interfaces.

The NetworkManager service starts the network services during startup and provides a management interface while the host is running.

## The NetworkManager service

Red Hat introduced the Network Manager in 2004 as a way to simplify and automate network configuration and connections, especially wireless connections. The intent is to prevent the user from having to manually configure each new wireless network before using it.

[22]See Volume 1, Chapter 19, for a description of the Linux FHS.

The NetworkManager service makes network management better and easier for non-technical users because it integrates well with udev and D-Bus to deal with pluggable devices and various wireless networks. It requires some adjustment by the Linux SysAdmins who have been around for a while because many of the configuration functions we have been familiar with are now handled by a new layer of configuration files, scripts, and commands. I found it fairly easy to make this switch.

Our primary interface with NetworkManager are the **`ip`** and **`nmcli`** commands. We can also use **`nmtui`**, the NetworkManager text user interface, a menu driven interface that I find cumbersome compared to the command line. Others, like Jason, my technical reviewer, find it to be "awesome." You should try it yourself, but for this course, we will stick to the command line.

## Name services

Surfing the Web is fun and easy, but think what it would be like if you had to type in the IP address of every web site you wanted to view. For example, locating a web site would look like this when you type it in, https://93.184.216.34, which would be nearly impossible for most of us to remember. Of course using bookmarks would help, but suppose your friend tells you about a cool new web site and tells you to go to 93.184.216.34. How would you remember that? Telling someone to go to "example.com" is far easier to remember.

The Domain Name System provides the database to be used in the translation from human-readable hostnames, such as www.example.com, to IP addresses, like 93.184.216.34 so that your Internet-connected computers and other devices and access them. The primary function of the BIND software, the Berkeley Internet Name Domain, is that of a domain name resolver which utilizes that database. There is other name resolver software, but BIND is currently the most widely used DNS software on the Internet. I will use the terms name server, DNS, and resolver pretty much interchangeably throughout this article.

Without these name resolver services, it would be nearly impossible to surf the Web as freely and easily as we do. As humans, we tend to do better with names like example.com, while computers do much better with numbers like 93.184.216.34. So we need a translation service to convert the names that are easy for us to the numbers that are easy for our computers. This process is called name resolution.

In small networks the /etc/hosts file on each host can be used as a name resolver. Maintaining copies of this file on several hosts can become very time-consuming and errors can cause much confusion and wasted time before they are found. I did this for several years on my own network, and it ultimately became too much trouble to maintain even with the usual 8 to 12 computers I have operational. So I ultimately converted to running my own name server to resolve both internal and external hostnames.

Most networks of any size require centralized management of this service with name services software such as the Berkeley Internet Name Domain (BIND). BIND is called that because it was developed by the University of California Berkeley (UCB) in the early 1980s. Hosts use the Domain Name System (DNS) to locate IP addresses from the names given in software such as web browsers, email clients, SSH, FTP, and many other Internet services.

## How a name search works

Let's take a look at a simplified example of what happens when a name request for a web page is made by a client service on your computer. For this example, I will use www.example.com as the web site I want to view in my browser. I also assume that there is a local name server on the network, as is the case with my own network:

1. First, I type in the URL or select a bookmark containing that URL. In this case, the URL is www.example.com.
2. The browser client, whether it is Opera, Firefox, Chrome, Lynx, Links, or any other browser, sends the request to the operating system.
3. The operating system first checks the /etc/hosts file to see if the URL or hostname is there. If so, the IP address of that entry is returned to the browser. If not, we proceed to the next step. In this case we assume that it is not.
4. The URL is then sent to the first name server specified in /etc/resolv.conf. In this case the IP address of the first name server is my own internal name server. For this example, my name server does not have the IP address for www.example.com cached and must look further afield. So we go on to the next step.

5. The local name server sends the request to a remote name server. This can be one of two destination types, one type of which is a forwarder. A forwarder is simply another name server such as the ones at your ISP or a public name server such as Google at 8.8.8.8 or 8.8.4.4. The other destination type is that of the top-level root name servers. The root servers don't usually respond with the desired target IP address or www.example.com; they respond with the authoritative name server for that domain. The authoritative name servers are the only ones that have the authority to maintain and modify name data for a domain.
6. The local name server is configured to use the root name servers so the root name server for the .com top-level domain returns the IP address of the authoritative name server for example.com. That IP address could be for any one of the three (at the time of this writing) name servers, a.iana-servers.net or b.iana-servers.net.
7. The local name server then sends the query to the authoritative name server which returns the IP address for www.example.com.
8. The browser uses the IP address for www.example.com to send a request for a web page which is downloaded to my browser.

One of the important side effects of this name search is that the results are cached for a period of time by my local name server. That means that the next time I, or anyone on my network, want to access example.com, the IP address is probably already stored locally which prevents the need to perform a remote lookup.

## Using the /etc/hosts file

Most computers need little configuration to enable them to access name services. That usually consists of adding the IP addresses of one to three name servers to the /etc/resolv.conf file. And that is typically performed at boot time on most home and laptop computers because they are configured using the DHCP protocol which provides them with their IP address, gateway address, and the IP addresses of the name servers. For hosts that are configured statically, the /etc/resolv.conf file is usually generated during installation from information entered by the SysAdmin during the installation.

Many of the name servers provided by ISPs in my home office and for remote connections in public places such as hotels, coffee shops, and even friends' personal WiFi connections can be unreliable and in some cases can use forwarders that intentionally censor results or redirect queries to pages of advertisements. So I always add at least one of the Google public name servers on my hosts. We did this using brute force in Experiment 12-10. If you are interested in more privacy, you could also try Cloudflare at 1.1.1.1.

Adding entries to the /etc/hosts file can make accessing remote hosts by name possible in the absence of centralized name services. So let's look at the default version of the file and add some hosts to it.

## EXPERIMENT 12-11

Perform this experiment as root. Start by examining the default version of the /etc/hosts file. It has only two lines and those are both for the localhost.

```
[root@studentvm1 ~]# cd /etc ; cat hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.
localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.
localdomain6
```

These entries allow us to address commands using the names for the localhost.

```
[root@studentvm1 etc]# ping -c2 localhost
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.076 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.077 ms

--- localhost ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 43ms
rtt min/avg/max/mdev = 0.076/0.076/0.077/0.008 ms
[root@studentvm1 etc]# ping -c2 localhost4
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.046 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.074 ms

--- localhost ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 31ms
```

```
rtt min/avg/max/mdev = 0.046/0.060/0.074/0.014 ms
[root@studentvm1 etc]# ping -c2 localhost6
PING localhost6(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.066 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.083 ms

--- localhost6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 66ms
rtt min/avg/max/mdev = 0.066/0.074/0.083/0.012 ms
```

That works, but suppose we want to ping the localhost by its actual hostname? For this we will use the current IP address of enp0s3 rather than the loopback address. This IP address may change in the future – and will in the next course.

First verify the current IP address.

```
[root@studentvm1 ~]# ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 08:00:27:e1:0c:10 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 824sec preferred_lft 824sec
    inet6 fe80::68da:2fbe:6325:4f3f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

This shows the IP address for StudentVM1 as 10.0.2.7.

Add the following lines to the /etc/hosts file. I use the comment like I do in Bash scripts, so I know what I did and why at later times and for other SysAdmins who will take over my job in the future. Be sure to use the IP address for your own VM because it can be different.

```
# Added the following lines for testing 2019-05-09
10.0.2.7        studentvm1 svm1 vm1 s1
```

Note that you can have multiple hostnames for each IP address. This makes it possible to use both the full hostname as well as any nicknames you may have for a particular host. Notice that the /etc/hosts file is generally not used to provide services for fully qualified domain names (FQDN) such as studentvm1.example.com; rather, only the unqualified hostnames themselves are typically used.

```
[root@studentvm1 ~]# ping -c2 studentvm1
PING studentvm1 (10.0.2.7) 56(84) bytes of data.
```

```
64 bytes from studentvm1 (10.0.2.7): icmp_seq=1 ttl=64 time=0.069 ms
64 bytes from studentvm1 (10.0.2.7): icmp_seq=2 ttl=64 time=0.088 ms

--- studentvm1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 60ms
rtt min/avg/max/mdev = 0.069/0.078/0.088/0.013 ms
[root@studentvm1 ~]# ping -c2 svm1
PING studentvm1 (10.0.2.7) 56(84) bytes of data.
64 bytes from studentvm1 (10.0.2.7): icmp_seq=1 ttl=64 time=0.081 ms
64 bytes from studentvm1 (10.0.2.7): icmp_seq=2 ttl=64 time=0.098 ms

--- studentvm1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 35ms
rtt min/avg/max/mdev = 0.081/0.089/0.098/0.012 ms
[root@studentvm1 ~]# ping -c2 vm1
PING studentvm1 (10.0.2.7) 56(84) bytes of data.
64 bytes from studentvm1 (10.0.2.7): icmp_seq=1 ttl=64 time=0.085 ms
64 bytes from studentvm1 (10.0.2.7): icmp_seq=2 ttl=64 time=0.079 ms

--- studentvm1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 11ms
rtt min/avg/max/mdev = 0.079/0.082/0.085/0.003 ms
```

Now let's add an entry for the virtual router. Add the following line to the bottom of the /etc/hosts file.

```
10.0.2.1        router gateway
```

Now ping the router.

```
[root@studentvm1 ~]# ping -c2 router
PING router (10.0.2.1) 56(84) bytes of data.
64 bytes from router (10.0.2.1): icmp_seq=1 ttl=255 time=0.254 ms
64 bytes from router (10.0.2.1): icmp_seq=2 ttl=255 time=0.266 ms

--- router ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 31ms
rtt min/avg/max/mdev = 0.254/0.260/0.266/0.006 ms
[root@studentvm1 ~]# ping -c2 gateway
PING router (10.0.2.1) 56(84) bytes of data.
64 bytes from router (10.0.2.1): icmp_seq=1 ttl=255 time=0.246 ms
64 bytes from router (10.0.2.1): icmp_seq=2 ttl=255 time=0.253 ms
```

```
--- router ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 33ms
rtt min/avg/max/mdev = 0.246/0.249/0.253/0.016 ms
```

I highly recommend against adding entries for any hosts outside the local network. It can be done, but that links an IP address to the external hostname on our local systems. The actual IP address may change at a later time, and you will not be able to access the host. It can be difficult to troubleshoot such a problem unless you remember that an entry exists in the localhosts file. Yes, I have done this and it took a couple hours to resolve this problem.

Close the editor session, leaving the changes to /etc/hosts in place.

# Introduction to network routing

Every computer attached to a network requires some type of routing instructions for network TCP/IP packets when they leave the localhost. This is usually very straightforward because many network environments are very simple and there are only two options for departing packets. All packets are sent either to a device on the local network or to some other remote network.

Let's be sure to define the "local" network as the logical, and usually also the physical, network in which the localhost resides. Logically that means the local subnet in which the host is assigned one of the range of the local subnet's IP addresses. Physically that means the host is physically connected to one or more switches that are also connected to the rest of the local network.

## The routing table

All network devices, whether they are hosts, routers, or other types of network nodes such as network-attached printers, need to make decisions about where to route TCP/IP data packets. The routing table provides the configuration information required to make those decisions. The routing table for any host on the network is used to define the single route available to that localhost and to determine whether to send packets to the default gateway router.

For hosts connected to the network using DHCP, the DHCP server provides that configuration information for the default route along with DNS, the hosts' IP address, and possibly other information such as the IP address for a NTP server.

## EXPERIMENT 12-12

Perform this experiment as the student user. Root privileges are not required.

The **route -n** command lists the routing table; the -n option displays the results as IP addresses only and does not attempt to perform a DNS lookup which would replace the IP address with hostnames if they are available.

```
[student@studentvm1 ~]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.1        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
```

The **netstat -rn** command produces very similar results. Both the **netstat** and route commands are obsolete and are replaced by the **ip** command.

The default gateway is always shown with the destination 0.0.0.0 when the -n option is used. If -n is not used, the word "Default" appears in the Destination column of the output. The IP address in the Gateway column is that of the outbound gateway router. The netmask of 0.0.0.0 for the default gateway means that any packet not addresses to the local network or another outbound router by additional entries in the routing table are to be sent to the default gateway regardless of the network class.

The Iface (Interface) column in the output from the **route** command is the name of the outbound NIC, in this case, eno1. For hosts that are acting as routers, there will likely be at least two and sometimes more NICs used. Each NIC used as a route will be connected to a different physical and logical network. The flags in the Flag column indicate that the route is Up (U) and which is the default Gateway (G). Other flags may also be present.

On my own StudentVM1 host, I do have two virtual interfaces. Each interface is on a different network and so each has a different default gateway as shown in the following. My VM is not a router, but it would take very little to turn it into one. We will explore how to configure a Linux host as a router in the next book in this series.

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.1        0.0.0.0         UG    100    0        0 enp0s3
0.0.0.0         192.168.0.254   0.0.0.0         UG    101    0        0 enp0s8
```

```
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
192.168.0.0     0.0.0.0         255.255.255.0   U     101    0        0 enp0s8
```

I like these older commands, but let's explore using the **ip** command.

```
[student@studentvm1 ~]$ ip route
default via 10.0.2.1 dev enp0s3 proto dhcp metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.7 metric 100
```

This result is not as pretty and a bit harder to read because of it, but it does have all of the information we need to describe the routing table.

What happens when there is a network problem and a web site or remote host cannot be reached? We need to do some problem determination. We can use the **traceroute** tool to view the complete route our packets will take to a remote host.

```
[root@studentvm1 ~]# traceroute www.example.org
traceroute to www.example.org (93.184.216.34), 30 hops max, 60 byte packets
 1  studentvm2.example.com (192.168.56.1)  0.323 ms  0.796 ms  0.744 ms
 2  10.0.2.1 (10.0.2.1)  1.106 ms  1.089 ms  1.063 ms
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * *^C
```

That does not help much. This tool used to work very nicely, and it is not deprecated so it should work better than this. We could force it to use ICMP instead of UDP. This seems to work better, but traceroute is still deprecated.

```
[root@studentvm1 ~]# traceroute -I www.example.org
traceroute to www.example.org (93.184.216.34), 30 hops max, 60 byte packets
 1  router.example.com (192.168.56.1)  4.825 ms  4.823 ms  4.808 ms
 2  10.0.2.1 (10.0.2.1)  5.423 ms  5.428 ms  5.408 ms
 3  192.168.0.254 (192.168.0.254)  5.464 ms  5.450 ms  5.439 ms
```

```
 4  rrcs-24-199-159-57.midsouth.biz.rr.com (24.199.159.57)  7.583 ms  9.298 ms  9.653 ms
 5  142.254.207.205 (142.254.207.205)  26.082 ms  26.092 ms  26.068 ms
 6  cpe-174-111-105-178.triad.res.rr.com (174.111.105.178)  24.964 ms  18.706 ms
    18.774 ms
 7  cpe-024-025-062-106.ec.res.rr.com (24.25.62.106)  22.292 ms  20.565 ms  21.236 ms
 8  be31.chrcnctr01r.southeast.rr.com (24.93.64.186)  24.967 ms  26.658 ms  26.683 ms
 9  bu-ether11.atlngamq46w-bcr00.tbone.rr.com (66.109.6.34)  37.310 ms  35.686 ms
    35.631 ms
10  152.195.80.196 (152.195.80.196)  35.450 ms  42.214 ms  42.651 ms
11  152.195.80.141 (152.195.80.141)  42.003 ms  34.385 ms  33.853 ms
12  93.184.216.34 (93.184.216.34)  36.722 ms  32.929 ms  33.293 ms
[root@studentvm1 ~]#
```

Besides, my preferred tool for this is **mtr**. This tool started out as Matt's traceroute because Matt wrote it and it was designed as a dynamic replacement for the old **traceroute** tool. Because Matt no longer maintains this and someone else has taken over, it is now referred to as "my traceroute."

```
[student@studentvm1 ~]$ mtr example.com
                                My traceroute  [v0.92]
studentvm1 (10.0.2.7)                                2019-05-10T09:16:05-0400
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                             Packets               Pings
 Host                                        Loss% Snt  Last   Avg  Best  Wrst  StDev
 1. _gateway                                  0.0%  11   0.3   0.3   0.3   0.4   0.0
 2. wally1.both.org                           0.0%  11   0.5   0.5   0.5   0.6   0.0
 3. rrcs-24-199-159-57.midsouth.biz.rr.com    0.0%  11  29.6   7.8   1.9  29.6   7.6
 4. 142.254.207.205                           0.0%  11  18.9  30.1  11.9  82.3  20.0
 5. cpe-174-111-105-178.triad.res.rr.com      0.0%  11  24.9  28.1  19.9  65.0  12.6
 6. cpe-024-025-062-106.ec.res.rr.com         0.0%  11  24.1  25.9  16.2  39.0   6.5
 7. be31.chrcnctr01r.southeast.rr.com         0.0%  11  29.6  43.2  26.9 108.2  22.3
 8. bu-ether11.atlngamq46w-bcr00.tbone.rr.com 0.0%  10  36.4  38.8  29.5  56.8   8.4
 9. 152.195.80.196                            0.0%  10  32.1  35.7  32.1  39.9   2.9
10. 152.195.80.131                            0.0%  10  31.8  37.7  25.9  47.8   6.1
11. 93.184.216.34                             0.0%  10  36.4  33.1  29.6  40.8   3.4
```

This provides good results and shows us the path that our data packets are taking to the remote host. This is a dynamic display, and it keeps checking the route until you press **q** to quit. Because of this, **mtr** can display statistics for each hop along the way to the destination including response times and packet loss at each intermediate router along the way.

Another thing you might see for any given hop number (the sequential numbers down the left side of the display) is multiple routers indicating that the path to the remote host is not always through the same sequence of routers.

Using the -n option displays only the IP addresses of the routers. The routers shown in your results will be different from my results until the last few hops, as it gets closer to the target host.

```
[student@studentvm1 ~]$ mtr -n example.org
                                 My traceroute  [v0.92]
studentvm1 (10.0.2.7)                          2019-05-10T09:25:41-0400
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                      Packets               Pings
 Host                               Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 10.0.2.1                         0.0%    25    0.3   0.3   0.2   0.4   0.1
 2. 192.168.0.254                    0.0%    25    0.5   0.6   0.4   0.9   0.1
 3. 24.199.159.57                    0.0%    25    3.7   6.3   2.1  13.4   3.4
 4. 142.254.207.205                  0.0%    25   69.3  23.6  12.0  69.3  11.9
 5. 174.111.105.178                  0.0%    25   25.3  24.0  12.6  38.9   6.5
 6. 24.25.62.106                     0.0%    25   28.1  24.9  17.7  36.4   4.5
 7. 24.93.64.186                     0.0%    25   30.4  43.3  26.5 161.2  29.0
 8. 66.109.6.34                      8.3%    24   35.5  47.8  27.5 158.8  30.9
 9. 152.195.80.196                   0.0%    24   45.7  45.7  26.1 156.3  30.5
10. 152.195.80.131                   0.0%    24   37.3  56.8  27.4 238.1  50.8
11. 93.184.216.34                    0.0%    24   32.3  43.0  26.4 142.0  26.9
```

Note the packet loss at hop 8. Although this could indicate a problem, it is more likely that the router is programmed to discard unimportant packets such as ICMP if the router is heavily loaded. If you try this at another time, the packet loss will probably be zero.

Now we can use another tool to learn more about who owns that IP address. Sometimes communicating with the abuse contact for the remote network can be useful in tracking down and stopping some types of attacks. Some contacts will go out of their way to help but, in some areas of the world, don't expect any help.

```
[student@studentvm1 ~]$ whois 93.184.216.34
[Querying whois.ripe.net]
[whois.ripe.net]
% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf
```

```
% Note: this output has been filtered.
%       To receive output for a database update, use the "-B" flag.

% Information related to '93.184.216.0 - 93.184.216.255'

% Abuse contact for '93.184.216.0 - 93.184.216.255' is 'abuse@
verizondigitalmedia.com'

inetnum:        93.184.216.0 - 93.184.216.255
netname:        EDGECAST-NETBLK-03
descr:          NETBLK-03-EU-93-184-216-0-24
country:        EU
admin-c:        DS7892-RIPE
tech-c:         DS7892-RIPE
status:         ASSIGNED PA
mnt-by:         MNT-EDGECAST
created:        2012-06-22T21:48:41Z
last-modified:  2012-06-22T21:48:41Z
source:         RIPE # Filtered

person:         Derrick Sawyer
address:        13031 W Jefferson Blvd #900, Los Angeles, CA 90094
phone:          +18773343236
nic-hdl:        DS7892-RIPE
created:        2010-08-25T18:44:19Z
last-modified:  2017-03-03T09:06:18Z
source:         RIPE
mnt-by:         MNT-EDGECAST

% This query was served by the RIPE Database Query Service version 1.94
(WAGYU)
```

Routing decisions are fairly simple for most hosts:

1. If the destination host is on the local network, send the data directly to the destination host.
2. If the destination host is on a remote network that is reachable via a local gateway listed in the routing table, send it to the explicitly defined gateway.

3. If the destination host is on a remote network and there is no other entry that defines a route to that host, send the data to the default gateway.

These rules simply mean that if all else fails because there is no match, send the packet to the default gateway.

# iptraf-ng

In troubleshooting network connection problems, it can be helpful to use a tool such as **iptraf-ng** (IP traffic next generation) to monitor the network traffic on one or more interfaces. This is an easy tool to use, and it employs a text-mode menu style interface.

This is probably not a tool that anyone who has been a SysAdmin for a short time would be likely to use on a regular basis. In order to interpret the results in a meaningful way, it would be necessary to understand networking in far more depth than we cover here. However, I think that it is an interesting tool and knowing that it exists can be helpful. I found that just watching the traffic helped me learn about networking and it became easier to see when a host was not responding. This is a tool that takes some time to build experience with.

## EXPERIMENT 12-13

Perform this experiment as the root user. The **iptraf-ng** program will not run when launched by a non-root user. Install iptraf-ng if it is not already installed.

```
[root@studentvm1 ~]# dnf -y install iptraf-ng
```

In one terminal, ping the router. Do not limit the ping count as we want this to continue until we have completed this experiment. Notice that we are using the name we assigned to the router in /etc/hosts.

```
[root@studentvm1 ~]# ping router
PING router (10.0.2.1) 56(84) bytes of data.
64 bytes from router (10.0.2.1): icmp_seq=1 ttl=255 time=0.231 ms
64 bytes from router (10.0.2.1): icmp_seq=2 ttl=255 time=0.272 ms
64 bytes from router (10.0.2.1): icmp_seq=3 ttl=255 time=0.240 ms
64 bytes from router (10.0.2.1): icmp_seq=4 ttl=255 time=0.266 ms
64 bytes from router (10.0.2.1): icmp_seq=5 ttl=255 time=0.281 ms
<snip>
```

In another terminal session, start **iptraf-ng**.

```
[root@studentvm1 ~]# iptraf-ng
```

This command launches the menu-driven interface shown in Figure 12-6, which allows selection of various options.

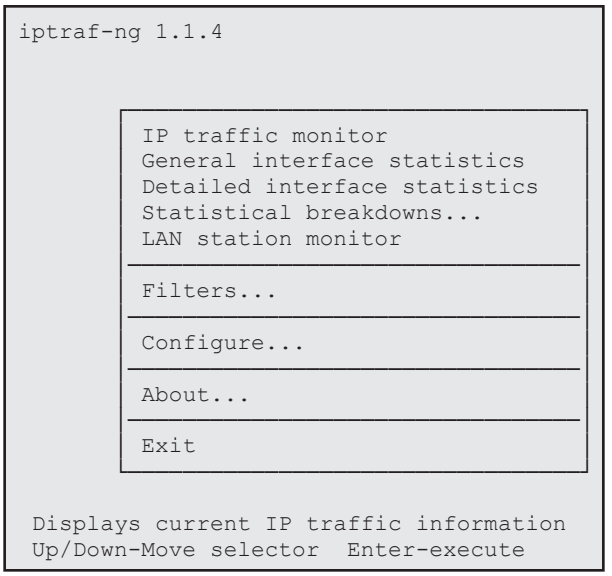


***Figure 12-6.*** *The main menu of* ***iptraf-ng***

Highlight the "IP traffic monitor" line and press the **Enter** key. Then select the **enp0s3** interface to monitor and press **Enter**.

After selecting the enp0s3 interface, the bottom portion of the resulting screen contains a continuous display of the ICMP packets and the responses from the router. Figure 12-7 also shows the total number of ICMP packets.

```
iptraf-ng 1.1.4
┌ TCP Connections (Source Host:Port) ───────────────── Packets ──── Bytes   Flag Iface ───┐
│                                                                                          │
│                                                                                          │
│                                                                                          │
│                                                                                          │
│                                                                                          │
│                                                                                          │
└ TCP:      0 entries ─────────────────────────────────────────────────────────── Active ──┘
┌──────────────────────────────────────────────────────────────────────────────────────────┐
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3
│ UDP (329 bytes) from 10.0.2.7:68 to 10.0.2.3:67 on enp0s3
│ UDP (576 bytes) from 10.0.2.3:67 to 10.0.2.7:68 on enp0s3
└ Bottom ──────── Elapsed time:   0:00 ──────────────────────────────────────────────────────┘
Packets captured:                          23 │ No TCP entries
Up/Dn/PgUp/PgDn-scroll  M-more TCP info   W-chg actv win  S-sort TCP  X-exit
```

***Figure 12-7.*** *The ICMP traffic resulting from the ping command*

But that really does not show us very much. Leave **`iptraf-ng`** and **`ping`** running as they are. On the desktop as the student user, open the browser and go to my web page at `www.both.org` or the example.com web site. Or, to be horrified, go to your local newspaper or TV station's home page and watch all of the ad and tracking connections pile up. While doing so, watch the iptraf-ng session. Figure 12-8 shows a sample of the results you might expect.

```
 iptraf-ng 1.1.4
┌ TCP Connections (Source Host:Port) ────────────── Packets ────── Bytes  Flag  Iface ─┐
│┌10.0.2.7:50370                                 >       5        231  CLOS  enp0s3  │
│└52.46.130.13:443                               >       4        184  CLOS  enp0s3  │
│┌10.0.2.7:39116                                 >      10       1543  CLOS  enp0s3  │
│└34.193.242.172:443                             >       8        796  CLOS  enp0s3  │
│┌10.0.2.7:49168                                 >       5        231  CLOS  enp0s3  │
│└23.13.210.16:443                               >       4        209  -PA-  enp0s3  │
│┌10.0.2.7:43974                                 >       5        231  CLOS  enp0s3  │
│└23.36.34.36:443                                >       4        209  -PA-  enp0s3  │
│┌10.0.2.7:55306                                 >       5        224  CLOS  enp0s3  │
│└23.36.32.49:443                                >       4        202  -PA-  enp0s3  │
│┌10.0.2.7:44002                                 >       5        231  CLOS  enp0s3  │
│└23.36.34.36:443                                >       4        209  -PA-  enp0s3  │
│┌10.0.2.7:43998                                 >       5        231  CLOS  enp0s3  │
│└23.36.34.36:443                                >       4        209  -PA-  enp0s3  │
│┌10.0.2.7:43996                                 >       5        231  CLOS  enp0s3  │
│└23.36.34.36:443                                >       4        209  -PA-  enp0s3  │
│┌10.0.2.7:43994                                 >       5        231  CLOS  enp0s3  │
│└23.36.34.36:443                                >       4        209  -PA-  enp0s3  │
│┌23.36.34.36:443                                =       1         46  RSET  enp0s3  │
│└10.0.2.7:43994                                 =       0          0  ----  enp0s3  │
│┌10.0.2.7:59892                                 >       6        363  CLOS  enp0s3  │
└ TCP:     68 entries ─────────────────────────────────────────────────────── Active ─┘
┌────────────────────────────────────────────────────────────────────────────────────┐
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3                 │
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3                  │
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3                 │
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3                  │
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3                 │
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3                  │
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3                 │
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3                  │
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3                 │
│ ICMP echo req (84 bytes) from 10.0.2.7 to 93.184.216.34 on enp0s3                  │
│ ICMP echo rply (84 bytes) from 93.184.216.34 to 10.0.2.7 on enp0s3                 │
└ Bottom ──────── Elapsed time:   0:00 ───────────────────────────────────────────────┘
 Packets captured:                          857  │ TCP flow rate:         0.00 kbps
 Up/Dn/PgUp/PgDn-scroll  M-more TCP info   W-chg actv win  S-sort TCP  X-exit
```

***Figure 12-8.*** *The results get more interesting when other activity is taking place, in this case, using the web browser*

Port 80 is the HTTP port used by web servers. In the first line of the output in the TCP Connections section of iptraf-ng, you can see the source address and port of the request from the host VM as 10.0.2.7:60868. The next line is the destination IP address and port number which is 23.45.181.162:80. So we are sending a request to port 80 (HTTP) at IP address 23.45.181.162.

You can see other conversations taking place with other servers as the embedded images and links are accessed by the browser.

The UDP entries in the bottom section are domain name services (DNS) requests looking for the IP addresses of other web servers to supply the linked content.

Press **x** twice to exit from iptraf-ng.

There are plenty of GUI tools available for performing these types of analysis, such as Wireshark, but iptraf-ng will work without a GUI while providing a lot of valuable information.

# Cleanup

Let's perform a bit of cleanup.

**CLEANUP**

Remove the immutable attribute bit from /etc/resolv.conf.

```
[root@studentvm1 etc]# cd /etc ; chattr -i resolv.conf ; lsattr resolv.conf
--------------e---- resolv.conf
```

# Chapter summary

In this chapter we have explored some of the basics of networking and been introduced to some common tools that enable exploration of networking on a Linux host.

We have explored basic network configuration of the client host in a DHCP environment and created an interface configuration file that allows us to turn a connection on and off. We have explored name services, CIDR notation, a bit about IPv6,

and routing. We have also used a simple tool, **sipcalc**, to explore CIDR notation and how to use it to determine the variable data for a network, such as the number of usable IP addresses, the network address, and more.

We have just scratched the surface and there is much more to learn. More of networking will be covered in the next course in this series, *Advanced Linux System and Server Administration.*

# Exercises

Perform the following exercises to complete this chapter:

1. Name the layers of the TCP/IP network model and list the protocols used in each.
2. Describe the flow of data from one network host to another using the TCP/IP model.
3. In Step 4 of Experiment 12-1, the results of the **arp** command are shown as IP addresses for those hosts connected to the network on NIC enp0s3 but as their respective hostnames on the network connected to NIC enp0s8. Why?
4. How many usable network addresses are available to the 10.125.16.32/31 network?
5. What is the function of name services?
6. Describe how you could use the /etc/hosts file to provide name services for a small network.
7. Describe the function of the default route.
8. Determine the route from your host to `www.apress.com`. Is there any packet loss?
9. Geographically, where is the server for `www.apress.com` located?

CHAPTER 13

# systemd

## Objectives

In this chapter you will learn

- To describe the functions of systemd
- To state the reasons for the controversy surrounding SystemV vs. systemd
- To list running services
- To list all active systemd units
- To use systemd to start, manage, and stop Linux services
- To manage filesystem mounts using systemd
- To create and activate a .mount unit file
- To use systemd timers to launch programs in a manner similar to using cron
- To select and display journal entries using journalctl

## Introduction

We have already explored systemd in the context of the Linux startup process in Chapter 16, and one might be tempted to think that we have covered the greater part of systemd and that there is no reason for a full chapter about it.

That would be incorrect. There is much left to explore.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_13

systemd[1,2] is the mother of all processes, and it is responsible for bringing the Linux host up to a state in which productive work can be done. Some of the functions assumed by systemd, which is far more extensive than the old init program, are to manage many aspects of a running Linux host, including mounting filesystems, managing hardware, and starting and managing system services required to have a productive Linux host. In this chapter we explore the functions of systemd that begin after startup is completed.

## Controversy

SystemV and systemd are two different methods of performing the Linux startup sequence. SystemV start scripts and the init program are the old method, and systemd using targets is the new method.

Just to ensure that we are all on the same page here, the Linux startup sequence begins after the kernel has loaded either init or systemd, depending upon whether the distribution uses the new or old startup, respectively. The init and systemd programs start and manage all of the other processes, that is, programs, and are both known as the mother of all processes on their respective systems.

Although most modern Linux distributions use the newer systemd for startup, shutdown, and process management, there are still some that do not. One reason for this is that some of the distribution maintainers and some SysAdmins prefer the older SystemV method over the newer systemd.

I think both have their advantages so let me explain my reasoning.

## Why I prefer SystemV

I prefer SystemV because it is more open. Startup is accomplished using bash scripts. After the kernel starts the init program, which is a compiled binary, init launches the rc.sysinit script which performs many system initialization tasks. After rc.sysinit has completed, init launches the /etc/rc.d/rc script which in turn starts the various services as defined by the SystemV start scripts in the /etc/rc.d/rcX.d, where "X" is the number of the runlevel that is being started.

[1]Wikipedia, systemd, https://en.wikipedia.org/wiki/Systemd

[2]Yes, systemd should always be spelled like this without any uppercase even at the beginning of a sentence. The documentation for systemd is very clear about this.

All of these programs are open and easily knowable scripts. It is possible to read through these scripts and learn exactly what is taking place during the entire startup process. Each script is numbered so that it starts the service for which it is intended in a specific sequence. Services are started serially and only one service is started at a time.

systemd is a complex system of large compiled binary executables that are not understandable without access to the source code. Although it's open source, so "access to the source code" isn't hard, just less convenient. It represents a significant refutation of multiple tenets of the Linux Philosophy. As a binary, systemd is not directly open to view or easy to change by the SysAdmin. systemd tries to do everything such as manage running services while providing significantly more status information than SystemV. It also manages hardware, processes and groups of processes, filesystem mounts, and much more. systemd is present in almost every aspect of the modern Linux host making it the one-stop tool for system management. All of this is a clear violation of the tenets that programs should be small and that each program should do one thing and do it well.

## Why I prefer systemd

I prefer systemd as my startup mechanism because it starts as many services as possible in parallel, depending upon the current stage in the startup process. This speeds the overall startup and gets the host system to a login screen faster than SystemV. This was discussed in Volume 1, Chapter 16.

systemd manages almost every aspect of a running Linux system. Like the obsolete SystemV tools, it can manage running services while providing significantly more status information than SystemV. It also manages hardware, processes and groups of processes, filesystem mounts, and much more. systemd is present in almost every aspect of the modern Linux host making it the one-stop tool for system management. Does this sound familiar?

systemd is open because all of the configuration files are ASCII text files. Startup configuration can be modified through various GUI and command-line tools, as well as adding or modifying various configuration files to suit the needs of the specific local computing environment.

## The real issue

Did you think I could not like both startup systems? I do and I can work with either one.

The real issue and the root cause of most of the controversy between SystemV and systemd is that there is no choice on the SysAdmin level.[3] The choice of whether to use SystemV or systemd has already been made by the developers, maintainers, and packagers of the various distributions. However, this is with good reason. Scooping out and replacing an init system, by its extreme, invasive nature, has a lot of consequences, and that would be hard to tackle outside of the distribution design process.

Despite the fact that this particular choice has been made for us, our Linux hosts boot up and work, which is what I usually care the most about. As an end user and even as a SysAdmin, my primary concern is whether I can get my work done – work such as writing this book, installing updates, and writing scripts to automate everything. So long as I can do my work, I don't really care about the start sequence used on my distro.

However, I do care when there is a problem during startup or service management. Regardless of which startup system is used on any host, I know enough and am able to follow the sequence of events to find the failure and fix it.

Also, despite the fact that most Linux developers agree that replacing the old SystemV startup is a good idea, many of them dislike systemd for that.[4]

## systemd suite

The fact is systemd is more than just a single program. It is a large suite of programs all designed to work together in order to manage nearly every aspect of a running Linux system. In Figure 13-1 we can see many of components belonging to systemd. This is a simplified diagram designed to provide a high-level overview so it does not include all of the individual programs or files. Nor does it provide any insight into data flow which is so complex as to be a useless exercise within the context of this course.

[3]OSnews, "*Editorial: Thoughts on Systemd [sic] and the Freedom to Choose*," www.osnews.com/story/28026/Editorial_Thoughts_on_Systemd_and_the_Freedom_to_Choose

[4]Steven J. Vaughan-Nichols, ZDNet, *Linus Torvalds and others on Linux's systemd*, www.zdnet.com/article/linus-torvalds-and-others-on-linuxs-systemd/

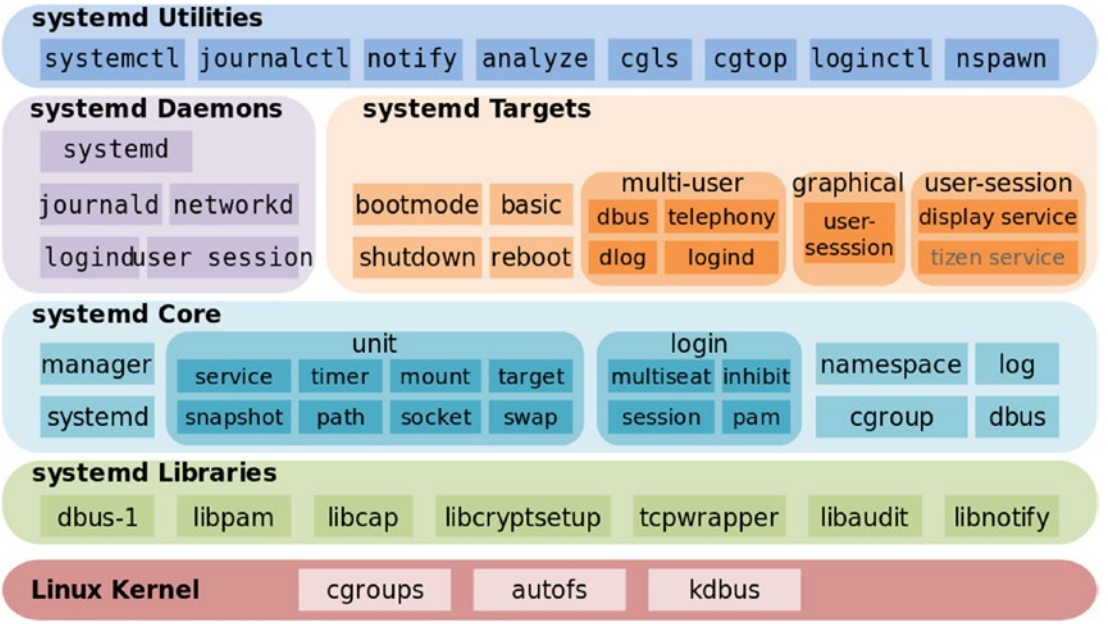


***Figure 13-1.*** *A simplified view of the structure of the systemd environment*
*"Architecture of systemd" by Shmuel Csaba Otto Traian is licensed under Creative Commons Attribution-Share Alike 3.0 Unported* https://creativecommons.org/licenses/by-sa/3.0/deed.en

A full exposition of systemd would take a book all on its own. We do not need to understand all of the details of how the systemd components shown in Figure 13-1 fit together, so our explorations will consist of those programs and components that enable us to manage various Linux services and to deal with log files and journals.

## Practical structure

The structure of systemd – outside of its executable files – is contained in its many configuration files. Although these files have different names and identifier extensions, they are all called "Unit" files. The basis of everything systemd is in units.

Unit files are ASCII plain text files that are accessible and which can be modified or created by the SysAdmin. There are a number of unit file types, each of which has its own man page. Figure 13-2 lists some of these unit file types along with a short description of each.

| systemd Unit | Description |
|---|---|
| **.automount** | The .automount units are used to implement on-demand, i.e., plug and play, mounting as well as mounting of filesystem units in parallel during startup. |
| **.device** | The .device unit files define hardware and virtual devices that are exposed to the SysAdmin in the /dev/directory. Not all devices have unit files, Typically block devices such as hard drives, and network devices have unit files, as well as some others. |
| **.mount** | The .mount unit defines a mount point on the Linux filesystem directory structure. |
| **.scope** | This unit defines and manages a set of system processes. This unit is not configured using unit files, but is created programmatically. Per the systemd.scope man page, "The main purpose of scope units is grouping worker processes of a system service for organization and for managing resources." |
| **.service** | .service unit files define processes that are managed by systemd. These include service such as crond cups (Common Unix Printing System), IPTables, multiple logical volume management (LVM) services, NetworkManager, and more. |
| **.slice** | The .slice unit defines a "slice," which is a conceptual division of system resources that are related to a group of processes. You can think of all system resources as a pie and this subset of resources as a "slice" out of that pie. |
| **.socket** | .socket units define inter-process communication sockets such as network sockets. |
| **.swap** | Defines swap devices or files. |
| **.target** | .target units define groups of unit files that define startup synchronization points, runlevels, and services. Target units define the services and other units that must be active in order to start successfully. |
| **.timer** | This unit defines timers that can initiate programs at specified times. |

***Figure 13-2.*** *A list of systemd unit file types*

# systemctl

Having already looked at its startup functions in Chapter 16 of Volume 1, we start further exploration of systemd with its service management functions. systemd provides the **`systemctl`** command that is used to start and stop services, configure them to launch – or not – at system startup, and to monitor the current status of running services.

All of the experiments in this chapter should be performed as the root user unless otherwise specified. Some of the commands that simply list various systemd units can be performed by non-root users, but those that make changes cannot.

## EXPERIMENT 13-1

In a terminal session as root user, ensure that root's home directory (~) is the PWD. Let's start by just looking at units in various ways.

List all of the loaded and active systemd units. **`systemctl`** automatically pipes its STDOUT data stream through the **`less`** pager so we do not need to do so.

```
[root@studentvm1 ~]# systemctl
UNIT                                          LOAD   ACTIVE SUB       DESCRIPTION
proc-sys-fs-binfmt_misc.automount             loaded active running   Arbitrary Executable File>
sys-devices-pci0000:00-0000:00:01.1-ata7-host6-target6:0:0-6:0:0:0-block-sr0.device loaded a>
sys-devices-pci0000:00-0000:00:03.0-net-enp0s3.device loaded active plugged   82540EM Gigabi>
sys-devices-pci0000:00-0000:00:05.0-sound-card0.device loaded active plugged   82801AA AC'97>
sys-devices-pci0000:00-0000:00:08.0-net-enp0s8.device loaded active plugged   82540EM Gigabi>
sys-devices-pci0000:00-0000:00:0d.0-ata1-host0-target0:0:0-0:0:0:0-block-sda-sda1.device loa>
sys-devices-pci0000:00-0000:00:0d.0-ata1-host0-target0:0:0-0:0:0:0-block-sda-sda2.device loa>
<snip - removed lots of lines of data from here>

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

206 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

As you scroll through the data in your terminal session, look for the following specific things.

The first section lists devices such as hard drives, sound cards, network interface cards, and TTY devices. Another section shows the filesystem mount points. Other sections include various services and a list of all loaded and active targets.

The sysstat timers, along with other timers, at the very bottom of the output, are used to collect and generate daily system activity summaries for SAR, the System Activity Reporter. We covered SAR in Volume 1, Chapter 13, as a very useful problem-solving tool.

Near the very bottom, three lines describe the meanings of the status, loaded, active, and sub. Press **q** to exit from the pager.

Use the following command, as suggested by the last line of the preceding output, to see all units that are installed, whether they are loaded or not. I won't reproduce more than a very few lines here, because you can scroll through them on your own. The systemctl program has an excellent tab completion facility that makes it easy to enter complex commands without the need to memorize each of the many options.

```
[root@studentvm1 ~]# systemctl list-unit-files
UNIT FILE                                  STATE
proc-sys-fs-binfmt_misc.automount          static
-.mount                                    generated
boot.mount                                 generated
dev-hugepages.mount                        static
dev-mqueue.mount                           static
home.mount                                 generated
```

You will see that some units are disabled. Table 1 in the man page for **systemctl** lists and provides short descriptions of the entries you might see in this listing. Let's use the -t (type) option to view only the timer units.

```
[root@studentvm1 ~]# systemctl list-unit-files -t timer
```

Or we could do the same thing like this.

```
[root@studentvm1 ~]# systemctl list-timers
```

Or list the mount points. There is no option to do **systemctl list-mounts**, but we can list the mount point unit files.

```
[root@studentvm1 ~]# systemctl list-unit-files -t mount
```

Now let's look at the service units. This command will show all services installed on the host whether they are active or not.

```
[root@studentvm1 ~]# systemctl --all -t service
```

At the bottom of this listing of service units, it displays 166 as the total number of loaded units on my host. Your number will very probably be different from that.

Unit files do not have a file name extension such as .unit to help identify them. Rather we can generalize that most configuration files belonging to systemd are unit files of one type or another. The few remaining files are mostly .conf files located in /etc/systemd.

Unit files are stored in the /usr/lib/systemd directory and its subdirectories, while the /etc/systemd/ directory and its subdirectories contain symbolic links to the unit files necessary to the local configuration of this host.

Make /etc/systemd the PWD and list its contents. Then make /etc/systemd/system the PWD and list its contents. Also list the contents of at least a couple of the subdirectories of the current PWD.

Let's explore one-unit file. The default.target file determines which runlevel target the system will boot to. Chapter 16 discussed how to change the default target from the GUI (graphical.target) to command-line-only (multi-user) target. The default.target file is simply a symlink to /usr/lib/systemd/system/graphical.target.

Take a few minutes to examine the contents of the /etc/systemd/system/default.target file.

```
[root@studentvm1 system]# cat default.target
#  SPDX-License-Identifier: LGPL-2.1+
#
#  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
```

Notice that it requires the multi-user.target. The graphical.target cannot be started if the multi-user.target is not already up and running. It also says it "wants" the display-manager.service unit.

A "want" does not need to be fulfilled in order for the unit to be started successfully. If the "want" cannot be fulfilled, it will be ignored by systemd and the rest of the target will be started regardless.

The subdirectories in /etc/systemd/system are lists of wants for various targets. Take a few minutes to explore the files and their contents in the /etc/systemd/system/graphical.target.wants directory.

---

The systemd.unit man page contains much good information about unit files, their structure, the sections into which they can be divided, and lists of the options that can be used. It also lists many of the unit types, all of which have man pages of their own. If you want to interpret a unit file, this would be one good place to start.

We will continue our exploration of the systemctl as well as other related commands throughout the rest of this chapter.

## Service units

Now that we know a bit about unit files and targets, let's explore some other units.

Much of the time a Fedora installation installs and enables services that are not needed for normal operation for a particular host or, conversely, other services may need to be installed, enabled, and started. Services that are not needed for the Linux host to function as desired but which are installed and possibly running represent a security risk and should be stopped and disabled at the least and at best uninstalled.

The **`systemctl`** command is used to manage systemd units including services, targets, mounts, and more.

## EXPERIMENT 13-2

Perform this experiment as the root user.

In Experiment 13-1 we looked at a list of services. Let's take a closer look because we can easily find services that will never be used.

```
[root@studentvm1 ~]# systemctl --all -t service
UNIT                                LOAD      ACTIVE SUB        DESCRIPTION
<snip>
chronyd.service                     loaded    active running    NTP client/server
crond.service                       loaded    active running    Command Scheduler
cups.service                        loaded    active running    CUPS Scheduler
dbus-daemon.service                 loaded    active running    D-Bus System Message Bus
<snip>
● ip6tables.service                 not-found inactive dead     ip6tables.service
● ipset.service                     not-found inactive dead     ipset.service
● iptables.service                  not-found inactive dead     iptables.service
<snip>
firewalld.service                   loaded    active   running  firewalld - dynamic
                                                                 firewall daemon
<snip>
● ntpd.service                      not-found inactive dead     ntpd.service
● ntpdate.service                   not-found inactive dead     ntpdate.service
pcscd.service                       loaded    active   running  PC/SC Smart Card Daemon
```

I have pruned out all but a few lines of output from the command to save space. The services that show as "loaded active running" are obvious.

The "not-found" services are ones of which systemd is aware but which are not installed on the Linux host. If you want to run those services, the packages that contain them must be installed first.

Notice the pcscd.service unit. This is the PC/SC smart card daemon. Its function is to communicate with smart card readers. Many Linux hosts – including our virtual machines – have no need for such a reader or the service that is loaded and taking up memory and CPU resources. We can stop this service and disable it so it will not start again on the next boot. First we will simply check its status.

```
[root@studentvm1 ~]# systemctl status pcscd.service
● pcscd.service - PC/SC Smart Card Daemon
   Loaded: loaded (/usr/lib/systemd/system/pcscd.service; indirect; vendor
           preset: disabled)
   Active: active (running) since Fri 2019-05-10 11:28:42 EDT; 3 days ago
     Docs: man:pcscd(8)
 Main PID: 24706 (pcscd)
    Tasks: 6 (limit: 4694)
   Memory: 1.6M
   CGroup: /system.slice/pcscd.service
           └─24706 /usr/sbin/pcscd --foreground --auto-exit

May 10 11:28:42 studentvm1 systemd[1]: Started PC/SC Smart Card Daemon.
```

This data is what I mean about the additional information provided by systemd; SystemV would tell us only that the service was running or not. Note that specifying the .service unit type is optional. Now stop and disable the service. Then re-check its status.

```
[root@studentvm1 ~]# systemctl stop pcscd ; systemctl disable pcscd
Warning: Stopping pcscd.service, but it can still be activated by:
  pcscd.socket
Removed /etc/systemd/system/sockets.target.wants/pcscd.socket.
[root@studentvm1 ~]# systemctl status pcscd
● pcscd.service - PC/SC Smart Card Daemon
   Loaded: loaded (/usr/lib/systemd/system/pcscd.service; indirect; vendor
           preset: disabled)
   Active: failed (Result: exit-code) since Mon 2019-05-13 15:23:15 EDT; 48s ago
     Docs: man:pcscd(8)
 Main PID: 24706 (code=exited, status=1/FAILURE)

May 10 11:28:42 studentvm1 systemd[1]: Started PC/SC Smart Card Daemon.
May 13 15:23:15 studentvm1 systemd[1]: Stopping PC/SC Smart Card Daemon...
May 13 15:23:15 studentvm1 systemd[1]: pcscd.service: Main process exited,
                                       code=exited, status=1/FAIL>
May 13 15:23:15 studentvm1 systemd[1]: pcscd.service: Failed with result
                                       'exit-code'.
May 13 15:23:15 studentvm1 systemd[1]: Stopped PC/SC Smart Card Daemon.
```

The short log entry display for most services prevents us having to search through various log files to locate this type of information.

Check the status of the system runlevel targets. Notice that specifying the "target" unit type is required for this.

```
[root@studentvm1 ~]# systemctl status multi-user.target
● multi-user.target - Multi-User System
   Loaded: loaded (/usr/lib/systemd/system/multi-user.target; static; vendor
           preset: disabled)
   Active: active since Thu 2019-05-09 13:27:22 EDT; 4 days ago
     Docs: man:systemd.special(7)

May 09 13:27:22 studentvm1 systemd[1]: Reached target Multi-User System.
[root@studentvm1 ~]# systemctl status graphical.target
● graphical.target - Graphical Interface
   Loaded: loaded (/usr/lib/systemd/system/graphical.target; indirect; vendor
           preset: disabled)
   Active: active since Thu 2019-05-09 13:27:22 EDT; 4 days ago
     Docs: man:systemd.special(7)

May 09 13:27:22 studentvm1 systemd[1]: Reached target Graphical Interface.
[root@studentvm1 ~]# systemctl status default.target
● graphical.target - Graphical Interface
   Loaded: loaded (/usr/lib/systemd/system/graphical.target; indirect; vendor
           preset: disabled)
   Active: active since Thu 2019-05-09 13:27:22 EDT; 4 days ago
     Docs: man:systemd.special(7)

May 09 13:27:22 studentvm1 systemd[1]: Reached target Graphical Interface.
```

Note that the default target *is* the graphical target. The status of any unit can be checked in this way.

## Mount units

A mount unit is one that defines all of the parameters required to mount a filesystem on a designated mount point.

systemd can manage mount units with more flexibility than those using the /etc/fstab filesystem configuration file. Despite this, systemd still uses the /etc/fstab file for filesystem configuration and mounting purposes. systemd uses the **systemd-fstab-generator** tool to create transient mount units from the data in the fstab file.

## EXPERIMENT 13-3

Perform this experiment as root. In Experiment 13-1 we listed, among other units, the mount units. The TestFS.mount unit was one of those. So let's explore mounts in much more detail. Let's start with our TestFS filesystem and mount.

```
[root@studentvm1 ~]# systemctl status TestFS.mount
● TestFS.mount - /TestFS
   Loaded: loaded (/etc/fstab; generated)
   Active: active (mounted) since Tue 2019-05-14 09:18:51 EDT; 2h 17min ago
    Where: /TestFS
     What: /dev/sdb1
     Docs: man:fstab(5)
           man:systemd-fstab-generator(8)
    Tasks: 0 (limit: 4694)
   Memory: 68.0K
   CGroup: /system.slice/TestFS.mount

May 13 21:51:00 studentvm1 systemd[1]: Mounting /TestFS...
May 13 21:51:00 studentvm1 systemd[1]: Mounted /TestFS.
```

This shows that the unit file for this device was generated and that the device is mounted, which may not be the case on your VM, so mount it if it is not. We could use the **mount** and **umount** commands to unmount this unit, but let's use **systemctl** instead. First let's stop (umount) TestFS.mount and check its status.

```
[root@studentvm1 ~]# systemctl stop TestFS.mount ; systemctl status TestFS.mount
● TestFS.mount - /TestFS
   Loaded: loaded (/etc/fstab; generated)
   Active: inactive (dead) since Tue 2019-05-14 11:40:57 EDT; 13ms ago
    Where: /TestFS
     What: /dev/sdb1
     Docs: man:fstab(5)
           man:systemd-fstab-generator(8)

May 13 21:59:44 studentvm1 systemd[1]: Mounting /TestFS...
May 13 21:59:44 studentvm1 systemd[1]: Mounted /TestFS.
May 13 22:00:00 studentvm1 systemd[1]: Unmounting /TestFS...
May 13 22:00:00 studentvm1 systemd[1]: Unmounted /TestFS.
```

And restart (mount) it.

```
[root@studentvm1 ~]# systemctl start TestFS.mount ; systemctl status TestFS.
mount
● TestFS.mount - /TestFS
   Loaded: loaded (/etc/fstab; generated)
   Active: active (mounted) since Tue 2019-05-14 11:42:52 EDT; 12ms ago
    Where: /TestFS
     What: /dev/sdb1
     Docs: man:fstab(5)
           man:systemd-fstab-generator(8)
    Tasks: 0 (limit: 4694)
   Memory: 96.0K
   CGroup: /system.slice/TestFS.mount

May 13 22:00:31 studentvm1 systemd[1]: Mounting /TestFS...
May 13 22:00:31 studentvm1 systemd[1]: Mounted /TestFS.
```

We can also use the **systemd-mount** and **systemd-umount** commands. The advantage of using these commands is that **systemd-mount**, for example, can take into account any prerequisite mount units and mount them as required.

Suppose that we have two mounts, mount1 and mount2, and that mount1 contains a mount point, mount2, on which the mount2 unit will be mounted. Using the traditional mount command, we would first need to mount the mount1 unit on its mount point, /mount1, and then we could mount the mount2 unit on /mount1/mount2. The **systemd-mount** command automatically does that for us when we mount the mount2 unit.

Mount units may be configured either with the traditional /etc/fstab file or with systemd units. Fedora uses the fstab file as it is created during the installation. However, systemd uses the **systemd-fstab-generator** program to translate the fstab file into systemd units for each entry in the fstab file.

Now that we know that we can use systemd .mount unit files for filesystem mounting, let's try that out. We will create a mount unit for this filesystem.

First, unmount /TestFS. Edit the /etc/fstab file and comment out the TestFS line. Now create a new file with the name TestFS.mount in the /etc/systemd/system directory. Edit it to contain the following configuration data. It is required that the unit file name and the name of the mount point be identical or the mount will fail.

The Description line in the [Unit] section is for us humans, and it provides the name we see when we list mount units with **systemctl -t mount**. The data in the [Mount] section of this file contains essentially the same data that would be found in the fstab file.

```
# This mount unit is for the TestFS filesystem created in the course
# "Using and Administering Linux"
# By David Both
# Licensed under GPL V2
#

[Unit]
Description=TestFS Mount

[Mount]
What=/dev/sdb1
Where=/TestFS
Type=ext4
Options=defaults

[Install]
WantedBy=multi-user.target
```

Now enable and then start the mount unit.

```
[root@studentvm1 etc]# systemctl enable TestFS.mount
Created symlink /etc/systemd/system/multi-user.target.wants/TestFS.mount → /
etc/systemd/system/TestFS.mount.
[root@studentvm1 ~]# systemctl start TestFS.mount
```

Verify that the filesystem has been mounted.

```
[root@studentvm1 ~]# systemctl status TestFS.mount
● TestFS.mount - TestFS Mount
   Loaded: loaded (/etc/systemd/system/TestFS.mount; enabled; vendor preset:
           disabled)
   Active: active (mounted) since Tue 2019-05-14 11:58:07 EDT; 18s ago
    Where: /TestFS
     What: /dev/sdb1
    Tasks: 0 (limit: 4694)
```

```
    Memory: 88.0K
    CGroup: /system.slice/TestFS.mount

May 14 09:18:51 studentvm1 systemd[1]: Mounting TestFS Mount...
May 14 09:18:51 studentvm1 systemd[1]: Mounted TestFS Mount.
```

This experiment has been specifically about creating a unit file for a mount, but it can be applied to other types of unit files as well. The details will be different but the concepts are the same.

## systemd timers

Having already explored the use of **cron** and **at** to run commands and scripts in Chapter 11, we now turn to systemd timers to accomplish a similar function. systemd timers are similar to cron jobs and provide no equivalent to the **at** command for one-time execution.

systemd timers have additional capabilities that are not present with cron which only triggers on specific real-time dates and times. systemd timers can be configured to trigger based on the status changes of other systemd units. For example, a timer might be configured to trigger a specific elapsed time after system boot, after startup,[5] or after activation of a defined service unit. These are called monotonic[6] timers. Figure 13-3 lists the monotonic timers along with a short definition of each, as well as the OnCalendar timer which is used to specify particular times in the future. This information is taken from the systemd.timer man page.

[5]Remember that boot and startup are different things. In this case startup refers to the startup of systemd itself.

[6]A count or sequence that continually increases

| Timer | Monotonic | Definition |
|---|---|---|
| OnActiveSec= | X | Defines a timer relative to the moment the timer itself is activated. |
| OnBootSec= | X | Defines a timer relative to when the machine was booted up. |
| OnStartupSec= | X | Defines a timer relative to when systemd was first started. |
| OnUnitActiveSec= | X | Defines a timer relative to when the unit the timer is activating was last activated. |
| OnUnitInactiveSec= | X | Defines a timer relative to when the unit the timer is activating was last deactivated. |
| OnCalendar= | | Defines realtime (i.e. wallclock) timers with calendar event expressions. See systemd.time(7) for more information on the syntax of calendar event expressions. Otherwise, the semantics are similar to OnActiveSec= and related settings. |

***Figure 13-3.*** *A list of systemd timer definitions*

Configuring systemd timers is a bit different than using crontab. Each timer requires a timer file which refers to a service unit file that defines the script or commands.

## Time specification

systemd itself and its timers use a different style for time and date specifications than the format used in crontab. It is more flexible than crontab and allows fuzzy dates and times in the manner of the **at** command. It should also be familiar enough that it will be easy to understand.

The basic format for systemd timers is DOW YYYY-MM-DD HH:MM:SS. The DOW (day of week) is optional and other fields can use an asterisk (*) to match any value for that position. All of the various calendar time forms are converted to a normalized form for use. If the time is not specified, it is assumed to be 00:00:00. If the date is not specified but the time is, the next match may be today or tomorrow.

systemd provides us with an excellent tool for validating and examining calendar time events that are to be used in a specification. The **systemd-analyze calendar** tool will parse a calendar time events specification and provide the normalized form as well as other interesting information such as the date and time of the next "elapse," that is, match, and the approximate amount of time before the trigger time is reached.

The systemd.time man page has a complete explanation of these time specifications.

## EXPERIMENT 13-4

This experiment should be performed as the student user. In this experiment we will explore the use of the **systemd-analyze calendar** tool.

**Tip** If the current date on which you are performing this experiment is past the date used in these experiments, add 2 years to the current year to ensure that these dates are in the future.

First, let's look at a date in the future without a time.

**Tip** The times for "Next elapse" and "UTC" will differ based on your local time zone.

```
[student@studentvm1 ~]$ systemd-analyze calendar 2030-06-17
  Original form: 2030-06-17
Normalized form: 2030-06-17 00:00:00
    Next elapse: Mon 2030-06-17 00:00:00 EDT
       (in UTC): Mon 2030-06-17 04:00:00 UTC
       From now: 11 years 1 months left
```

Now let's add a time. Note that the date and time are analyzed separately as non-related entities.

```
[student@studentvm1 ~]$ systemd-analyze calendar 2030-06-17 15:21:16
  Original form: 2030-06-17
Normalized form: 2030-06-17 00:00:00
    Next elapse: Mon 2030-06-17 00:00:00 EDT
       (in UTC): Mon 2030-06-17 04:00:00 UTC
       From now: 11 years 1 months left

  Original form: 15:21:16
Normalized form: *-*-* 15:21:16
    Next elapse: Fri 2019-05-17 15:21:16 EDT
       (in UTC): Fri 2019-05-17 19:21:16 UTC
       From now: 23h left
```

To analyze the date and time as a single unit, enclose them together in quotes.

```
[student@studentvm1 ~]$ systemd-analyze calendar "2030-06-17 15:21:16"
Normalized form: 2030-06-17 15:21:16
    Next elapse: Mon 2030-06-17 15:21:16 EDT
       (in UTC): Mon 2030-06-17 19:21:16 UTC
       From now: 11 years 1 months left
```

Now just specify a time earlier than the current time and one later. In this case the current time was 16:16 on 2019-05-15.

```
[student@studentvm1 ~]$ systemd-analyze calendar 15:21:16 22:15
  Original form: 15:21:16
Normalized form: *-*-* 15:21:16
    Next elapse: Fri 2019-05-17 15:21:16 EDT
       (in UTC): Fri 2019-05-17 19:21:16 UTC
       From now: 23h left

  Original form: 22:15
Normalized form: *-*-* 22:15:00
    Next elapse: Thu 2019-05-16 22:15:00 EDT
       (in UTC): Fri 2019-05-17 02:15:00 UTC
       From now: 5h 59min left
```

The **systemd-analyze calendar** tool – despite the systemd.time(7) man page to the contrary – does not work with all timestamps or calendar time specifications. Any specification that does not result in a distinct time will not work. So things like "tomorrow" or "today" cause errors.[7]

Despite its limitations, the **systemd-analyze calendar** tool can still help you understand the structure of the calendar time specifications used by systemd timers. I strongly recommend reading the systemd.time(7) man page for a more complete understanding of the time formats that can be used with systemd timers.

[7]I reported this limitation on May 16, 2019, as bug number 1711065 on the Red Hat Bugzilla web site. The response was that a new tool would be added to systemd-analyze. Within 24 hours, new code was submitted via GitHub and the man page was rewritten a bit to clarify the difference between timestamps and time specifications. All of the discussion and a link to the GitHub page for this bug is at https://bugzilla.redhat.com/show_bug.cgi?id=1711065. This is an excellent example of how those of us who are not developers can contribute to the betterment of open source software. It is not clear how long testing will take and when the revised code will appear in updates to released versions of Fedora.

## Timer configuration

Let's configure a timer that will perform the same task as that in Experiment 11-3 and log the results of the **free** command to the file, freemem.log.

**EXPERIMENT 13-5**

This experiment must be performed as the root user.

First let's create the shell program that will be executed by the service. Create the shell program /usr/local/bin/freemem.sh and make it executable. Add the following content to freemem.sh.

```
#!/usr/bin/bash

# This timer unit is for testing timer units in the course
# "Using and Administering Linux"
# By David Both
# Licensed under GPL V2
#

/usr/bin/date >> /tmp/freemem.log
/usr/bin/free >> /tmp/freemem.log
```

Now create the service unit that will run this shell program. In the /etc/systemd/system/ directory, create the file freemem.service. It does not need to be executable. Add the following content to the freemem.service file.

```
# This service unit is for testing timer units in the course
# "Using and Administering Linux"
# By David Both
# Licensed under GPL V2
#

[Unit]
Description=Logs free memory to /tmp/freemem.log
Wants=freemem.timer

[Service]
ExecStart=/usr/local/bin/freemem.sh
```

```
[Install]
WantedBy=multi-user.target
```

Finally create the freemem.timer unit file in /etc/systemd/system and add the following content. This file does not need to be executable.

```
# This timer unit is for testing timer units in the course
# "Using and Administering Linux"
# By David Both
# Licensed under GPL V2
#

[Unit]
Description=Logs free memory to /tmp/freemem.log
Requires=freemem.service

[Timer]
Unit=freemem.service
OnCalendar=*-*-* *:*:00

[Install]
WantedBy=timers.target
```

The OnCalendar time specification in the freemem.timer file *-*-* *:*:00 should trigger the timer to execute the freemem.service unit every minute.

Now let's activate the timer.

```
[root@studentvm1 system]# systemctl start freemem.timer
[root@studentvm1 system]# systemctl -t timer
UNIT                         LOAD   ACTIVE SUB     DESCRIPTION
dnf-makecache.timer          loaded active waiting dnf makecache --timer
freemem.timer                loaded active waiting Logs free memory to /tmp/
                             freemem.log
mlocate-updatedb.timer       loaded active waiting Updates mlocate database
                             every day
sysstat-collect.timer        loaded active waiting Run system activity
                             accounting tool every 10 minutes
sysstat-summary.timer        loaded active waiting Generate summary of
                             yesterday's process accounting
```

```
systemd-tmpfiles-clean.timer loaded active waiting Daily Cleanup of Temporary
Directories
unbound-anchor.timer         loaded active waiting daily update of the root
                             trust anchor for DNSSEC
[root@studentvm1 system]# systemctl status freemem.timer
● freemem.timer - Logs free memory to /tmp/freemem.log
   Loaded: loaded (/etc/systemd/system/freemem.timer; enabled; vendor preset:
   disabled)
   Active: active (waiting) since Fri 2019-05-17 09:51:21 EDT; 10min ago
  Trigger: Fri 2019-05-17 10:02:00 EDT; 29s left

May 17 09:51:21 studentvm1 systemd[1]: Started Logs free memory to /tmp/
freemem.log.
```

Check that the timer is updating the log file.

```
[root@studentvm1 tmp]# tail -f freemem.log
Mem:       4036976     493316     1346668        8800     2196992     3287784
Swap:      6291452          0     6291452
Fri May 17 09:52:56 EDT 2019
             total       used        free      shared  buff/cache   available
Mem:       4036976     492256     1347700        8800     2197020     3288844
Swap:      6291452          0     6291452
Fri May 17 09:53:19 EDT 2019
             total       used        free      shared  buff/cache   available
Mem:       4036976     492084     1347692        8800     2197200     3288948
Swap:      6291452          0     6291452
```

What do you notice about the results? *Hint:* look at the timestamps. They are not consistent every minute on the minute. This illustrates that the default accuracy is 1 minute. That means that any given timer may elapse +/- 1 minute off from its ideal configured time. That is fine for many tasks but not for any that require very fine accuracy.

This flexible accuracy is intentional, and it is designed to optimize power consumption to suppress unnecessary CPU wake-ups.

There is an optional variable that I discovered in the systemd.timer(5) man page. We can add AccuracySec to the timer unit file, which allows us to define the desired accuracy down to 1 microsecond (1μs).

Of course, there is also a variable that allows setting of the maximum random delay time, RandomizedDelaySec. Its default setting is 0.

Together, these two variables prevent multiple timers from elapsing at the exact same moment. This randomization is meant to ensure that the tasks launched by timers do not interfere with each other.

Let's set the AccuracySec variable to .01 seconds to improve accuracy. Add the following line to the end of the [Timer] section of the freemem.timer unit file.

```
AccuracySec=.01s
```

We do not need to restart the timer for this to take effect, although we could do that. It is easier to use the following command which forces systemd to reload all of its unit files and to rerun all of its generators for devices that do not have explicit unit files.

```
[root@studentvm1 system]# systemctl daemon-reload
```

May 17 09:51:21 studentvm1 systemd[1]: Started Logs free memory to /tmp/freemem.log.

You should still be using tail to follow the changes to the freemem.log file so the changes will look like this.

```
Fri May 17 12:49:36 EDT 2019
              total        used        free      shared  buff/cache    available
Mem:        4036976      494380     1341448        8800     2201148     3286668
Swap:       6291452           0     6291452
Fri May 17 12:50:56 EDT 2019
              total        used        free      shared  buff/cache   available
Mem:        4036976      495272     1340116        9000     2201588     3285584
Swap:       6291452           0     6291452
Fri May 17 12:51:00 EDT 2019
              total        used        free      shared  buff/cache   available
Mem:        4036976      494928     1340468        8800     2201580     3286080
Swap:       6291452           0     6291452
Fri May 17 12:52:00 EDT 2019
              total        used        free      shared  buff/cache   available
Mem:        4036976      494016     1341312        8800     2201648     3286996
Swap:       6291452           0     6291452
```

This is exactly what we want.

If we want the timer to run after a reboot, we can enable it.

```
[root@studentvm1 system]# systemctl enable freemem.timer
Created symlink /etc/systemd/system/timers.target.wants/freemem.timer →
/etc/systemd/system/freemem.timer.
```

Close the editor sessions and reboot the StudentVm1 host. After logging back in, verify that the timer is working. To clean up, stop and disable the timer.

```
[root@studentvm1 tmp]# systemctl stop freemem.timer
[root@studentvm1 tmp]# systemctl disable freemem.timer
Removed /etc/systemd/system/timers.target.wants/freemem.timer.
```

# systemd-analyze

The systemd-analyze tool also allows us to look at which processes take the most time during startup as well as output of system data to various types of graphics files that can be used by other viewing tools for visual analysis.

**EXPERIMENT 13-6**

This experiment can be performed as the student user. Let's start with a simple look at startup to see which services take the most time to initialize.

```
[root@studentvm1 ~]# systemd-analyze blame
         13.514s firewalld.service
         13.004s udisks2.service
          7.437s sssd.service
          6.741s abrtd.service
          6.000s plymouth-quit-wait.service
          5.595s polkit.service
          5.410s ModemManager.service
          4.555s NetworkManager-wait-online.service
          2.932s accounts-daemon.service
          2.598s initrd-switch-root.service
          2.337s lvm2-monitor.service
          2.199s gssproxy.service
          2.183s avahi-daemon.service
          2.137s sysstat.service
<snip>
```

This could be helpful if the startup takes a long time. If you check this out before problems arise, you will have something to compare when you are seeking the cause of a problem. Any service that has a significantly longer startup time than normal might have a configuration or other problem that is causing the delay. Note that service startup times will vary from one startup to another.

A more complex look at startup can be achieved by creating graphics files that can be viewed on the desktop. These files display timelines of when each service began its startup sequence and finished it. These graphics can be created in formats suitable for viewing in a web browser or one of the graphics programs like LibreOffice Draw or a PDF viewer like Okular. In this example we will create an svg vector graphics file and then use the Firefox browser to view it.

```
Ensure that the student home directory (~) is the PWD.
```

```
[student@studentvm1 ~]$ systemd-analyze plot > system.svg
```

Click the "Home" icon on the desktop to open the Thunar file manager. Locate the file we just created, system.svg, and right-click it. Click through the menu items **Open with ➤ Open with other application** to get to the **Open With** window. Scroll down as necessary and select Firefox. Click the **Open** button or just double-click Firefox.

Scroll to the lower left side of the image to view the legend which will make interpreting the graph easier. You can then see that the dark red portions of each bar are the startup times for the services.

This is a large image so take some time to scroll around it. You can also use the key combinations Ctrl+ and Ctrl- to enlarge or reduce the size of the image on the screen to get an overall view.

This view of the system startup also shows all of the services that start making it easier to find some background processes that may not be necessary. For example, I always stop and disable the avahi daemon.

# Journals

Log files are a SysAdmin's best friend, and Fedora has a /var/log directory full of log files and subdirectories with even more log files. All of the log files are composed of ASCII plain text and can be perused with standard text tools.

systemd keeps its "logs" in files called journals with an extension of ".journal" for ease of identification. These journals are stored in a binary format in the /var/log/journal directory and are accessed and processed into readable format by the **journalctl** utility.

The systemd journals contain entries from services controlled by systemd.

## EXPERIMENT 13-7

Perform this experiment as the root user. Although non-root users have some access to the data in these journals, some, such as kernel entries, are only accessible by root.

Start by listing all journal entries. Notice that the pager, **less**, is automatically invoked. Entries are listed from the oldest at the top to the newest at the bottom.

```
[root@studentvm1 ~]# journalctl
-- Logs begin at Sat 2019-03-30 05:34:48 EDT, end at Wed 2019-05-15 14:05:37 EDT. --
Mar 30 05:34:48 studentvm1 audit[1]: SERVICE_START pid=1 uid=0 auid=4294967295
ses=4294967295 msg='un>
Mar 30 05:34:48 studentvm1 dbus-daemon[914]: [system] Successfully activated service
'org.freedesktop>
Mar 30 05:34:48 studentvm1 systemd[1]: Started Network Manager Script Dispatcher
Service.
Mar 30 05:34:48 studentvm1 nm-dispatcher[25225]: req:1 'dhcp4-change' [enp0s8]: new
request (3 script>
Mar 30 05:34:48 studentvm1 nm-dispatcher[25225]: req:1 'dhcp4-change' [enp0s8]:
start running ordered>
Mar 30 05:34:48 studentvm1 rsyslogd[902]: imjournal: sd_journal_get_cursor() failed:
Cannot assign re>
Mar 30 05:34:48 studentvm1 dhclient[1320]: bound to 192.168.0.181 -- renewal in 257
seconds.
Mar 30 05:34:49 studentvm1 rsyslogd[902]: imjournal: journal reloaded... [v8.39.0
try http://www.rsys>
Mar 30 05:34:58 studentvm1 audit[1]: SERVICE_STOP pid=1 uid=0 auid=4294967295
ses=4294967295 msg='uni>
Mar 30 05:35:07 studentvm1 dhclient[1302]: DHCPREQUEST on enp0s3 to 10.0.2.11 port
67 (xid=0x33207958)
Mar 30 05:35:07 studentvm1 dhclient[1302]: DHCPACK from 10.0.2.11 (xid=0x33207958)
Mar 30 05:35:07 studentvm1 NetworkManager[1040]: <info>  [1553938507.6748] dhcp4
(enp0s3):   address >
```

```
Mar 30 05:35:07 studentvm1 NetworkManager[1040]: <info>  [1553938507.6748] dhcp4
(enp0s3):   plen 24 >
Mar 30 05:35:07 studentvm1 NetworkManager[1040]: <info>  [1553938507.6749] dhcp4
(enp0s3):   gateway >
Mar 30 05:35:07 studentvm1 NetworkManager[1040]: <info>  [1553938507.6749] dhcp4
(enp0s3):   lease ti
<snip>
```

The journal files are rotated on a regular basis and older ones are deleted. This prevents any single file from getting too large and also prevents these journal files from filling up all of the space on the /var filesystem.

Press **G** (uppercase) to go to the end of the list. It may take a few seconds for all of the files to load in order to get to the end.

```
May 15 14:05:26 studentvm1 nm-dispatcher[24400]: req:2 'dhcp4-change'
[enp0s8]: start running ordered>
May 15 14:05:26 studentvm1 dhclient[1207]: bound to 192.168.0.181 -- renewal
in 270 seconds.
May 15 14:05:37 studentvm1 audit[1]: SERVICE_STOP pid=1 uid=0 auid=4294967295
ses=4294967295 msg='uni>
lines 777688-777726/777726 (END)
```

The last line is highlighted and shows the total number of lines in the journal listing. That amounts to 777,726 lines in this journal on my VM. That is a lot of data.

To make it easier to locate specific data, we can use the -t option to specify a syslog identifier such as a service name. Look at the entries for the kernel.

```
[root@studentvm1 ~]# journalctl -t kernel
```

Search for "Reboot" in this data to locate places where reboots occurred. The data stored here looks very much like the data found in the output data stream from the **dmesg** command. We could also do this.

```
[root@studentvm1 ~]# journalctl -k
```

The -k option shows all of the kernel entries but does not explicitly mark the reboots.

We can also specify a message priority using the -pX where X ranges from 0 (Emergency) to 6 (info) and 7 (debug). Check the **journalctl** man page for more information on these priorities.

```
[root@studentvm1 ~]# journalctl -p2
```

Try other priority levels as well.

Do you want to see a list of every time that the host was booted?

```
[root@studentvm1 ~]# journalctl --list-boots
```

Or all entries between two specified times? The -S option is "since" a specified time and -U is "until" a specified time. Omitting a date implies "today."

```
[root@studentvm1 ~]# journalctl -S 01:00:00 -U 01:10:05
```

Or

```
[root@studentvm1 ~]# journalctl -S "2019-05-10 01:00:00" -U "2019-05-10 01:10:05"
```

The date and time specification must be in double quotes due to the space used to separate the date from the time. Omitting the time specification implies a time of 00:00:00.

Options can be combined. In this next example, we select all of the journal entries for the DHCP client (dhclient) between 01:00:00 and 01:10:05 on May 10, 2019.

```
[root@studentvm1 ~]# journalctl -S "2019-05-10 01:00:00" -U "2019-05-10
01:10:05" -t dhclient
```

If you get no results for this, expand the time differential by making the "Until" option a later time.

# Chapter summary

The systemd suite of tools is very complex and quite powerful. We have learned how to use systemctl to explore the unit structure and how to manage services. We have explored mount units in some detail and created a custom unit file with which we can manage the new mount unit. We have also learned to use the **journalctl** program to view systemd journals.

We have seen that systemd can provide more status information than the old SystemV tools. Despite the controversy still surrounding systemd, it is a powerful and very flexible tool for managing many aspects of a running Linux host.

There is much more to systemd than we have been able to explore in this course, including both Chapter 16 of Volume 1 and this one. You should now have enough basic knowledge of systemd, its functions, configuration structure, and tools to be able to perform further exploration on your own.

# References

There is a great deal of information about systemd available on the Internet, but much of it can be terse, obtuse, or even misleading. In addition to the resources referred to in the various footnotes throughout this chapter, I have compiled here a list of web pages with more detailed and reliable information about systemd startup:

- There is a good practical guide at systemd – Fedora Project. This site has pretty much everything you need to know in order to configure, manage, and maintain a Fedora computer using systemd.

  https://docs.fedoraproject.org/en-US/quick-docs/understanding-and-administering-systemd/index.html

- For detailed technical information about systemd and the reasons for creating it, check out this "Description of systemd" at Freedesktop.org.

  www.freedesktop.org/wiki/Software/systemd

- A basic introduction to systemd at Linux.com.

  www.linux.com/learn/tutorials/524577-here-we-go-again-another-linux-init-intro-to-systemd

- Managing services on systemd at Linux.com.

  www.linux.com/learn/tutorials/527639-managing-services-on-linux-with-systemd

- More advanced systemd information and tips at Linux.com.

  www.linux.com/learn/tutorials/539856-more-systemd-fun-the-blame-game-and-stopping-services-with-prejudice

There is also a series of articles written by Lennart Poettering, the designer and primary developer behind systemd. These are deeply technical articles that are meant for Linux system administrators. These articles were written between April 2010 and September 2011, but they are just as relevant now as they were then. Much of everything else good that has been written about systemd and its ecosystem is based on these papers:

- Rethinking PID1

  http://0pointer.de/blog/projects/systemd.html

- systemd for Administrators, Part 1

  http://0pointer.de/blog/projects/systemd-for-admins-1.html

- systemd for Administrators, Part II

  http://0pointer.de/blog/projects/systemd-for-admins-2.html

- systemd for Administrators, Part III

  http://0pointer.de/blog/projects/systemd-for-admins-3.html

- systemd for Administrators, Part IV

  http://0pointer.de/blog/projects/systemd-for-admins-4.html

- systemd for Administrators, Part V

  http://0pointer.de/blog/projects/three-levels-of-off.html

- systemd for Administrators, Part VI

  http://0pointer.de/blog/projects/changing-roots

- systemd for Administrators, Part VII

  http://0pointer.de/blog/projects/blame-game.html

- systemd for Administrators, Part VIII

  http://0pointer.de/blog/projects/the-new-configuration-files.html

- systemd for Administrators, Part IX

  http://0pointer.de/blog/projects/on-etc-sysinit.html

- systemd for Administrators, Part X

  http://0pointer.de/blog/projects/instances.html

- systemd for Administrators, Part XI

  http://0pointer.de/blog/projects/inetd.html

# Exercises

Complete the following exercises to finish this chapter:

1. Define a unit file.
2. How many loaded units are there on your VM host?
3. How many active units are there on your VM host?
4. Does the TestFS.mount unit file mount the TestFS filesystem after a reboot?
5. What unit files are required to implement a timer?
6. Use systemd-analyze to determine the total amount of time it took for startup at the last boot.
7. What does avahi do?
8. How long did avahi take to start?
9. Terminate avahi and disable it.
10. Are there any emergency level entries in the systemd journal?

CHAPTER 14

# D-Bus and udev

## Objectives

In this chapter you will learn

- How Linux treats all devices as plug and play
- What D-Bus and udev are
- How D-Bus and udev work together to make device access easy
- How to write rules for udev

## /dev chaos

The /dev directory has always been the location for the device files in all Unix and Linux operating systems. Note that device files are not the same as device drivers. Each device file represents one actual or potential physical device connected to the host.

In the past, device files were created at the time the operating system was created. This meant that all possible devices that might ever be used on a system needed to be created in advance. In fact, tens of thousands of device files needed to be created to handle all of the possibilities. It became very difficult to determine which device file actually related to a specific physical device.

The development of two very important tools, D-Bus and udev, has provided Linux with the ability to create device files only when they are needed by a device that is already installed or one that is hot-plugged into the running system.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_14

# About D-Bus

D-Bus[1] is a Linux software interface used for inter-process communications (IPC). It was first released in 2006. We looked at one form of IPC in Volume 1, Chapter 13, the named pipe, in which one program would push data into the named pipe and another program would extract the data.

D-Bus is a system-wide and more complex form of IPC that allows many kernel- and system-level processes to send messages to the logical message bus. Other processes listen to the messages on the bus and may choose to react to those messages or not.

# About udev

The udev[2] daemon is designed to simplify the chaos that had overtaken the /dev directory with huge numbers of mostly unneeded devices. At startup, udev creates entries in /dev only for those devices that actually currently exist or which have a high probability of actually existing on the host. This significantly reduces the number of device files required.

In addition to detecting devices, udev assigns names to those devices when they are plugged into the system, such as USB storage and printers, and other non-USB types of devices as well. In fact, udev treats all devices as plug and play, even at boot time. This makes dealing with devices consistent at all times. udev also moves device naming out of kernel space and into user space.

Greg Kroah-Hartman, one of the authors of udev, wrote an interesting and informative article[3] for *Linux Journal*. It provides insight into the details of udev and how it is supposed to work. That article discusses udev, a program that replaces and improves upon the functionality of the old devfs. It provides /dev entries for devices in the system at any moment in time. It also provides features previously unavailable through devfs alone, such as persistent naming for devices when they move around the device tree, a flexible device naming scheme, notification of external systems of device changes, and moving all naming policy out of the kernel.

[1]Wikipedia, *D-Bus*, https://en.wikipedia.org/wiki/D-Bus

[2]Wikipedia, *udev*, https://en.wikipedia.org/wiki/Udev

[3]Greg Kroah-Hartman, Kernel Korner – udev — Persistent Naming in User Space, *Linux Journal*, June, 2004, www.linuxjournal.com/article/7316

Note that udev has matured since the article was written and some things have changed, such as the udev rule location and structure, but the overall objectives and architecture are the same.

One of the main consequences of using udev for persistent plug'n'play naming is that it makes things much easier for the average non-technical user. This is a good thing in the long run; however, there have been migration problems.

## EXPERIMENT 14-1

Perform this experiment as the student user on the GUI desktop. This experiment assumes that the USB stick is formatted and has one partition.

Open the Thunar file manager and ensure that the side panel is visible. It does not matter whether it is in Shortcuts or Tree mode because the storage devices are visible in both.

Plug a USB thumb drive that is known to be working into the physical host. Then, on the VirtualBox window for StudentVM1, use the menu bar to open Devices ➤ USB and, while watching the Thunar window, place a check mark next to the USB device you just plugged in. The device will now be available to the VM, and it will be shown in the Thunar side panel.

Verify that the new device special file has been created in /dev/.

```
[root@studentvm1 ~]# ll /dev | grep sd
brw-rw----  1 root    disk      8,   0 May 17 11:35 sda
brw-rw----  1 root    disk      8,   1 May 17 11:35 sda1
brw-rw----  1 root    disk      8,   2 May 17 11:35 sda2
brw-rw----  1 root    disk      8,  16 May 17 11:35 sdb
brw-rw----  1 root    disk      8,  17 May 17 11:35 sdb1
brw-rw----  1 root    disk      8,  18 May 17 11:35 sdb2
brw-rw----  1 root    disk      8,  32 May 17 11:35 sdc
brw-rw----  1 root    disk      8,  48 May 17 11:35 sdd
brw-rw----  1 root    disk      8,  64 May 20 08:29 sde
brw-rw----  1 root    disk      8,  65 May 20 08:29 sde1
```

You should see that the new devices, /dev/sde and /dev/sde1, have been created within the last couple minutes or so. One device special file was created for the entire device, /dev/sde, and one was created for the partition on the device, /dev/sde1. The device name may be different on your VM depending upon how closely you have been performing the experiments in the preceding chapters of this course.

The D-Bus and udev services work together to make this happen.

Here is a simplified version of what takes place when a new device is connected to the host. I stipulate here that the host system is already booted and running at multi-user.target (runlevel 3) or graphical.target (runlevel 5):

1. The user plugs in a new device, usually into an external USB, SATA, or eSATA connector.
2. The kernel detects this and sends a message on D-Bus to announce the new device.
3. udev reads the message proffered on D-Bus.
4. Based on the device properties and its location in the hardware bus tree, udev creates a name for the new device if one does not already exist.
5. The udev system creates the device special file in /dev.
6. If a new device driver is required, it is loaded.
7. The device is initialized.
8. udev may send a notification to the desktop so that the desktop may display an icon for the new device to the user.

The process of hot-plugging a new hardware device into a running Linux system and making it ready is very complex – for the operating system. It is very simple for the user who just wants to plug in a new device and have it work. This simplifies things immensely for the end user. For USB and SATA hard drives, USB thumb drives, keyboards, mice, printers, displays, and nearly anything else, all that a user needs to do is to plug the device into the appropriate USB or SATA port and it will work.

# Naming rules

udev stores its default naming rules in files in the /usr/lib/udev/rules.d directory, and it's local configuration files in the /etc/udev/rules.d directory. Each file contains a set of rules for a specific device type. These rules should not be changed.

In earlier versions of udev, there were many local rule sets created, including a set for NIC naming. As each NIC was discovered by the kernel and renamed by udev for the very first time, a rule was added to the rule set for the network device type. This was initially done to ensure consistency before names had changed from "ethX" to more consistent ones.

Now that udev has multiple consistent default rules for determining device names, especially for NICs, storing the specific rules for each device in local configuration files is not required to maintain that consistency.

## EXPERIMENT 14-2

Perform this experiment as the student user. We can look at the startup log for our VM and see where the NIC names were changed. You may also encounter some other messages that match the search pattern.

```
[student@studentvm1 ~]$ dmesg | grep -i eth
[    5.014594] e1000 0000:00:03.0 eth0: (PCI:33MHz:32-bit) 08:00:27:e1:0c:10
[    5.014643] e1000 0000:00:03.0 eth0: Intel(R) PRO/1000 Network Connection
[    5.592516] e1000 0000:00:08.0 eth1: (PCI:33MHz:32-bit) 08:00:27:f6:b0:68
[    5.592559] e1000 0000:00:08.0 eth1: Intel(R) PRO/1000 Network Connection
[    5.595377] e1000 0000:00:03.0 enp0s3: renamed from eth0
[    5.614190] e1000 0000:00:08.0 enp0s8: renamed from eth1
```

This result is from my VM which has two NICs so there are lines for both. You should have only a single NIC so only one set of entries in the **dmesg** data stream.

The first line for eth0 shows when the NIC was "discovered," at 5.014594 seconds after startup. The second line shows the ID of the device as an (virtual) Intel network device. The last line for eth0 records the renaming from eth0 to enp0s3.

# Making udev work

This section first appeared as an article[4] by Seth Kenlon at Opensource.com. It was published there under a CC-by-SA 4[5] license and is used here with permission of the author. I thought about writing this section myself, but this is such a well-written piece and it covers everything I wanted to do myself, so I decided to use it here. The only changes I have made are to remove the use of sudo and to better incorporate it into the format of this book.

This is an excellent example of authors who share their work using a Creative Commons license I did not legally need to ask permission because the license does not require that, but I felt that was the appropriate thing to do.

§§

## Using Udev for your success

Udev is the subsystem in Linux that supplies your computer with device events. In plain English, that means it's the code that detects when you have things plugged into your computer, like a network card, external hard drives (including USB thumb drives), mice, keyboards, joysticks and gamepads, DVD-ROM drives, and so on. That makes it a potentially useful utility, and it's well enough exposed to a standard user such that you can manually script it to, for instance, perform certain tasks when a certain hard drive is plugged in.

This article teaches you how to create a udev script triggered by some udev event, such as plugging in a specific thumb drive. Once you understand the process for working with udev, you can use it to do all manner of things, like loading a specific driver when a gamepad is attached or performing an automatic backup when your backup drive is attached.

### A basic script

The best way to work with udev is in small chunks. Don't write the entire script up front, but instead start with something that simply confirms that udev does indeed trigger some custom event.

[4]Kenlon, Seth, *An introduction to Udev: The Linux subsystem for managing device events*, https://opensource.com/article/18/11/udev

[5]http://creativecommons.org/licenses/by-sa/4.0/

Depending on the ultimate goal of your script, you won't be able to guarantee that you will ever see the results of a script with your own eyes, so make your script log that it was successfully triggered. The usual place for log files is in the /var directory, but that's mostly the root user's domain, so for testing, use /tmp, which is accessible by normal users and also usually gets cleaned out every so often.

Open your favorite text editor as root and enter this simple script.

```
#!/usr/bin/bash
/usr/bin/date >> /tmp/udev.log
```

Place this in /usr/local/bin or some such place in the default executable path. Call it trigger.sh and, of course, make it executable with chmod +x.

```
[root@studentvm1 bin]# chmod +x /usr/local/bin/trigger.sh
```

This script has nothing to do with udev. When this script is executed, this script places a timestamp in the file /tmp/udev.log. Test the script yourself.

```
[root@studentvm1 ~]# trigger.sh ; cat /tmp/udev.log
Mon May 20 17:03:25 EDT 2019
```

The next step is to make udev, rather than yourself, trigger the script.

## Unique device identification

In order for your script to be triggered by a device event, udev must know under what conditions it should call the script. In real life, you can identify a thumb drive by its color, the manufacturer, and the fact that you just plugged it into your computer. Your computer, however, obviously needs a different set of criteria.

Udev identifies devices by serial numbers, manufacturers, and even vendor ID and product ID numbers. Since this is early in the life span of your udev script, be as broad, non-specific, and all-inclusive as possible. In other words, you want first to catch nearly any valid udev event to trigger your script.

With the **udevadm** monitor command, you can tap into udev in real time and see what it sees when you plug in different devices. Try it as root.

```
[root@studentvm1 ~]# udevadm monitor
```

The monitor function prints received events for

- UDEV: The event which udev sends out after rule processing
- KERNEL: The kernel uevent

With **udevadm monitor** running, plug in a thumb drive and watch as all kinds of information are spewed out onto your screen. Notice, particularly, that the type of event is an ADD event. That's a good way of identifying what type of event you want.

The **udevadm monitor** command provides a lot of good info, but you can see it with prettier formatting with the command **udevadm info**, assuming you know where your thumb drive is currently located in your /dev tree. If not, unplug and then plug your thumb drive back in and then immediately issue this command.

```
[root@studentvm1 ~]# dmesg | tail | grep -i sd*
[265211.509658] scsi host7: usb-storage 1-1:1.0
[265212.528687] scsi 7:0:0:0: Direct-Access     JetFlash TS512MJF150
8.07 PQ: 0 ANSI: 2
[265212.529157] sd 7:0:0:0: Attached scsi generic sg5 type 0
[265212.550431] sd 7:0:0:0: [sde] 1003520 512-byte logical blocks: (514
MB/490 MiB)
[265212.556999] sd 7:0:0:0: [sde] Write Protect is off
[265212.557006] sd 7:0:0:0: [sde] Mode Sense: 03 00 00 00
[265212.563576] sd 7:0:0:0: [sde] No Caching mode page found
[265212.563582] sd 7:0:0:0: [sde] Assuming drive cache: write through
[265212.610157]  sde: sde1
[265212.647708] sd 7:0:0:0: [sde] Attached SCSI removable disk
```

Assuming that command returned sde: sde1, for instance, then your thumb drive is being assigned the sde label by the kernel. Alternately, you can use the **lsblk** command to see all drives, including sizes and partitions, attached to your system. Now that you have established where your drive is currently located in your filesystem, you can view udev information about that device.

```
[root@studentvm1 ~]# udevadm info -a -n /dev/sde | less
```

This returns a lot of information. Focus on the first block of info for now. Your job is to pick out parts of udev's report about a device that are most unique to that device and then tell udev to trigger your script when those unique attributes are detected.

What's happening on a technical level is that the **udevadm info** process reports on a device (specified by the device path) and then "walks" up the chain of parent devices. For every device found, it prints all possible attributes using a key-value format. You can compose a rule to match according to the attributes of a device plus attributes from one single parent device.

```
looking at device '/devices/pci0000:00/0000:00:0b.0/usb1/1-1/1-1:1.0/
host7/target7:0:0/7:0:0:0/block/sde':
  KERNEL=="sde"
  SUBSYSTEM=="block"
  DRIVER==""
  ATTR{alignment_offset}=="0"
  ATTR{capability}=="51"
  ATTR{discard_alignment}=="0"
  ATTR{events}=="media_change"
  ATTR{events_async}==""
  ATTR{events_poll_msecs}=="-1"
  ATTR{ext_range}=="256"
  ATTR{hidden}=="0"
  ATTR{inflight}==""       0        0"
  ATTR{range}=="16"
  ATTR{removable}=="1"
  ATTR{ro}=="0"
  ATTR{size}=="1003520"
  ATTR{stat}==""     109     0     4184     1434     0     0     0
                       0     0      116     1369     0     0     0     0
```

A udev rule must contain one attribute from one single parent device.

Parent attributes are things that describe a device from the most basic level, such as it's something that has been plugged into a physical port or it is something with a size or this is a removable device. Since the KERNEL label of sde can change depending upon how many other drives you happen to have plugged in before you plug that thumb drive in, that's not the optimal parent attribute for a udev rule. However, it works for a proof of concept, so you could use it. An even better candidate is the SUBSYSTEM attribute, which identifies that this is a "block" system device (which is why the lsblk command lists the device).

Create a new file called 80-local.rules in /etc/udev/rules.d and enter this code.

```
SUBSYSTEM=="block", ACTION=="add", RUN+="/usr/local/bin/trigger.sh"
```

Save the file, unplug your test thumb drive, and then reboot.

Wait, reboot on a Linux machine?

Theoretically, you can just issue **udevadm control --reload**, which should load all rules, but at this stage in the game, it's best to eliminate all variables. Udev is complex enough without wondering if that rule didn't work because of a syntax error or if you just should have rebooted. So reboot regardless of what your POSIX pride tells you.

---

**Tip** Although rebooting your StudentVM1 host at this juncture is still a very good idea, I did try using the **udevadm control --reload** command and the trigger.sh script did trigger as expected, leaving a new entry in /tmp/udev.log.

---

When your system is back online, switch to a text console (with Ctrl-Alt-F3 or similar) and plug your thumb drive in. If you are running a recent kernel, you will probably see a bunch of output in your console when you plug the drive in. If you see an error message such as "Could not execute /usr/local/bin/trigger.sh," then you probably forgot to make the script executable. Otherwise, hopefully all you see is that a device was plugged in and that it got some kind of kernel device assignment and so on. Now, the moment of truth.

```
[root@studentvm1 rules.d]# cat /tmp/udev.log
Mon May 20 17:03:25 EDT 2019
Tue May 21 07:58:30 EDT 2019
```

If you see a very recent date and time returned from /tmp/udev.log, then the udev has successfully triggered your script.

## Refining the rule into something useful

The problem with the rule right now is that it's very generic. Plugging in a mouse, a thumb drive, or someone else's thumb drive will all indiscriminately trigger your script. Now is the time to start focusing in on the exact thumb drive you want to trigger your script.

One way to do this is with the vendor ID and product ID. To get these numbers, you can use the lsusb command.

```
[root@studentvm1 rules.d]# lsusb
Bus 001 Device 006: ID 058f:6387 Alcor Micro Corp. Flash 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
```

In this example, the 058f:6387 before "Alcor Micro Corp. Flash Drive" denotes the idVendor and idProduct attributes. You can now include these attributes in your rule. Be sure to use the IDs for your device and not the ones for my device.

```
SUBSYSTEM=="block", ATTRS{idVendor}=="058f", ACTION=="add", RUN+="/usr/
local/bin/trigger.sh"
```

Test this (yes, you should still reboot, just to make sure you're getting fresh reactions from udev), and it should work the same as before, only now if you plug in, say, a thumb drive manufactured by a different company (therefore with a different idVendor) or a mouse or a printer, the script is not triggered.

Keep adding in new attributes to further focus in on that one unique thumb drive that you actually want to have trigger your script. Using **`udevadm info -a -n /dev/sde`**, you can find out things like the vendor name, or sometimes a serial number, or the product name, and so on.

For your own sanity, be sure to add only one new attribute at a time. Most mistakes I have made and have seen other people make are to throw a bunch of attributes into their udev rule and wonder why the thing no longer works. Testing attributes one by one is the safest way to ensure udev can identify your device successfully.

## Security

This brings up the security concerns of writing udev rules to automatically do something when a drive is plugged in.

On my machines, I don't even have automount turned on, and yet this article proposed scripts and rules that execute commands just by having something plugged in.

Two things to bear in mind here.

Focus your udev rules once you have them working so that they only trigger scripts when you really want them to. Executing a script that blindly copies data to or from your computer is a bad idea if anyone who happens to be carrying a thumb drive of the same

brand as yours comes along and plugs it into your box. Do not write your udev rule and scripts and then forget about them. I know which computers have my udev rules on them, and those boxes are much more my personal computers than the ones that I take around to conferences or have in my office at work. The more "social" a computer is, the less likely it is to get a udev rule on it that could potentially result in my data ending up on someone else's device or someone else's data or malware on my device.

In other words, as with so much of the power that a GNU system provides you, it is your job to be mindful of how you are wielding that power. If you abuse it or fail to treat it with respect, then it very well could go horribly wrong.

## Udev in the real world

Now that you can confirm that our script is triggered by udev, you can turn your attention to the function of the script. Right now, it is useless, doing nothing more than logging the fact that it has been executed. I use udev to trigger automated backups of my thumb drives. The idea is that the master copies of my active documents are on my thumb drive (since it goes everywhere I go and could be worked on at any moment), and those master documents get backed up to my computer each time I plug the drive into that machine. In other words, my computer is the backup drive and my production data is mobile.

Since that's what I use udev for the most, it's the example I'll use here, but udev can grab lots of other things, like gamepads (this is useful on systems that aren't set to load the xboxdrv module when a gamepad is attached) and cameras and microphones (useful to set inputs when a specific mic is attached), so don't think that this one example is all it's good for.

A simple version of my backup system is a two-command process.

```
SUBSYSTEM=="block", ATTRS{idVendor}=="03f0", ACTION=="add",
SYMLINK+="safety%n"
SUBSYSTEM=="block", ATTRS{idVendor}=="03f0", ACTION=="add", RUN+="/usr/
local/bin/trigger.sh"
```

The first line detects my thumb drive with the attributes already discussed and then assigns the thumb drive a symlink within the device tree. The symlink it assigns is safety%n. The %n is a udev macro that resolves to whatever number the kernel gives to the device, such as sdb1, sdb2, sdb3, and so on. So %n would be the 1 or the 2 or the 3.

This creates only a symlink in the dev tree, so it does not interfere with the normal process of plugging in a device. This means that if you do use a desktop environment that likes to automount devices, you won't be causing problems for it. The second line runs the script.

My backup script looks like this.

```
#!/usr/bin/bash
 mount /dev/safety1 /mnt/hd
sleep 2
rsync -az /mnt/hd/ /home/seth/backups/ && umount /dev/safety1
```

The script uses the symlink, which avoids the possibility of udev naming the drive something unexpected (for instance, if I have a thumb drive called DISK plugged into my computer already, and I plug in my other thumb drive also called DISK, the second one will be labelled DISK_, which would foil my script). It mounts safety1 (the first partition of the drive) at my preferred mount point of /mnt/hd.

Once safely mounted, it uses rsync to back up the drive to my backup folder (my actual script uses rdiff-backup, and yours can use whatever automated backup solution you prefer).

### Udev is your Dev

Udev is a very flexible system and enables you to define rules and functions in ways that few other systems dare provide users. Learn it and use it, and enjoy the power of POSIX.

§§

## Chapter summary

In this chapter we have explored how D-Bus and udev work together to enable a very powerful and flexible plug and play feature for Linux. We have looked at how udev works to provide names for newly plugged-in devices and how it created device special files in the /dev directory.

We have also created custom udev rules of our own that are used to trigger events of various types. This capability enables us to exercise control over what happens when a device is plugged in to our Linux hosts to a degree that is impossible in most other PC operating systems.

This chapter once again merely provides a very brief experience with D-Bus and udev. There is much more that can be done using udev rules, but you should now at least be aware of some of the possibilities.

# Exercises

Perform the following exercises to complete this chapter:

1. Describe the relationship between D-Bus and udev when a new hardware device is plugged into a Linux host.
2. List the steps taken by D-Bus and udev from when a USB thumb drive is plugged into a host until the device icon appears on the desktop.
3. Given that the udev action when a device like the USB drive used in this chapter is unplugged from the host is "removed". Write a udev rule that adds a timestamp to the /tmp/udev.log file when the USB thumb drive is removed. Test to verify that this new rule works as expected.

CHAPTER 15

# Logs and Journals

## Objectives

In this chapter you will learn

- To use the System Activity Report (SAR) to monitor system performance and potential problems using historical performance data
- To use log files to monitor system performance and events
- To configure and use logrotate to manage log files
- To use systemd journals to monitor system events and performance
- To configure and use Logwatch to provide a daily summary of log and journal entries for use in problem determination

## Logs are your friend

Use the log files to help determine the source of problems and performance issues. They contain large amounts of data that can be used to track down many types of problems. The most common error I make when troubleshooting is not going to the log files sooner.

Almost all of the log files are located in /var/log and can be accessed either directly or with simple commands. The most current of each type of log file has no date as part of its name, while older log file names have dates to differentiate them. In general and by default, the log files are maintained for a period of 1 month with each log file containing a maximum of 1 week of data. If the amount of data in a file passes a pre-configured threshold, the file may be rotated when it reaches that threshold rather than waiting for the full 7-day time period to pass.

The logrotate facility manages log rotation and deletion.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_15

# SAR

My long-time favorite tool for problem determination is System Activity Report, or SAR. SAR is an excellent place to start looking for information about a Linux computer's performance and performance-related problems.

SAR has a daemon that runs in the background collecting data. Every 10 minutes, the collected data is stored in the /var/log/sa directory. These logs are in a binary format and cannot be read directly. The `sar` command is used to view these records.

One of the advantages of SAR is the fact that it reports historical data for up to 30 days. This enables us to go back in time and see if we can locate patterns or specific periods when the load on one or more resources was very high. None of the other performance monitoring tools available from the Fedora repositories provide this type of historical data. Commands like **`top`**, **`iostat`**, **`vmstat`**, and so on all provide only instant-in-time views of the data they monitor.

---

**Note** SAR is not installed or enabled on some distributions. Recent releases of Fedora do install and enable SAR.

---

**PREPARATION FOR EXPERIMENT 15-1**

Perform this preparation section as root to install SAR if it is not already installed. First, test to see if SAR is installed and enabled.

```
[root@studentvm1 log]# sar
Linux 5.0.7-200.fc29.x86_64 (studentvm1)        05/21/2019      _x86_64_        (2 CPU)

12:00:01 AM     CPU     %user     %nice   %system   %iowait    %steal     %idle
12:10:04 AM     all      0.10      0.00      1.41      1.88      0.00     96.61
12:20:17 AM     all      0.08      0.00      1.36      1.35      0.00     97.21
12:30:17 AM     all      0.08      0.00      1.36      1.16      0.00     97.40
<snip>
```

If the **`sar`** command results in a data stream similar to this, skip the rest of this preparation; otherwise, continue with the rest of this preparation.

The package we need to install is sysstat.

```
[root@studentvm1 ~]# dnf -y install sysstat
```

Enable the sysstat service and start it. This also enables the sysstat timers.

```
[root@studentvm1 log]# systemctl enable sysstat
Created symlink /etc/systemd/system/multi-user.target.wants/sysstat.service
→ /usr/lib/systemd/system/sysstat.service.
Created symlink /etc/systemd/system/sysstat.service.wants/sysstat-collect.
timer → /usr/lib/systemd/system/sysstat-collect.timer.
Created symlink /etc/systemd/system/sysstat.service.wants/sysstat-summary.
timer → /usr/lib/systemd/system/sysstat-summary.timer.
[root@studentvm1 log]# systemctl start sysstat
```

SAR is now installed and the system data collection processes have been started.

If you just installed the sysstat package, there will not be any data collected until after the next 10-minute time increment, like on the hour, 10 after, 20 after, and so on. If you had to install the sysstat package, I suggest you wait for an hour or so to allow some data to accumulate. You can check the contents of the /var/log/sa directory to verify that data is being collected. You could also check the messages file to look for entries pertaining to sysstat.

In current versions of Fedora, the data aggregation is managed by systemd and the several control files are located in the /usr/lib/systemd/system directory. SAR data collection is performed every 10 minutes, and the daily aggregation is done once per day. These data collections used to be triggered by cron jobs but are now triggered by systemd timers.

## EXPERIMENT 15-1

This experiment can be performed as the student user. In its simplest form, the **sar** command displays CPU statistics in 10-minute summary increments since midnight or the last boot.

```
[student@studentvm1 ~]$ sar | head -25
Linux 5.0.7-200.fc29.x86_64 (studentvm1)  05/21/2019 _x86_64_  (2 CPU)

12:00:01 AM     CPU     %user     %nice   %system   %iowait    %steal     %idle
12:10:04 AM     all      0.10      0.00      1.41      1.88      0.00     96.61
12:20:17 AM     all      0.08      0.00      1.36      1.35      0.00     97.21
12:30:17 AM     all      0.08      0.00      1.36      1.16      0.00     97.40
12:40:11 AM     all      0.07      0.02      1.34      1.09      0.00     97.49
12:50:17 AM     all      0.07      0.00      1.31      1.51      0.00     97.11
01:00:17 AM     all      0.08      0.00      1.33      1.29      0.00     97.30
```

```
01:10:17 AM     all      0.08      0.00      1.36      1.73      0.00     96.83
01:20:17 AM     all      0.10      0.00      1.43      1.29      0.00     97.18
01:30:17 AM     all      0.20      0.00      1.43      1.57      0.00     96.80
01:40:14 AM     all      0.07      0.03      1.36      1.52      0.00     97.03
01:50:00 AM     all      0.07      0.00      1.31      1.41      0.00     97.22
02:00:17 AM     all      0.07      0.00      1.39      1.35      0.00     97.20
02:10:07 AM     all      0.08      0.00      1.38      1.79      0.00     96.74
02:20:17 AM     all      0.09      0.00      1.40      1.30      0.00     97.21
02:30:17 AM     all      0.07      0.00      1.34      1.28      0.00     97.31
02:40:17 AM     all      0.10      0.31      1.61      2.06      0.00     95.92
02:50:14 AM     all      0.07      0.00      1.32      1.44      0.00     97.18
03:00:17 AM     all      0.10      0.00      1.43      1.52      0.00     96.95
03:10:17 AM     all      0.12      0.00      1.48      1.46      0.00     96.94
03:20:03 AM     all      0.08      0.00      1.38      2.13      0.00     96.41
03:30:09 AM     all      0.07      0.00      1.34      1.57      0.00     97.02
03:40:00 AM     all      0.07      0.01      1.36      1.19      0.00     97.38
```

I have used the **head** utility to truncate the output after 25 lines for this experiment. Each line in the output displays the averages of all the data collected during each 10-minute period. So for the period ending at 03:10:17, the idle time for the CPU was 96.4%.

Now run the **sar** command using the -A option to display all of the data types collected by SAR. Run it through the **less** utility so you can page through the data which is far too long for me to reproduce here. The -A option also displays more information for some sections, including the CPU usage section. You may need to widen the terminal session window to show it all on one line so that it is more easily readable; at least one section, memory statistics, is 171 columns in width. You can also shrink the terminal session font size to aid in achieving this. Of course there are limits and some of my older monitors are just too small.

```
[student@studentvm1 ~]$ sar -A | less
```

By default the **sar** command shows the data collected for today up to the current time. Data for days up to 1 month in the past can be located in files in the /var/log/sa directory. The files are named saXX where XX is the day of the month. To see data from a previous day, use the following command. It is not necessary to do this as the root user, although I have done so. Be sure to use the name of a file that is present in your own sa directory.

```
[root@studentvm1 sa]# cd /var/log/sa ; ls
sa01  sa04  sa07  sa10  sa13  sa16  sa19  sa22  sa25   sar03  sar06  sar09
sar12  sar15  sar18  sar22
sa02  sa05  sa08  sa11  sa14  sa17  sa20  sa23  sar01  sar04  sar07  sar10
sar13  sar16  sar19  sar23
```

```
sa03  sa06  sa09  sa12  sa15  sa18  sa21  sa24  sar02  sar05  sar08  sar11
sar14  sar17  sar20  sar24
[root@studentvm1 sa]# sar -A -f sa07 | less
```

The preceding command displays all of the data for the 7th day of the month and pipes it to the `less` command.

The large amount of data produced by SAR can be daunting to try to interpret, but I have found it to be very useful in locating various types of problems.

I suggest that you spend some time on a regular basis to look through the SAR results. This will provide you with some knowledge of what your system should look like when it is running correctly. That will make performance problems easier to spot when they do occur.

The SAR man page has a lot of information about the data collected and how to display specific types of data such as disk, CPU, network, and others. Despite that, many of the headings in the SAR reports can be difficult to decipher at first. Much googling has turned up very little in the way of decoding keys for the SAR report column headings, but I did find one web site that has the best descriptions that I have discovered anywhere.[1] The best book I have found in my own Linux reference collection, one that contains many references to SAR and its use, is *The Unix and Linux System Administration Handbook*.[2] Most other books that cover SAR stick to CPU statistics, but SAR provides far more data than that and this book covers at least some of that.

We covered SAR in some detail in Volume 1, Chapter 13, so there is no need to spend more time on it here.

# logrotate

Before we examine other log files, we need to explore the logrotate facility. Many system services and programs dump log entries into log files which enable us as SysAdmins to view them and locate the causes of some types of system problems. That is a good thing.

[1]Computer Hope web site, www.computerhope.com/unix/usar.htm

[2]Nemeth, Evi, et al, ***The Unix and Linux System Administration Handbook***, Pearson Education, Inc., ISBN 978-0-13-148005-6. This title is also available on Amazon in Kindle format.

But what if those log files were to grow to many gigabytes in size and perhaps ultimately fill up the /var filesystem? That would not be so good. I have had that happen and many errors are generated, but they cannot be stored in the log files because there is no more room on the filesystem for them. Other symptoms include programs refusing to start because there is no room to create their PID files on /var or running programs being unable to perform certain tasks because they cannot open files due to being unable to create lock files on /var. It gets messy very quickly.

The logrotate facility is designed to prevent these potential problems. It accomplishes this by rotating the logs on a regular basis, as its name implies. Log rotation can be triggered by time parameters such as weekly or monthly, as well as by log file size.

The task of log rotation consists of renaming a file such as `messages` by appending the date the file was "closed" to further additions such as `messages-20190501` and starting a new `messages` file. The maximum number of older files to be retained is defined in the configuration files for each service, and if the creation of the new log file results in more than the maximum specified number, the oldest file is deleted.

By default, Fedora specifies four as the maximum number of older files to keep. The system defaults for logrotate are defined in /etc/logrotate.conf. Individual services can override the defaults.

## EXPERIMENT 15-2

This experiment should be performed as the student user because we will not be making any changes to the logrotate configuration, just exploring it.

First, **cat** the /etc/logrotate.conf file. This file is not long so it should fit in your terminal session. This file is well commented so it should be self-explanatory. One of the options is that of compression. I do not typically compress my log files so I leave this option commented out.

Make /etc/logrotate.d the PWD. List the contents of this directory and you can see the configuration files for several different services. Several services do not have separate files in this directory, but they are all aggregated into the single file, rsyslog. The syslog service is the system logger and is responsible for logging messages that are sorted into the appropriate log files listed in the beginning of the configuration file.

Look at the contents of the rsyslog file.

```
[student@studentvm1 logrotate.d]$ cd /etc/lograte.d ; cat rsyslog
/var/log/cron
/var/log/maillog
/var/log/messages
/var/log/secure
/var/log/spooler
{
    missingok
    sharedscripts
    postrotate
        /usr/bin/systemctl kill -s HUP rsyslog.service >/dev/null 2>&1 ||
true
    endscript
}
```

First, we have a list of the log files for which this is the configuration file, then a list of directives contained within curly braces {}. The man page for logrotate lists over 60 directives, but we will only look at the most common ones:

- **missingok:** This means that if any log file is missing, to simply ignore that fact and to not throw an error message. Logrotate is to continue with the next file.
- **sharedscripts:** Instead of running the script that is contained between the postrotate and endscript directives once for each log that is rotated, the script is run once. If there are no scripts that need rotating, the script is not run.
- **postrotate:** This designates that the following script is to be run after the log files are rotated.
- **endscript:** This defines the end of the script.
- **create mode owner group:** This specifies the file mode and ownership of the new log file when it is created.
- **nocreate:** This prevents new log files from being created. The /etc/logrotate.d/chrony file uses this directive to prevent logrotate from creating a new log file. The chronyc program uses its own cyclelogs directive in the script to generate its own new log files.

- **compress:** This specifies that the rotated log file is to be compressed. The current log file is not compressed.
- **delaycompress:** This delays compression of the newly rotated log file until the next rotation cycle so that not only the current log file but also the most recently rotated one is uncompressed and can be easily viewed without having to be decompressed.
- **notifempty:** Do not rotate the log if the file is empty.
- **rotate X:** This defines the number, specified by X, of old log files to keep.
- **size Y:** This rotates a log based on a specified size (Y) if that size is exceeded before the specified rotation time is reached. Thus, if a log file is to be rotated weekly but it reaches the size, Y, before the week is complete, the log is rotated anyway.
- Time-related options such as **hourly**, **daily**, **weekly**, **monthly**, and **yearly** define the time intervals at which the logs are to be rotated. Be sure to check the man page for special considerations if a log needs to be rotated hourly.

View the contents of the `dnf` file in the logrotate.d directory. It manages several dnf-related log files, but each has its own stanza despite the fact that they are all configured identically.

Now let's look at the log files themselves. Make /var/log the PWD and list the contents of the directory.

```
[student@studentvm1 ~]$ cd /var/log ; ls
anaconda                        dnf.rpm.log-20190501        messages-20190519
atop                            dnf.rpm.log-20190505        pluto
audit                           dnf.rpm.log-20190512        ppp
blivet-gui                      dnf.rpm.log-20190519        private
boot.log                        firewalld                   README
btmp                            grubby                      sa
btmp-20190501                   hawkey.log                  samba
chrony                          hawkey.log-20190501         secure
cron                            hawkey.log-20190505         secure-20190501
cron-20190501                   hawkey.log-20190512         secure-20190505
cron-20190505                   hawkey.log-20190519         secure-20190512
cron-20190512                   iptraf-ng                   secure-20190519
cron-20190519                   journal                     speech-dispatcher
```

```
cups                          lastlog                    spooler
dnf.librepo.log               lightdm                    spooler-20190501
dnf.librepo.log-20190501      mail                       spooler-20190505
dnf.librepo.log-20190505      maillog                    spooler-20190512
dnf.librepo.log-20190512      maillog-20190501           spooler-20190519
dnf.librepo.log-20190519      maillog-20190505           sssd
dnf.log                       maillog-20190512           tallylog
dnf.log-20190501              maillog-20190519           wtmp
dnf.log-20190505              messages                     Xorg.0.log
dnf.log-20190512              messages-20190501            Xorg.0.log.old
dnf.log-20190519              messages-20190505            Xorg.9.log
dnf.rpm.log                   messages-20190512
```

This shows the log files with the file name extension .log-YYYYMMDD or just the date. These are the older, rotated log files. Some of these entries are directories, and that should be obvious on your terminal session.

The logrotate man page has a description of all the options available for use in the configuration files.

# messages

The /var/log/messages log files contain kernel and other system-level messages of various types and is another of the files I frequently use to assist me with problem determination. The entries found in the messages logs are not usually performance related and are more informational.

Entries from the kernel, systemd, the DHCP client, and many of the running services are logged here. Each log entry begins with the date and time to make it easy to determine the sequence of events and to locate entries made at specific times in the log file. The messages log files are full of interesting and useful information:

- User logins and logouts
- DHCP client requests for network configuration
- The resulting DHCP configuration information as shown by the NetworkManager
- Data logged by systemd during startup and shutdown

- Kernel data about things such as USB memory devices when they are plugged in
- USB hub information
- And much more

The messages file is usually the first place I look when working on non-performance issues. It can also be useful for performance issues, but I start with SAR for that.

Because it is so important, let's take a quick look at the messages file.

## EXPERIMENT 15-3

Perform this experiment as the root user. Make /var/log the PWD. Use the `less` command to view the messages log file.

```
[root@studentvm1 ~]# cd /var/log ; less messages
<snip>
May 21 07:56:49 studentvm1 dhclient[1211]: DHCPREQUEST on enp0s3 to 10.0.2.3
port 67 (xid=0xb11eed75)
May 21 07:56:49 studentvm1 dhclient[1211]: DHCPACK from 10.0.2.3 (xid=0xb11eed75)
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2008]
dhcp4 (enp0s3):   address 10.0.2.7
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2009]
dhcp4 (enp0s3):   plen 24 (255.255.255.0)
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2009]
dhcp4 (enp0s3):   gateway 10.0.2.1
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2009]
dhcp4 (enp0s3):   lease time 1200
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2009]
dhcp4 (enp0s3):   nameserver '192.168.0.52'
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2009]
dhcp4 (enp0s3):   nameserver '8.8.8.8'
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2009]
dhcp4 (enp0s3):   nameserver '8.8.4.4'
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2010]
dhcp4 (enp0s3):   domain name 'both.org'
```

```
May 21 07:56:49 studentvm1 NetworkManager[1048]: <info>  [1558439809.2010]
dhcp4 (enp0s3): state changed bound -> bound
May 21 07:56:49 studentvm1 dbus-daemon[902]: [system] Activating via systemd:
service name='org.freedesktop.nm_dispatcher' unit='dbus-org.freedesktop.
nm-dispatcher.service' requested by ':1.15' (uid=0 pid=1048 comm="/usr/sbin/
NetworkManager --no-daemon ")
May 21 07:56:49 studentvm1 systemd[1]: Starting Network Manager Script
Dispatcher Service...
May 21 07:56:49 studentvm1 dbus-daemon[902]: [system] Successfully activated
service 'org.freedesktop.nm_dispatcher'
May 21 07:56:49 studentvm1 audit[1]: SERVICE_START pid=1 uid=0
auid=4294967295 ses=4294967295 msg='unit=NetworkManager-dispatcher
comm="systemd" exe="/usr/lib/systemd/systemd" hostname=? addr=? terminal=?
res=success'
<snip>
```

You can see the DHCP request from enp0s3 to the DHCP server and the data provided back to the client including the IP address, gateway, name servers, and more.

Locate and view NetworkManager and USB device messages.

I have included only a little output from my own StudentVM1 host because of the large amount of data that is displayed. Browse through the contents of the messages file to get a feel for the types of messages you will typically encounter. Use **Ctrl-C** to terminate `less`.

# Mail logs

I run my own personal mail server and frequently use the logs to resolve problems. In the case of email, problems tend to be related to the non-delivery of mail or blocking desired email while failing to block spam and other unwanted email.

I find log entries in the /var/log/maillog files that tell me whether an email was delivered or not and sometimes enough information to tell me why it was not delivered. If you run a mail server, you should become very familiar with the maillog files. We will explore mail and maillog files in detail in the third volume of this course, but even our workstation virtual machines will have some entries in the maillog files.

**EXPERIMENT 15-4**

Because the maillog files are not accessible to non-root users, this experiment must be performed as root. Make /var/log the PWD. List the contents and find a maillog file that has a size greater than zero.

```
[root@studentvm1 log]$ ll mail*
-rw------- 1 root root    0 May 19 03:46 maillog
-rw------- 1 root root    0 Apr 21 03:31 maillog-20190501
-rw------- 1 root root 2809 May  2 21:42 maillog-20190505
-rw------- 1 root root 7017 May  9 13:27 maillog-20190512
-rw------- 1 root root 2874 May 17 15:37 maillog-20190519

mail:
total 4
-rw------- 1 root root 1448 May  2 21:42 statistics
```

List the contents of the maillog file that has some content. This information is pretty meaningless right now, but we will look at it in the next course in this series.

# dmesg

**dmesg** is not a log file; it is a command. At one time in the past, there was a log file, /var/log/dmesg, which contained all of the messages generated by the kernel during boot and most messages generated during startup. The startup process begins when the boot process ends, when init or systemd take control of the host.

The **dmesg** command displays all of the messages generated by the kernel including massive amounts of data about the hardware it discovers during the boot process. I always start with this command when looking for bootup problems and hardware issues.

---

**Tip** Much of the hardware data found in the output from **dmesg** can be found in the /proc filesystem.

---

Let's look at a bit of the output from the dmesg command.

## EXPERIMENT 15-5

This experiment can be performed as either the root or the student user.

```
[root@studentvm1 log]# dmesg | less
[    0.000000] Linux version 4.14.5-300.fc27.x86_64 (mockbuild@bkernel01.phx2.
               fedoraproject.org) (gcc version 7.2.1 20170915 (Red Hat 7.2.1-2)
               (GCC)) #1 SMP Mon Dec 11 16:00:36 UTC 2017
[    0.000000] Command line: BOOT_IMAGE=/vmlinuz-4.14.5-300.fc27.x86_64 root=/dev/
                            mapper/fedora_studentvm1-root ro rd.lvm.lv=fedora_
                            studentvm1/root rd.lvm.lv=fedora_studentvm1/swap
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point
                        registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes,
                        using 'standard' format.
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000dffeffff] usable
[    0.000000] BIOS-e820: [mem 0x00000000dfff0000-0x00000000dfffffff] ACPI data
[    0.000000] BIOS-e820: [mem 0x00000000fec00000-0x00000000fec00fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fee00000-0x00000000fee00fff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000fffc0000-0x00000000ffffffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000100000000-0x000000011fffffff] usable
<snip>
```

Each line of data starts with a timestamp accurate to within a microsecond. The timestamp represents the time since the kernel started. The data in this stream can be used to determine whether the kernel recognizes certain devices.

For one example, when a new USB device is plugged in, a number of lines are added to the dmesg data buffer. You should see something similar to that in the following data from near the end of the dmesg data. This data shows the sequence of events as the kernel detects the

new device and the kernel, D-Bus, and udev determine what type of device it is and assign a device name to it. Search for "USB device" in this data stream to locate similar entries for your VM.

```
[319346.149478] usb 1-1: new high-speed USB device number 7 using ehci-pci
[319346.545482] usb 1-1: New USB device found, idVendor=058f, idProduct=6387,
                         bcdDevice= 1.41
[319346.545994] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[319346.546430] usb 1-1: Product: Mass Storage Device
[319346.546987] usb 1-1: Manufacturer: Generic
[319346.547457] usb 1-1: SerialNumber: 1VXLET08
[319346.550084] usb-storage 1-1:1.0: USB Mass Storage device detected
[319346.550545] usb-storage 1-1:1.0: Quirks match for vid 058f pid 6387: 400
[319346.551785] scsi host7: usb-storage 1-1:1.0
[319347.564454] scsi 7:0:0:0: Direct-Access     JetFlash TS512MJF150
                              8.07 PQ: 0 ANSI: 2
[319347.566749] sd 7:0:0:0: Attached scsi generic sg5 type 0
[319347.588842] sd 7:0:0:0: [sde] 1003520 512-byte logical blocks: (514 MB/490 MiB)
[319347.595467] sd 7:0:0:0: [sde] Write Protect is off
[319347.596164] sd 7:0:0:0: [sde] Mode Sense: 03 00 00 00
[319347.602145] sd 7:0:0:0: [sde] No Caching mode page found
[319347.602668] sd 7:0:0:0: [sde] Assuming drive cache: write through
[319347.651234]  sde: sde1
[319347.695786] sd 7:0:0:0: [sde] Attached SCSI removable disk
[320800.360638] usb 1-1: USB disconnect, device number 7
```

Scroll through the data to familiarize yourself with the many different types of data to be found here.

The data displayed by the dmesg command is located in RAM rather than on the hard drive. No matter how much RAM memory you have in your host, the space allocated to the dmesg buffer is limited. When it fills up, the oldest data is discarded as newer data is added.

## secure

The /var/log/secure log file contains security-related entries. This includes information about successful and unsuccessful attempt to log in to the system. Let's look at some of the entries you might see in this file.

## EXPERIMENT 15-6

This experiment must be performed as root. Use the **less** command to view the contents of the secure log file. Ensure that /var/log is the PWD.

```
[root@studentvm1 log]# less secure
May 19 22:23:30 studentvm1 lightdm[1335]: pam_unix(lightdm-greeter:session):
session closed for user lightdm
May 19 22:23:30 studentvm1 systemd[16438]: pam_unix(systemd-user:session):
session opened for user student by (uid=0)
May 19 22:23:31 studentvm1 lightdm[1477]: pam_unix(lightdm:session): session
opened for user student by (uid=0)
May 19 22:23:34 studentvm1 polkitd[990]: Registered Authentication Agent for
unix-session:4 (system bus name :1.1357 [/usr/libexec/xfce-polkit], object
path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_US.utf8)
May 20 11:18:54 studentvm1 sshd[29938]: Accepted password for student from
192.168.0.1 port 52652 ssh2
May 20 11:18:54 studentvm1 sshd[29938]: pam_unix(sshd:session): session
opened for user student by (uid=0)
May 20 17:08:52 studentvm1 sshd[3380]: Accepted publickey for root from
192.168.0.1 port 56306 ssh2: RSA SHA256:4UDdGg3FP5sITB8ydfCb5JDg2QCIrsW4cfoN
gFxhC5A
May 20 17:08:52 studentvm1 sshd[3380]: pam_unix(sshd:session): session opened
for user root by (uid=0)
May 21 07:49:05 studentvm1 sshd[3382]: Received disconnect from 192.168.0.1
port 56306:11: disconnected by user
May 21 07:49:05 studentvm1 sshd[3382]: Disconnected from user root
192.168.0.1 port 56306
May 21 07:49:05 studentvm1 sshd[3380]: pam_unix(sshd:session): session closed
for user root
May 21 08:17:15 studentvm1 login[18310]: pam_unix(login:auth): authentication
failure; logname=LOGIN uid=0 euid=0 tty=tty2 ruser= rhost=  user=root
May 21 08:17:15 studentvm1 login[18310]: pam_succeed_if(login:auth):
requirement "uid >= 1000" not met by user "root"
May 21 08:17:17 studentvm1 login[18310]: FAILED LOGIN 1 FROM tty2 FOR root,
Authentication failure
May 21 08:17:23 studentvm1 login[18310]: pam_unix(login:session): session
opened for user root by LOGIN(uid=0)
```

```
May 21 08:17:23 studentvm1 login[18310]: ROOT LOGIN ON tty2
May 21 13:31:16 studentvm1 sshd[24111]: Accepted password for student from
192.168.0.1 port 54202 ssh2
May 21 13:31:16 studentvm1 sshd[24111]: pam_unix(sshd:session): session
opened for user student by (uid=0)
```

Most of the data in /var/log/secure pertains to records of user logins and logouts and information about whether a password or public key was used for authentication.

This log also contains failed password attempts as shown in the data below the line where I snipped out much of the data in this file.

My primary use for the secure log file is to identify break-in attempts from hackers. But I don't even do that – I use automation tools for that too, in this case, the logwatch tool which we will explore a bit later in this chapter.

# Following log files

Searching through log files can be a time-consuming and cumbersome task even when using tools like `grep` to help isolate the desired lines. Many times while troubleshooting it can be helpful to continuously view the contents of a text format log file especially to see the newest entries as they arrive. Using **`cat`** or **`grep`** to view log files displays the contents at the moment in time the command was entered.

I like to use the **`tail`** command to view the end of the file, but it can be time-consuming and disruptive to my problem determination process to rerun the tail command to see new lines. Use **`tail -f`** to enable the tail command to "follow" the file and immediately display new lines of data as they are added to the end of the log file.

### EXPERIMENT 15-7

Perform this experiment as root. We need two terminal sessions with root logins. These terminal sessions should be in separate windows and arranged so you can see both of them at the same time. If your terminal emulator supports multiple panes, like Tilix does, use two panes for this experiment. In one root terminal session, make /var/log the PWD, then follow the messages file.

```
[root@studentvm1 ~]# cd /var/log
[root@studentvm1 log]# tail -f messages
Dec 24 09:30:21 studentvm1 audit[1]: SERVICE_STOP pid=1 uid=0 auid=4294967295
ses=4294967295 msg='unit=sysstat-collect comm="systemd" exe="/usr/lib/
systemd/systemd" hostname=? addr=? terminal=? res=success'
<snip>
Dec 24 09:37:58 studentvm1 systemd[1]: Starting dnf makecache...
Dec 24 09:37:59 studentvm1 dnf[29405]: Metadata cache refreshed recently.
Dec 24 09:37:59 studentvm1 systemd[1]: Started dnf makecache.
Dec 24 09:40:21 studentvm1 audit[1]: SERVICE_STOP pid=1 uid=0 auid=4294967295
ses=4294967295 msg='unit=sysstat-collect comm="systemd" exe="/usr/lib/
systemd/systemd" hostname=? addr=? terminal=? res=success'
```

Tail displays the last ten lines of the log file and then sits there waiting for more data to be appended. I have deleted some of these lines for brevity.

Let's make some log entries appear. There are several ways to do this, but the easiest is to use the `logger` command. In the second window, enter this command as root to log a new entry to the messages file.

```
[root@studentvm1 ~]# logger "This is test message 1."
```

The following line should have appeared in the other terminal at the end of the messages log file.

```
Dec 24 13:51:46 studentvm1 root[1048]: This is test message 1.
```

We can also use STDIO for this.

```
[root@studentvm1 ~]# echo "This is test message 2." | logger
```

And the results are the same – the message appears in the messages log file.

```
Dec 24 13:56:41 studentvm1 root[1057]: This is test message 2.
```

Note that your VM may pop additional messages on this log file while you are performing this experiment; in real life log messages are added to these files quite frequently. Use Ctrl-C to terminate following the log file.

# systemd journals

systemd has its own set of logs – journals, actually – many of which replace the traditional ASCII text files found in the /var/log directory. The journald daemon collects and manages messages for services managed by systemd. The **journalctl** command is used by SysAdmins to view and manipulate the systemd logs.

The intent of using systemd to manage the logs is to provide a central point of control for all of the log-producing entities in a Linux host.

Let's explore the basics of using **journalctl**.

**EXPERIMENT 15-8**

This experiment must be run as root. First let's look at the output we get with no options. By default, the results are piped through the `less` utility.

```
[root@studentvm1 ~]# journalctl
-- Logs begin at Sat 2017-04-29 18:10:23 EDT, end at Wed 2017-12-27 11:30:07 EST. --
Apr 29 18:10:23 studentvm1 systemd-journald[160]: Runtime journal (/run/log/
journal/) is 8.0M, max 197.6M,
Apr 29 18:10:23 studentvm1 kernel: Linux version 4.8.6-300.fc25.x86_64
                                   (mockbuild@bkernel02.phx2.fedorapro
Apr 29 18:10:23 studentvm1 kernel: Command line: BOOT_IMAGE=/vmlinuz-
                                   4.8.6-300.fc25.x86_64 root=/dev/mappe
Apr 29 18:10:23 studentvm1 kernel: x86/fpu: Supporting XSAVE feature 0x001:
                                   'x87 floating point registers'
Apr 29 18:10:23 studentvm1 kernel: x86/fpu: Supporting XSAVE feature 0x002:
                                   'SSE registers'
Apr 29 18:10:23 studentvm1 kernel: x86/fpu: Supporting XSAVE feature 0x004:
                                   'AVX registers'
```

I have only shown a small portion of the output from the journalctl command. It should look somewhat familiar because it is. This is much of the same information as the dmesg command provides. The main differences are that the timestamp for dmesg is in seconds since boot and the timestamps for journalctl are in a standard date and time format and that journalctl will probably go further back in time than dmesg. The dmesg data only goes back to the most recent boot, and even some of the oldest messages may get dropped off if new ones fill the dmesg buffer which is fixed in size. The systemd journal entries can cover months of

time through multiple reboots. While viewing the journal, look at the first line which tells you the journal date ranges and then search on "Reboot" to locate the instances where the host was rebooted.

```
May 30 14:34:46 studentvm1 pulseaudio[4524]: E: [pulseaudio] bluez5-util.c:
GetManagedObjects() failed: org.freed>
May 30 14:36:46 studentvm1 systemd[4517]: Starting Mark boot as successful...
May 30 14:36:46 studentvm1 systemd[4517]: Started Mark boot as successful.
May 30 14:39:56 studentvm1 sshd[4764]: Received disconnect from 192.168.0.1
port 53532:11: disconnected by user
May 30 14:39:56 studentvm1 sshd[4764]: Disconnected from user student1
192.168.0.1 port 53532
-- Reboot --
May 31 17:20:33 studentvm1 kernel: Linux version 5.0.7-200.fc29.x86_64
(mockbuild@bkernel04.phx2.fedoraproject.or>
May 31 17:20:33 studentvm1 kernel: Command line: BOOT_IMAGE=/
vmlinuz-5.0.7-200.fc29.x86_64 root=/dev/mapper/fedor>
May 31 17:20:33 studentvm1 kernel: x86/fpu: Supporting XSAVE feature 0x001:
'x87 floating point registers'
May 31 17:20:33 studentvm1 kernel: x86/fpu: Supporting XSAVE feature 0x002:
'SSE registers'
May 31 17:20:33 studentvm1 kernel: x86/fpu: Supporting XSAVE feature 0x004:
'AVX registers'
```

Take some time to page through the results and explore the types of log entries.

One of the features I learned about while researching this experiment is the ability to define a specific time frame in which to search for log entries. One example is shown here. Be sure to use dates that make sense for your VM host.

```
[root@studentvm1 ~]# journalctl --since 2017-12-20 --until 2017-12-24
```

It is also possible to specify times of day and to use fuzzy times like "yesterday" and usernames to further define the results.

```
[root@studentvm1 ~]# journalctl --since yesterday -u NetworkManager
-- Logs begin at Sat 2017-04-29 18:10:23 EDT, end at Wed 2017-12-27 11:50:07 EST. --
Dec 26 00:09:23 studentvm1 dhclient[856]: DHCPREQUEST on enp0s3 to
192.168.0.51 port 67 (xid=0xaa5aef49)
Dec 26 00:09:23 studentvm1 dhclient[856]: DHCPACK from 192.168.0.51
(xid=0xaa5aef49)
```

```
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5813]
dhcp4 (enp0s3):   address 192.168.0.101
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5819]
dhcp4 (enp0s3):   plen 24 (255.255.255.0)
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5821]
dhcp4 (enp0s3):   gateway 192.168.0.254
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5823]
dhcp4 (enp0s3):   lease time 21600
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5825]
dhcp4 (enp0s3):   nameserver '192.168.0.51'
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5826]
dhcp4 (enp0s3):   nameserver '8.8.8.8'
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5828]
dhcp4 (enp0s3):   nameserver '8.8.4.4'
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5830]
dhcp4 (enp0s3):   domain name 'both.org'
Dec 26 00:09:23 studentvm1 NetworkManager[731]: <info>  [1514264963.5831]
dhcp4 (enp0s3): state changed bound -> bound
Dec 26 00:09:23 studentvm1 dhclient[856]: bound to 192.168.0.101 -- renewal
in 9790 seconds.
Dec 26 02:52:33 studentvm1 dhclient[856]: DHCPREQUEST on enp0s3 to
192.168.0.51 port 67 (xid=0xaa5aef49)
Dec 26 02:52:33 studentvm1 dhclient[856]: DHCPACK from 192.168.0.51
(xid=0xaa5aef49)
<snip>
```

It is possible to list the previous boots for the system and to view only log entries from the current or a previous boot.

```
[root@studentvm1 ~]# journalctl --list-boots
-26 0b92905fe10b48d59649ade083497994 Mon 2019-04-01 08:59:01 EDT—Tue 2019-04-
09 13:02:03 EDT
-25 2094cec4d589434b8dad3afa864ac031 Tue 2019-04-09 14:15:14 EDT—Wed 2019-04-
10 16:05:13 EDT
-24 2214806f5f414263be6ccad96e30f140 Wed 2019-04-10 16:05:47 EDT—Wed 2019-04-
10 16:39:51 EDT
-23 b37a0ebda1054788a32d8e047c611645 Wed 2019-04-10 16:40:24 EDT—Thu 2019-04-
11 21:40:32 EDT
```

```
-22 a40dede07aed4376a4e76d65b760021b Thu 2019-04-11 21:41:06 EDT—Sun 2019-04-
14 09:22:49 EDT
-21 ab3f94c93a9c4746bdbc378acd471bde Tue 2019-04-16 04:16:04 EDT—Tue 2019-04-
16 08:18:34 EDT
-20 3d6bd2a0322d44fd844cd1e29eed0d02 Tue 2019-04-16 08:19:58 EDT—Tue 2019-04-
16 08:31:24 EDT
<snip>
 -8 d62a1f3edd144e4fbf54b3b3b2f90785 Thu 2019-05-09 13:07:00 EDT—Thu 2019-05-
09 13:10:50 EDT
 -7 72bebaf81cae439a8b4b2e312825e12e Thu 2019-05-09 13:24:26 EDT—Thu 2019-05-
09 13:26:08 EDT
 -6 cb4e21351e1a4e2fa9cf9aa70659266b Thu 2019-05-09 13:27:03 EDT—Tue 2019-05-
14 08:37:08 EDT
 -5 4b5eb7872593428da0bf5e19cdc8ac5d Tue 2019-05-14 08:37:41 EDT—Tue 2019-05-
14 09:14:05 EDT
 -4 65862e8a9cfc4bc1ab89caf13b460635 Tue 2019-05-14 09:14:38 EDT—Tue 2019-05-
14 11:45:53 EDT
 -3 be199a0240a34a70b105366ce62fc99e Tue 2019-05-14 11:46:26 EDT—Tue 2019-05-
14 12:12:05 EDT
 -2 a8fcd3957ed84a798aace5e7bda41edc Tue 2019-05-14 12:12:39 EDT—Tue 2019-05-
14 15:31:31 EDT
 -1 871499e9ce4a4ddaba3ef6a4785928 08 Tue 2019-05-14 15:33:17 EDT—Fri 2019-05-
17 15:34:53 EDT
  0 f0f0956fd3e8419b8696a0e2d9df38b3 Fri 2019-05-17 15:35:34 EDT—Wed 2019-05-
22 21:49:06 EDT
```

```
[root@studentvm1 ~]# journalctl -b  d62a1f3edd144e4fbf54b3b3b2f90785
```

The identifier for the boot that this command would list is from line 8 in the boot list. Be sure to use an identifier from your own system for this last command.

You could also do the following to see all of the information for a specific boot by its number in the list.

```
[root@studentvm1 ~]# journalctl -b 8
```

I don't show any of the output from the last command because it is long and it has already appeared before in this course. Be sure to spend some time looking through the data from this last command.

As you can see in Experiment 15-8, the systemd logging facilities collect data from the beginning of the boot process to the end of the shutdown. All types of logs are located in the journal database. You can use the search facility of the `less` utility to locate specific entries or you can use the options available within `journalctl` itself.

If you are interested in finding out more about managing systemd logs, you can start with the man page for journalctl. DigitalOcean has an excellent discussion of **`journalctl`**.[3]

# logwatch

Using tools like **`grep`** and **`tail`** to view a few lines from a log file while working on a problem is fine. But what if you need to search through a large number of log files? That can be tedious even when using those tools.

Logwatch is a tool that can analyze the system log files and detect anomalous entries that the SysAdmin should look at. It generates a report every night around 03:30 AM that is triggered by a shell script in /etc/cron.daily. The Logwatch report condenses thousands of lines of log files into a report that can be scanned by the SysAdmin to more easily locate those entries that may constitute a problem.

The default configuration is for Logwatch to email a report of what it finds in the log files to root. There are various methods for ensuring that the email gets sent to someone and someplace other than root on the localhost. One option is to set the mailto address in the local configuration file in the /etc/logwatch directory. The default configuration files are located in /usr/share/logwatch.

Logwatch can also be run from the command line and the data is sent to STDOUT.

**EXPERIMENT 15-9**

This experiment must be performed as root. Our objective is to run Logwatch from the command line and view the results. First we need to install it.

```
[root@studentvm1 log]# dnf -y install logwatch
```

[3]DigitalOcean, *How To Use Journalctl to View and Manipulate Systemd Logs*, www.digitalocean.com/community/tutorials/how-to-use-journalctl-to-view-and-manipulate-systemd-logs

I have used output from Logwatch from my personal workstation because there is more to see than there is on my instance of StudentVM1. Note that logwatch always scans the previous day's log entries by default although other dates can be specified.

In this command we specify a detail level of 10, which provides the most detail. I have once again removed a few pages and some empty lines of data to save space.

```
[root@david ~]# logwatch --detail 10 | less

 ################### Logwatch 7.5.1 (01/22/19) ####################
        Processing Initiated: Wed May 22 22:04:13 2019
        Date Range Processed: yesterday
                              ( 2019-May-21 )
                              Period is day.
        Detail Level of Output: 10
        Type of Output/Format: stdout / text
        Logfiles for Host: david
 ##################################################################
  --------------------- Kernel Audit Begin ------------------------

 **Unmatched Entries**
    audit[376]: CRYPTO_KEY_USER pid=376 uid=0 auid=4294967295 ses=4294967295
msg='op=destroy kind=server fp=SHA256:30:1c:bc:f3:39:14:d0:98:26:08:42:17:42:
df:55:01:0f:65:b9:7b:28:a3:7b:f9:df:39:15:af:8a:16:4c:a7 direction=? spid=376
suid=0  exe="/usr/sbin/sshd" hostname=? addr=? terminal=? res=success': 1
Time(s)
    audit[32194]: CRED_DISP pid=32194 uid=0 auid=0 ses=124
msg='op=PAM:setcred grantors=pam_env,pam_fprintd acct="root" exe="/usr/sbin/
crond" hostname=? addr=? terminal=cron res=success': 1 Time(s)
    audit[32310]: USER_END pid=32310 uid=0 auid=0 ses=126 msg='op=login id=0
exe="/usr/sbin/sshd"
<snip>
    audit[377]: CRYPTO_KEY_USER pid=377 uid=0 auid=0 ses=127 msg='op=destroy
kind=server fp=SHA256:30:1c:bc:f3:39:14:d0:98:26:08:42:17:42:df:55:01:0f:65:b
9:7b:28:a3:7b:f9:df:39:15:af:8a:16:4c:a7 direction=? spid=377 suid=0  exe="/
usr/sbin/sshd" hostname=? addr=? terminal=? res=success': 1 Time(s)
  ---------------------- Kernel Audit End -------------------------
```

```
    ---------------------- Cron Begin ------------------------
    Commands Run:
       User root:
          /sbin/hwclock --systohc --localtime: 1 Time(s)
          run-parts /etc/cron.hourly: 24 Time(s)
          systemctl try-restart atop: 1 Time(s)
          time /usr/local/bin/rsbu -vbd1 ; time /usr/local/bin/rsbu -vbd2: 1
   Time(s)
     ---------------------- Cron End -------------------------
    ---------------------- Kernel Begin ------------------------
    17 Time(s): radeon_dp_aux_transfer_native: 74 callbacks suppressed
    3 Time(s): scsi 14:0:0:0: Direct-Access     JetFlash TS512MJF150      8.07
               PQ: 0 ANSI: 2
    3 Time(s): scsi host14: usb-storage 1-3:1.0
    3 Time(s): sd 14:0:0:0: Attached scsi generic sg5 type 0
    3 Time(s): sd 14:0:0:0: [sdi] 1003520 512-byte logical blocks: (514 MB/490 MiB)
    3 Time(s): sd 14:0:0:0: [sdi] Assuming drive cache: write through
    3 Time(s): sd 14:0:0:0: [sdi] Attached SCSI removable disk
    3 Time(s): sd 14:0:0:0: [sdi] No Caching mode page found
    3 Time(s): sd 14:0:0:0: [sdi] Write Protect is off
    3 Time(s): sdi: sdi1
    1 Time(s): test1
    1 Time(s): test2
    1 Time(s): usb 1-3: USB disconnect, device number 22
    7 Time(s): usb 1-3: reset high-speed USB device number 22 using xhci_hcd
    3 Time(s): usb-storage 1-3:1.0: Quirks match for vid 058f pid 6387: 400
    3 Time(s): usb-storage 1-3:1.0: USB Mass Storage device detected
    1 Time(s): usblp 1-10.4:1.0: usblp0: USB Bidirectional printer dev 16 if 0
               alt 0 proto 2 vid 0x04F9 pid 0x0042
    1 Time(s): usblp0: removed
     ---------------------- Kernel End -------------------------
    -------------------- pam_unix Begin ------------------------
    sshd:
       Sessions Opened:
          root: 2 Time(s)
```

```
 systemd-user:
    Sessions Opened:
       root: 2 Time(s)

<snip>
 ---------------------- sendmail Begin ------------------------

 STATISTICS
 ----------

 Messages To Recipients:          8
 Addressed Recipients:            8
 Bytes Transferred:          165419
 Messages No Valid Rcpts:         0

 Message Size Distribution:
 Range           # Msgs       KBytes
 0 - 10k              2            1
 10k - 20k            0            0
 20k - 50k            6          160
 ---------------------------------
 TOTAL                8          161
 Avg. Size                        20

 Message recipients per delivery agent:
 Name          # Rcpts
 relay                4
 ---------------------
 TOTAL:               4
 in addition to       4 relay
      submission(s) from MSP

 Top 10 Email Recipients

     david@both.org                                              3 Times
     root                                                        2 Times
     dboth@millennium-technology.com                             2 Times
     root@localhost                                              1 Time

 STARTTLS used the following encryption mechanisms

 ---------------------- sendmail End ------------------------
```

```
 --------------------- SSHD Begin ------------------------
 Users logging in through sshd:
    root:
       192.168.0.1 (david.both.org): 2 Times
  ---------------------- SSHD End ------------------------
 --------------------- Systemd Begin ------------------------

 Reached target Exit the Session: 2 Time(s)

 Started:
     Cleanup of Temporary Directories: 1 Time(s)
     Exit the Session: 2 Time(s)

<snip>
     system activity accounting tool: 144 Time(s)
     update of the root trust anchor for DNSSEC validation in unbound: 1
Time(s)

 User Sessions:
     root:  124 126 127 128

<snip>
    unbound-anchor.service: Succeeded.: 1 Time(s)
    user-runtime-dir@0.service: Succeeded.: 2 Time(s)
    user@0.service: Succeeded.: 2 Time(s)

 ---------------------- Systemd End ------------------------
 --------------------- Disk Space Begin ------------------------

 Filesystem                      Size  Used Avail Use% Mounted on
 /dev/mapper/vg_david1-root      9.8G  431M  8.9G   5% /
 /dev/mapper/vg_david1-usr        45G   16G   27G  38% /usr
 /dev/mapper/vg_david2-Experiments  492G   26G  441G   6% /Experiments
 /dev/mapper/vg_david3-home      246G   49G  187G  21% /home
 /dev/mapper/vg_david2-stuff     246G  122G  112G  53% /stuff
 /dev/mapper/vg_david2-Virtual   787G  463G  284G  62% /Virtual
 /dev/mapper/vg_david1-tmp        45G   84M   42G   1% /tmp
 /dev/mapper/vg_david1-var        20G  9.7G  9.0G  52% /var
 /dev/sdb2                       4.9G  440M  4.2G  10% /boot
 /dev/sdb1                       5.0G   18M  5.0G   1% /boot/efi
 /dev/mapper/vg_Backups-Backups  3.6T  1.7T  1.8T  49% /media/Backups
```

```
 /dev/sdh1                          458G  213G  223G  49% /run/media/dboth/
USB-X47GF
 /dev/sde1                          3.6T  1.3T  2.2T  39% /media/4T-Backup
 ---------------------- Disk Space End -------------------------
 --------------------- lm_sensors output Begin ------------------------
 asus-isa-0000
 Adapter: ISA adapter
 cpu_fan:        0 RPM

 radeon-pci-6500
 Adapter: PCI adapter
 temp1:        +41.5 C  (crit = +120.0 C, hyst = +90.0 C)

 nct6796-isa-0290
 Adapter: ISA adapter
 Vcore:                     +0.93 V  (min =  +0.00 V, max =  +1.74 V)
 in1:                       +1.00 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
 AVCC:                      +3.38 V  (min =  +2.98 V, max =  +3.63 V)
 +3.3V:                     +3.33 V  (min =  +2.98 V, max =  +3.63 V)
 in4:                       +1.02 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
 in5:                       +0.00 V  (min =  +0.00 V, max =  +0.00 V)
 in6:                       +0.60 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
 3VSB:                      +3.38 V  (min =  +2.98 V, max =  +3.63 V)
 Vbat:                      +3.18 V  (min =  +2.70 V, max =  +3.63 V)
 in9:                       +1.03 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
 in10:                      +0.60 V  (min =  +0.00 V, max =  +0.00 V)  ALARM
 in11:                      +0.42 V  (min =  +0.00 V, max =  +0.00 V)  ALARM

<snip>

 coretemp-isa-0000
 Adapter: ISA adapter
 Package id 0:  +53.0 C  (high = +86.0 C, crit = +96.0 C)
 Core 0:        +50.0 C  (high = +86.0 C, crit = +96.0 C)
 Core 1:        +53.0 C  (high = +86.0 C, crit = +96.0 C)
 Core 2:        +49.0 C  (high = +86.0 C, crit = +96.0 C)
 Core 3:        +52.0 C  (high = +86.0 C, crit = +96.0 C)
 Core 4:        +49.0 C  (high = +86.0 C, crit = +96.0 C)
 Core 5:        +51.0 C  (high = +86.0 C, crit = +96.0 C)
 Core 6:        +40.0 C  (high = +86.0 C, crit = +96.0 C)
```

```
Core 7:        +48.0 C  (high = +86.0 C, crit = +96.0 C)
Core 8:        +52.0 C  (high = +86.0 C, crit = +96.0 C)
Core 9:        +53.0 C  (high = +86.0 C, crit = +96.0 C)
Core 10:       +53.0 C  (high = +86.0 C, crit = +96.0 C)
Core 11:       +50.0 C  (high = +86.0 C, crit = +96.0 C)
Core 12:       +50.0 C  (high = +86.0 C, crit = +96.0 C)
Core 13:       +48.0 C  (high = +86.0 C, crit = +96.0 C)
Core 14:       +53.0 C  (high = +86.0 C, crit = +96.0 C)
Core 15:       +51.0 C  (high = +86.0 C, crit = +96.0 C)

---------------------- lm_sensors output End ------------------------

####################### Logwatch End #########################
```

Page through the data produced by Logwatch and be sure to look for the kernel, cron, disk space, and systemd sections. If you have a physical host on which to run this experiment and if the lm_sensors package is installed, you may also see a section showing temperatures in various parts of the hardware, including that for each CPU.

We can use options to cause Logwatch to display the log data from previous days and for specific services. Note that using ALL when specifying the services to scan results in significantly more results than when no service is specified. The list of valid services is located in the default configuration tree for Logwatch: /usr/share/logwatch/scripts/services.

Try this with the following commands.

```
[root@studentvm1 ~]# logwatch --service systemd | less
[root@studentvm1 ~]# logwatch --service systemd --detail high | less
[root@studentvm1 ~]# logwatch --detail high | less
[root@studentvm1 ~]# logwatch --detail 1 | less
[root@studentvm1 ~]# logwatch --service ALL --detail high | less
```

We can also tell Logwatch to report on log entries for all of the stored logs and not just for yesterday. This one took a couple minutes for me but may be less for your VM.

```
[root@studentvm1 ~]# logwatch --service ALL --range All | less
```

The ALL specifications are not case sensitive. This is an anomaly in the usually lowercase world of Linux.

Or a day or range of days in the past. The --range option takes fuzzy entries like the following list. These should all be self-explanatory, but check the Logwatch man page in case you have questions:

- --range today
- --range yesterday
- --range “4 hours ago for that hour”
- --range “-3 days”
- --range “since 2 hours ago for those hours”
- --range “between -10 days and -2 days”
- --range “Apr 15, 2005”
- --range “first Monday in May”
- --range “between 4/23/2005 and 4/30/2005”
- --range “2005/05/03 10:24:17 for that second”

```
[root@studentvm1 ~]# logwatch --detail high --range "-3 days" | less
```

All of these options allow us to easily search for log entries using various criteria. Try some differing combinations of your own devising.

The sections that appear in the Logwatch output depends upon the software packages you have installed on your Linux computer. So if you are looking at the output from Logwatch for a basic installation rather than a primary workstation or even a server, you will see far fewer entries.

Since 2014 Logwatch has been able to search the journald database for log entries.[4] This compatibility with the systemd logging facility ensures that a major source of log entries is not ignored.

Logwatch runs once daily and is triggered by a Bash shell script in /etc/cron.daily. This script sets the output of the logwatch command to email which, by the default, Linux email configuration is sent to root. The /etc/aliases file defines where email addressed to root is sent.

[4]SourceForge, Logwatch repository, https://sourceforge.net/p/logwatch/patches/34/

## EXPERIMENT 15-10

Perform this experiment as root. Make /etc/cron.daily the PWD and then look at the contents of the file, 0logwatch. The zero at the beginning of the name ensures that this shell script is run before any other scripts in this directory. The scripts are run in alphanumeric sorted order.

```
[root@studentvm1 ~]# cd /etc/cron.daily/ ; ll ; cat 0logwatch
total 8
-rwxr-xr-x  1 root root 486 Jan 28 06:22 0logwatch
-rwxr-xr-x. 1 root root 193 Jan  4  2018 logrotate
#!/usr/bin/sh

#Set logwatch executable location
LOGWATCH_SCRIPT="/usr/sbin/logwatch"

# Add options to the OPTIONS variable. Most options should be defined in
# the file /etc/logwatch/conf/logwatch.conf, but some are only for the
# nightly cron run such as "--output mail" and should be set here.
# Other options to consider might be "--format html" or "--encode base64".
# See 'man logwatch' for more details.
OPTIONS="--output mail"

#Call logwatch
$LOGWATCH_SCRIPT $OPTIONS

exit 0
```

This script scans the log files using the default detail level of 5 where the range is from 0 to 10. We want to change that to the highest setting of 10. Although you can use text equivalents where low = 0, med = 5, and high = 10, I prefer to use the numeric value of 10 when setting the detail level. Edit the 0logwatch file and change the $OPTIONS variable to the following to set the detail level.

```
OPTIONS="--output mail --detail 10"
```

Save the file and make root's home the PWD. The next run of Logwatch triggered by the script located in cron.daily will run at the highest level of detail.

Because Logwatch will not be triggered by cron.daily until about 03:30 AM, wait until tomorrow to perform Experiment 15-11.

## EXPERIMENT 15-11

Perform this task as root. In a terminal session, start the mailx email client.

```
[root@studentvm1 ~]# mailx
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/root": 1 message 1 unread
>U  1 logwatch@studentvm1   Thu May 23 03:32  65/2418  "Logwatch for
studentvm1 (Linux)"
&
```

View the email by entering its number. If your mail queue is longer than just the single message, be sure to use the correct number for the email with "Logwatch" in the subject.

```
& 1
Message  1:
From root@studentvm1  Thu May 23 03:32:07 2019
Return-Path: <root@studentvm1>
Date: Thu, 23 May 2019 03:32:04 -0400
To: root@studentvm1
From: logwatch@studentvm1
Subject: Logwatch for studentvm1 (Linux)
Auto-Submitted: auto-generated
Precedence: bulk
Content-Type: text/plain; charset="UTF-8"
Status: RO

 ################### Logwatch 7.5.1 (01/22/19) ####################
        Processing Initiated: Thu May 23 03:31:04 2019
        Date Range Processed: yesterday
                              ( 2019-May-22 )
                              Period is day.
        Detail Level of Output: 0
        Type of Output/Format: mail / text
        Logfiles for Host: studentvm1
 ##################################################################
```

Exit the email. Then you can press **d** to delete the email. Or you can keep it, if you choose.

# Chapter summary

Logs and journals are a rich source of information for SysAdmins as we work to resolve problems of many kinds. Despite knowing this, I sometimes forget to use the logs and that has prevented me from solving those problems as quickly as I could. When I remember to go to the logs, the answers come quickly.

We have looked at some typical logs, both traditional text ones and the journals produced by systemd. We explored how to access and search these logs. We also used Logwatch to assist us in locating log entries that might indicate a problem.

# Exercises

Perform the following exercises to complete this chapter:

1. Use SAR to view disk activity for 2 days ago, displaying the device names like sda rather than the block device IDs like dev8-16.
2. What tools besides SAR can be used to view and analyze historical performance and event data?
3. What is the default data collection interval for SAR?
4. What would cause security log entries like the following?

```
May 23 12:54:29 studentvm1 login[18310]: pam_
unix(login:session): session closed for user root
May 23 12:54:35 studentvm1 login[20004]: pam_unix(login:auth):
check pass; user unknown
May 23 12:54:35 studentvm1 login[20004]: pam_unix(login:auth):
authentication failure; logname=LOGIN uid=0 euid=0 tty=tty2
ruser= rhost=
May 23 12:54:37 studentvm1 login[20004]: FAILED LOGIN 1
FROM tty2 FOR (unknown), User not known to the underlying
authentication module
May 23 12:54:49 studentvm1 login[20004]: pam_unix(login:auth):
check pass; user unknown
```

```
May 23 12:54:49 studentvm1 login[20004]: pam_unix(login:auth):
authentication failure; logname=LOGIN uid=0 euid=0 tty=tty2
ruser= rhost=
May 23 12:54:52 studentvm1 login[20004]: FAILED LOGIN 2
FROM tty2 FOR (unknown), User not known to the underlying
authentication module
May 23 12:56:04 studentvm1 login[20147]: pam_unix(login:auth):
authentication failure; logname=LOGIN uid=0 euid=0 tty=tty2
ruser= rhost=  user=root
May 23 12:56:04 studentvm1 login[20147]: pam_succeed_
if(login:auth): requirement "uid >= 1000" not met by user "root"
May 23 12:56:05 studentvm1 login[20147]: FAILED LOGIN 1 FROM
tty2 FOR root, Authentication failure
```

5. Use **logwatch** from the CLI to search for all logical volume management (LVM) entries. Did you have any?
6. What minimum detail level must be specified when using Logwatch in order to obtain non-null output for the systemd service?

CHAPTER 16

# Managing Users

## Objectives

In this chapter you will learn

- How user accounts and groups are used to provide access and security
- The function and structures of the passwd, group, and shadow files in /etc
- To add and delete user accounts using basic useradd and userdel commands
- To add and delete user accounts manually
- To create user-level configurations that replicate for new users
- To lock a user account
- The difference between login shells and nologin shells

## Introduction

You are probably asking why I waited so long to talk about managing users. The answer is that many Linux systems today are essentially single-user systems. Very seldom do we see multiple users needing simultaneous access to a Linux computer, but it does happen. In most cases we find that one of several users of a Linux system will log in and **`su -`** to root in order to perform some sort of administrative task. In other environments, several users may log in remotely as non-privileged users to a single system to perform normal work – whatever that might be.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_16

Even if you are the only human with access rights to a particular Linux host, you are still dealing with at least two user accounts, root and your own user account. There are also a number of user accounts that belong to various services and programs on a Linux host.

Much of this chapter is about creating and managing user accounts. We will spend a significant amount of time on the files used to manage user accounts, passwords, and security.

# The root account

Your Linux computer has many accounts even if no other human actually uses your computer on a regular basis. Most of those accounts are used by Linux when it performs particular functions.

One of the special accounts is that of root. The root account is present on all Linux computers, and it allows the person logged in as root to read, change, and delete any file on the computer regardless of who owns the files. The root account is restricted by file permissions, but root can change the permissions of any file on the computer. The root account can do anything and everything on a Linux computer even changing the password of any user or locking out users. To protect the integrity of the system, the only person who should have the root password to a Linux computer is the system administrator.

We explored working as root in Chapter 11 of Volume 1, including reasons not to use **sudo**. In this chapter we are more interested in using the root account to manage other users.

# Your account

By virtue of logging in using your account ID and password, you are granted access to read and write files that are located in your home directory because you are the owner of those files. You can create new files and directories in your home directory and modify them as you see fit.

Your account does not provide you enough rights to access other user's home directories let alone view or modify the files located there. Your account does not have

sufficient rights to alter any important system files, although you may be able to see some of them and view the contents of some.

There is a common practice to create account names using the first letter of your first name and your last name. Thus, the person Jo User would have an account name of juser. Notice that it is also common practice for the account name to be all lowercase. Case is important in Linux, so the account name JUser is not the same as juser.

## Your home directory

Your home directory is where files that belong to you are stored. Another word for directory is folder.

When you create files in your home directory or in any of the subdirectories in your home directory, they are created with the appropriate ownership and permissions to allow you to read and write them. This should allow you to create new documents and spreadsheets and so on and then to be able to modify them as needed and store them back to the disk after they have been modified.

You can also use one of the file managers to change the permissions of the files in your home directory, but we recommend that you do not do so unless you have a very good reason to do so and know exactly why you are doing it.

# User accounts and groups

User accounts and groups are the first line of security on your Linux computer. Knowledge of user accounts and file permissions will make it much less frustrating for you as you do your work.

The root account is always UID 0 and the root group is always GID 0.

Historically, all system-level users were assigned a UID and GID between 1 and 99. Convention defined the specific user and group IDs for various programs and services. For a time, Red Hat, and therefore Fedora, recommended starting human user and group IDs at UID 500 and GID 500. That was inconsistent with other Linux distributions and caused some issues.

Today, because of a proliferation of services and system-level account needs, all of the newer standard Linux system- and application-level users are located in the UID

range between 100 and 999 which is now reserved for this purpose.[1] All application-level users – those required by installed services and applications – should be added in the UID range between 101 and 999. All regular (human) users should be added starting at UID 1000 and above.

The RHEL 7 *System Administrator's Guide* goes further and recommends starting human user IDs at 5000[2] to allow for future expansion of the system IDs. However, the current RHEL and Fedora implementations still begin at UID/GID 1000 for human users. The guide does explain how to make this change so that new users are automatically assigned IDs in the recommended range.

Group ID assignments should follow the same practices as UIDs to help ensure consistency and to make troubleshooting easier.

There are some interesting historical anomalies within this structure. For example, GID 100 is reserved for the "Users" group. In some environments it is common for all regular users to be added to this group, but this is not recommended as it constitutes a security risk that would allow users to have access to one another's files.

For regular users the UID and GID should be identical, that is, a user with UID 1001 should have a GID of 1001 as well. Since each user belongs to its own group, security is enhanced because files are not automatically shared between users. By default, users should not all belong to a single common primary group such as the Users group (GID 100).

Making files available to other users should be accomplished by using secondary group memberships to which the sharing users all belong and a directory where shared files can be stored and which has group ownership by the common group. Group membership should be limited to those who have a specific need to share related files. This allows for more granular management of shared files. We experimented with how to do this in Chapter 18 of Volume 1.

When adding group IDs for things like shared directories and files, I like to choose numbers starting at 5000 and above, as we did in Chapter 18. This allows space for 4000 users with identical UID and GID numbers. That should be more than enough for most Linux installations. Your needs may differ so you should adapt as necessary.

[1]RHEL 7, *System Administrator's Guide*, Red Hat, 2018, 44

[2]Ibid., 44

Figure 16-1 shows the conventional and currently common usage assignments for UID and GID ranges. The range from 0 to 999 and the ID 65534 should be considered as completely unusable for use or assignment by the SysAdmin. The range between 1000 and 65533 can be considered flexible and can be used according to local requirements.

| Description | User ID range | Group ID range |
|---|---|---|
| root | 0 | 0 |
| Historical Linux system level accounts. These are all documented and assigned by convention. | 1 - 99 | 1 - 99 |
| Accounts used by services and applications. This has changed over the years, but this range is now consistent with Unix and other Linux distributions. | 100 - 999 | 100 – 999 |
| Accounts used by regular (Human) users. | 1000 – 4999 | 1000-4999 |
| Miscellaneous – Shared directory and file GIDs, for example. | 5000 - 9999 | 5000 - 9999 |
| Open | 10000 - 65533 | 10000 - 65533 |
| nfsnobody – an anonymous NFS (Network File System) user that is used for access to remote files. | 65534 | 65534 |

***Figure 16-1.*** *Recommended UID and GID numeric ranges*

User and group ID data are stored in the files /etc/passwd, /etc/shadow, and /etc/group.

## The /etc/passwd file

We will start by looking at the user and group information located in the /etc/passwd file. Here we will also look at the other information stored in this file.

## EXPERIMENT 16-1

Much of this experiment should be performed as root. As the root user, the **id** command shows us information about our ID.

```
[root@studentvm1 ~]# id
uid=0(root) gid=0(root) groups=0(root)
[root@studentvm1 ~]#
```

This shows that the UID and GID for root are both 0 (zero) and that the root user is a member of the root group with a GID of 0.

Now, as the root user, let's look at the information for the student user.

```
[root@studentvm1 ~]# id 1000
uid=1000(student) gid=1000(student) groups=1000(student),5000(dev),5001(shared)
```

The student user has both UID and GID set as 1000. The student user also has group memberships in the dev and shared groups. These memberships allow the student user to share files with other users. Look at the student1 user which also has memberships for these shared groups.

```
[root@studentvm1 ~]# id 1001
uid=1001(student1) gid=1001(student1) groups=1001(student1),5000(dev),5001(shared)
```

Now let's look at the file that defines and contains user information. Enter the following command as root.

```
[root@studentvm1 ~]# cat /etc/passwd
```

The result is not sorted in any meaningful way, so let's make it a bit easier by sorting on the UID. This number is located in the third field of each user. The -t option specifies the field delimiter character and -k specifies starting the sort at field 3, the first character. The -g option specifies the use of a general numeric sort. This results in a data stream that is much easier to read.

```
[root@studentvm1 etc]# cat /etc/passwd | sort -t: -k3.1 -g
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
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/
dev/null:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
pulse:x:171:171:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
systemd-resolve:x:193:193:systemd Resolver:/:/sbin/nologin
saslauth:x:976:76:Saslauthd user:/run/saslauthd:/sbin/nologin
pipewire:x:977:977:PipeWire System Daemon:/var/run/pipewire:/sbin/nologin
dictd:x:978:978:dictd dictionary server:/usr/share/dict/dictd:/sbin/nologin
systemd-timesync:x:979:979:systemd Time Synchronization:/:/sbin/nologin
dnsmasq:x:980:980:Dnsmasq DHCP and DNS server:/var/lib/dnsmasq:/sbin/nologin
vboxadd:x:987:1::/var/run/vboxadd:/sbin/nologin
colord:x:988:982:User for colord:/var/lib/colord:/sbin/nologin
setroubleshoot:x:989:983::/var/lib/setroubleshoot:/sbin/nologin
nm-openvpn:x:990:984:Default user for running openvpn spawned by
NetworkManager:/:/sbin/nologin
nm-openconnect:x:991:985:NetworkManager user for OpenConnect:/:/sbin/nologin
lightdm:x:992:986::/var/lib/lightdm:/sbin/nologin
chrony:x:993:987::/var/lib/chrony:/sbin/nologin
```

```
polkitd:x:994:990:User for polkitd:/:/sbin/nologin
openvpn:x:995:991:OpenVPN:/etc/openvpn:/sbin/nologin
sstpc:x:996:992:Secure Socket Tunneling Protocol(SSTP) Client:/var/run/
sstpc:/sbin/nologin
geoclue:x:997:993:User for geoclue:/var/lib/geoclue:/sbin/nologin
unbound:x:998:995:Unbound DNS resolver:/etc/unbound:/sbin/nologin
systemd-coredump:x:999:997:systemd Core Dumper:/:/sbin/nologin
student:x:1000:1000:Student User:/home/student:/bin/bash
student1:x:1001:1001:Student1:/home/student1:/bin/bash
student2:x:1002:1002:Student User 2:/home/student2:/bin/bash
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
```

Let's deconstruct the line for the student account, UID 1000. The field separator for this file is the colon (:). Figure 16-2 describes each of the fields in the /etc/passwd file.

| `student:x:1000:1000:Student User:/home/student:/bin/bash` | | | |
|---|---|---|---|
| **Field** | **Field name** | **Value** | **Description** |
| 1 | Account name | student | The user login name for the account |
| 2 | Password | x | No longer used to store passwords. This field is retained for backward compatibility. |
| 3 | User ID (UID) | 1000 | The User ID number for this account. |
| 4 | Group ID (GID) | 1000 | The primary Group ID number for this account. This is the default and many times the only group to which a user belongs. |
| 5 | GECOS | student | This is a text field that can contain multiple words for a description of the account. GECOS stands for General Electric Comprehensive Operating System. Yes, *that* GE[3]. |
| 6 | Home directory | /home/student | The home directories for users may differ depending upon the organizational needs, specifications, and historical usage. |
| 7 | Shell | /bin/bash | The default shell for this user. Bash is the default shell for most Linux distributions. Users can change their default shell. |

***Figure 16-2.*** *Deconstructing the /etc/passwd entry for the student user*

The password field in account entries is no longer used. Storing a password in this file is a security issue because the file needs to be accessible by all user accounts. Reading a password from this file, even an encrypted password, constitutes a serious security issue. For this reason, passwords have long ago been moved to the /etc/shadow file which is not universally readable and thus is more secure.

## nologin shells

Many of the system users in the /etc/passwd file have a nologin shell, /sbin/nologin. This type of shell is a small shell that does not allow a login of any type. This is a security feature as it prevents crackers from accessing a Linux host system by escalating privileges to or beyond these accounts.

## The /etc/shadow file

As mentioned previously, the password location for user accounts has been moved from /etc/passwd to /etc/shadow. The shadow file is more secure because it is readable only by root and by system processes that run with the root user ID.

**EXPERIMENT 16-2**

Perform this experiment as the root user. We now look at the /etc/shadow file. Your root terminal session should still have /etc as the PWD.

View the content of the /etc/shadow file. I have indented the long lines to help clarify the data. I have also removed a number of lines to save space.

```
[root@studentvm1 etc]# cat shadow
root:$6$/VoB.UfR5MtuBi7b$Wnf7nLT/.
EFz4W4lAOvq1kSoIXhVR4Mnp8XlDJpRHjEpH2gJw348ZIUhE8rsGFRk7yIuh/
2pnKbNBbhz9n./y.::0:99999:7:::
bin:*:17725:0:99999:7:::
daemon:*:17725:0:99999:7:::
adm:*:17725:0:99999:7:::
lp:*:17725:0:99999:7:::
sync:*:17725:0:99999:7:::
<snip>
```

```
sshd:!!:17833::::::
vboxadd:!!:17833::::::
dnsmasq:!!:17833::::::
tcpdump:!!:17833::::::
student:$6$81UxQUSOXIejpASu$ovP6j8Bs/rvyq5j3q/
cWFsajWlhdN8YCySx7glnnBKYEEkBUG9SaAz.mAb8eW.
nXewOoWJx4czjIMay6zbDKq0::0:99999:7:::
systemd-timesync:!!:17889::::::
dictd:!!:17910::::::
student1:$6$iUF3OxuKzezktM4c$Wxqik.oRHLgoxI9qwV3CO2z5/
Q9p7XPgRpHt2qq6D3FlSlSLgUN36FBdIOB9UfxIJz.
b3xoQnHyirnZvisky8/:17987:0:99999:7:::
student2:$6$WgztalwLQSIlMM3w$6lxcmrMOXJietxHbsILmRwFfyKwrxV7
ye/   OLPmZz7TTGVUAgHcvFMM2JX.WLTWncNTGhAGJIpnfDViJwDWAVC.:17987:0:99999:7:::
pipewire:!!:18002::::::
saslauth:!!:18018::::::
mailnull:!!:18018::::::
smmsp:!!:18018::::::
```

Note that only root and other human user accounts have passwords.

Let's deconstruct the shadow file entry for the student user in Figure 16-3. The entries for this file contain 9 colon (:) separated fields.

| student:$6$81UxQUSOXI<snip>::0:99999:7::: | | | |
|---|---|---|---|
| **Field** | **Field name** | **Value** | **Description** |
| 1 | Account name | student | The user login name for the account |
| 2 | Password | $6$81UxQUSOXI<snip> | The encrypted password, also called a hash, and truncated here for brevity. If this field starts with an exclamation point (!) the account is locked. |
| 3 | Date of the last password change. | Empty | The date of the last password change in days since Jan 1, 1970 00:00 UTC. If this field is empty, the password has never been changed. If this field is 0, then the password must be changed at the next login. |
| 4 | Minimum password age. | 0 | If this number is non-zero the user must wait that number of days before changing the password again. This prevents users from doing a required change and then immediately changing the password back to their "favorite" password. |
| 5 | Maximum password age. | 99999 | The number of days that the password will remain valid. The value of 99999 is interpreted as the password will never expire. |
| 6 | Password warning period. | 7 | The number of days left until the password expires during which the user will be warned each day. |
| 7 | Password inactivity period. | Empty | The number of days after password expiration during which the old password will be accepted and the user required to create a new password. |
| 8 | Account expiration date. | Empty | The number of days since Jan 1, 1970 00:00 UTC at which the account will expire. The user will not be allowed to login to an expired account. If this field is empty the account will never expire. This is different from the password expiration. |
| 9 | Reserved | Empty | Reserved for future use. |

***Figure 16-3.*** *Deconstructing the /etc/shadow entry for the student user*

Fields 4 through 8 are typically used to implement password security policies by forcing users to change their passwords on a regular basis. Notice that the student user has not changed their password.

## EXPERIMENT 16-3

Perform this experiment as the student user. Change the student user's password from the command line. Let's start with a bad password to see what happens.

```
[student@studentvm1 ~]$ passwd
Changing password for user student.
Current password: <Enter the current password>
New password: mypassword
BAD PASSWORD: The password fails the dictionary check - it is based on a
dictionary word
passwd: Authentication token manipulation error
```

So entering a password that contains a dictionary word or a string of characters that are typically substituted in dictionary words causes an error and does not allow the password to be changed. Try this – the 0 is a zero.

```
[student@studentvm1 ~]$ passwd
Changing password for user student.
Current password: <Enter the current password>
New password: myp@ssw0rd
BAD PASSWORD: The password fails the dictionary check - it is based on a
dictionary word
passwd: Authentication token manipulation error
```

A non-root user is not allowed to create passwords that do not pass certain minimum criteria. The root user can set any password for any user although the same messages will be displayed as a reminder. But root can do anything.

Let's change the password for real.

```
[student@studentvm1 ~]$ passwd
Changing password for user student.
Current password: <Enter old password>
New password: Yu2iyief
Retype new password: Yu2iyief
passwd: all authentication tokens updated successfully.
```

This works because the password is not too short – it must be at least eight characters in length – and it is a series of random letters, upper- and lowercase, as well as a numeric digit. Now let's see what happens when root changes the student user's password with a dictionary word.

```
[root@studentvm1 etc]# passwd student
Changing password for user student.
New password: myp@ssw0rd
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: myp@ssw0rd
passwd: all authentication tokens updated successfully.
You have new mail in /var/spool/mail/root
```

Root does not need to enter the current password and, although the bad password message is displayed, the password is changed anyway.

---

**Tip** Root has mail in the previous example. This is the daily logwatch email report. Depending upon how fast you are working through this course, you may have seen this already or you may not see it until later.

---

Now look at the shadow file entry for the student user.

```
[root@studentvm1 etc]# grep student shadow
student:$6$.9B/OvGhNwsdf.cc$X/Ed1<snip>dDcB4uBHF.
HVqA1ZGkJ5L5yB2G/:18041:0:99999:7:::
<snip>
```

The third field, the date of the last password change, now has a number in it, 18041 on my host. Your number will be different.

Change the password to something you choose.

---

Note that for a non-root user changing their own password, they must also enter their original password before they can change it. This is a simple security procedure that can prevent passersby from changing a user's password. Remember, root can do anything, including changing the password of any user without the need to know the old password.

## The /etc/group file

The /etc/group file contains a list of all of the groups on the localhost. This includes the standard system groups as well as groups created by the system administrator for specific local use.

We created two groups in Chapter 18 of Volume 1, dev and shared, to allow users to have a place to share files and to work cooperatively. Having already looked at the group file there and earlier in this chapter, there is no need to explore it more.

## The /etc/login.defs file

The /etc/login.defs file is used to set certain default configuration items that are incorporated when adding new users. The values in this file include the starting and maximum UIDs and GIDs for new users, the default mail directory location, and the default password expiration options.

**EXPERIMENT 16-4**

Perform this experiment as the root user. View the contents of the /etc/login.defs file. You can read the comments in that file to understand it a bit, but there are two lines that need some discussion.

**PASS_MIN_LEN:** The minimum length for a password is specified as five characters. This should be set to a minimum of eight to ensure a reasonable amount of security.

**PASS_MAX_DAYS:** This line is set to 99999 so that the passwords never expire. In a real-world environment, passwords should be set to expire no less frequently than every 30 days.

Remember that, although the root user will still see password warnings, root can ignore them. Non-root users are not able to ignore the warnings and must create a password that meets the length policy set in this file.

## Account configuration files

As we have already seen, there are several configuration files that are present in a new account home directory. All user shell configuration files that are located in the /etc/skel directory, such as ~/.bash_profile and ~/.bashrc, are copied into the new account home directory when each new user account is created.

If you have local configuration files that should be in the user's home directories instead of in one of the system-wide configuration files found in /etc/profile.d, you can place them in /etc/skel and the files will be copied to the new home directory. One reason to place them in the account home directory is so that the user can alter them as needed.

## Password security

It is advisable as a good security precaution to change your password about once a month. This prevents other people from using your password for very long even if they happen to discover it. Once you have changed it, they can no longer use your previous password to access the system. You never know when or how someone might obtain or guess your password, so even if you do not think it has been compromised, you should change your password regularly. Of course a password should be changed immediately if you suspect that it has been compromised.

Passwords should be protected and never written down. If a password is stolen, it can be used to access your computer and the network if your computer is so connected and thus compromise your data.

Linux requires passwords to be a specified minimum length. The default is five characters but that can be changed. I recommend that you use a longer password to increase the difficulty of someone guessing your password. Passwords should never be dates, initials, acronyms, words, or easy-to-remember sequences such as "ASDFG" from the left of the middle row of the keyboard. Passwords should be composed of upper- and lowercase alphabetic characters as well as numbers and special characters.

My personal calculations show that an automated attack of 500 access attempts per second on a host with a five-character password can be cracked in anywhere from 6 hours to 21 days depending upon whether the password contains only lowercase alpha or upper- and lowercase as well as numbers. The time to crack a really good randomly generated password that includes special characters (#$%^, etc.) rises to 152 days. This should be set to a minimum of eight to ensure a reasonable amount of security. The time to crack an eight-character password rises to 13 years to over 325,000 years, once again depending upon using upper- and lowercase, numbers, and special characters.

Crackers – bad hackers, people who want to get into your computer – have dictionaries of words, common acronyms, and key sequences that they can try to attempt to crack into your system. They also try easy-to-guess sequences that are

available to anyone with a little persistence such as birthdays; anniversaries; the names of spouses, children, pets, or significant others; Social Security numbers; and other possible passwords of this type. The point is that when you change your password, you should choose one that is not based on a dictionary word or one that will be easy to guess or deduce from your personal information. Passwords based on any of these non-random sources will likely be cracked in seconds.

There is a significant downside of changing passwords frequently and setting strict policies that require them to be excessively long and not easily memorized. Such policies will almost certainly result in users who write their passwords on post-it notes and stick them on the display or under the keyboard. There is a fine line between workable security and self-defeating security.

## Password encryption

Passwords cannot be safely stored on the hard drive in plain text format as this would leave them open to incredibly easy hacking. In order to ensure that the user accounts are secure, passwords are encrypted using the OpenSSL encryption libraries. The **`openssl`** command-line tool can be used to access the encryption libraries from the command line so that we can explore a bit about password encryption.

**EXPERIMENT 16-5**

Perform this experiment as the student user. The **`openssl passwd`** command-line utility allows encryption of a plain text string into an encrypted password that can be used when creating a new account. It also lets us explore the structure of the passwords in the /etc/shadow file.

Starting with a simple example and without specifying a specific encryption method, the password is truncated to eight characters and encrypted with the crypt algorithm which is not particularly secure. For most of the examples in this experiment, we will use the string "mypassword" for the password to be encrypted.

```
[student@studentvm1 ~]$ openssl passwd mypassword
Warning: truncating password to 8 characters
Gvp.W6K7c5FJc
```

Compare the password generated to that of the student1 password in the /etc/shadow file, which looks like that in the following data. This password hash is much longer than we got from using the default settings.

```
student1:$6$wVc137Z/fbPCOdrK$VhTwe4ooOxVrYWiZp3Z1mHDSpEBGopGjTho6OdjOYEbzkSN
LeoSc5k7njmqdSYbLadnkBqCQcXHxLF.f42sG..:18046:0:99999:7:::
```

I have highlighted the first three characters of the password because they tell us – and the system password and login utilities – the encryption algorithm used to create the password. $6$ means that the SHA512 bit algorithm was used to create this password. We can specify the SHA512 algorithm with the -6 option.

```
[student@studentvm1 ~]$ openssl passwd -6 mypassword
$6$uMdQ9QJqEgeQXEEn$mAcO.fzVdtsdLFyZPkNe6HdTI3nqYYnOrf8S9Jz3U.yiDFOoyWtbcbA
TNmwLzrUyfqsPPHeOlaDXMoax6xZME0
```

Note the $6$ characters at the beginning of this password indicating that it is SHA512. Now do the same with the SHA256 option.

```
[student@studentvm1 ~]$ openssl passwd -5 mypassword
$5$TwLEuxmTGPbdZyFn$6yj9vc/OOyKnU4hmZ.x3DWsfRNk5fm3ywtGNYqRLK74
```

Notice the first three characters are now $5$ and that the password is shorter.

Open the openssl-passwd man page (**man openssl-passwd**) and view the other encryption options available. Create password hashes using crypt, MD5, apr1, SHA256 (-5), and SHA512 (-6) algorithms.

Let's get back to the SHA512 algorithm. Run the command several times in sequence.

```
[student@studentvm1 ~]$ openssl passwd -6 mypassword
```

Notice that the password hash is always different despite the fact that the password string is the same. This is because the password encryption algorithms use a different random seed for every iteration. This is called the "salt,[3]" presumably because it spices things up a bit. Normally, the salt is taken from the /dev/urandom data stream. This adds a bit of randomness into the algorithm and produces a different result for every iteration.

[3]The Free On-line Dictionary of Computing (FOLDOC) defines salt as "A tiny bit of near-random data inserted where too much regularity would be undesirable;…."

We also can specify a salt to use with the algorithm instead of using a random one, using the -salt option. Execute the following command several times; it will always produce the same result.

```
[student@studentvm1 ~]$ openssl passwd -salt 123456 -6 mypassword
$6$123456$KKcK3jDXxN5TVYNLbMdEljnfRjaSlbqj5X9bBgryaa4qLD04lrM9kswCpAZL27/
WXlbsDQcJ8kBxPjcpips781
```

Notice that the salt string is included after the algorithm specifier, $6$123456$. Using the same salt string removes the randomness from the algorithm to always produce the same result, given the same password. Using a different string for the salt creates a new password hash.

Go back and look at some of the passwords you created without the -salt option and locate the random string.

---

Theoretically, the well-known algorithms used to generate password hashes do so in such a way that there is no known algorithm that can reverse the process and generate the plain text password from the password hash. However, if a cracker has access to the hash – which also contains the salt string – a brute force attack could conceivably eventually find the plain text that generated the hash. This would not take long if the password was based on dictionary words, but could take years if good passwords were used.

## Generating good passwords

Creating good passwords is a challenge. It can take some thought and effort. Linux has at least one tool that provides us with suggestions for good passwords.

The **pwgen** utility gives us the ability to specify the number and length of passwords we want to generate as well as to make them relatively easy or impossible to remember. By default, when used without the -s (secure) and -y (use special characters) options, the resulting passwords are alleged to be easy for humans to memorize. My experience is that some are and others not so much.

## EXPERIMENT 16-6

This experiment can be performed as the student user. We explore the **pwgen** utility to learn how it can help us create reasonably secure passwords.

Start by using **pwgen** with no options which, when STDOUT is to a terminal, generates a list of 160 random eight-character passwords using uppercase, lowercase, and numeric characters. I can choose one of the passwords from the list to use.

```
[student@studentvm1 ~]$ pwgen
Iiqu4ahY Eeshu1ei raeZoo8o ahj6Sei3 Moo5ohTu ieGh6eit IsOEisae eiVo5Ohv
Gooqu5ji ieX9VoN5 aiy3kiSo Iphaex4e Vait1thu oi5ruaPh eL7Mohch iel2Aih6
Elu5Fiqu eeZ4aeje Ienooj6v iFie2aiN ruu7ohSh foo4Chie Wai5Ap1N ohRae1lu
urahn2Oo eal6Zuey GuX3choO iesh1Oot eepha1Ai oe6Chaij ISaeb3ch OK7Iuchu
aeNgee6O Iequit9U OoNgi2oo cohY4Xei Ziengi3E quohTei4 eefe2ieC eong8Qui
Vo5aip8m EishiOei Xith9eil aongu4Ai paiFe1zo gaiPh5Ko Be7ieYu2 Fathah9h
Gu7UcePh lee7aiSh aj4AuChe Zo3caeR1 Yo8jei5x maeChe5a IdObaigh Fu4tei4e
geiLeid7 quaeK4Ro ohVoe5iZ AY2Noodi nemOtahJ ahPiw1oh gah6baeH Aa5pohCh
ahShai1h uQu3Hah1 Eth3coo5 EChoboc9 IeyOahCh Mee3iewu Iek6oMai aePoo2ei
aeVoM8Sh IeROhohr Duew9ogh toh8AeXu NohghOme ain4Ooph ooyuKoh1 huth1Mei
si4ohCao ahthaeOI ohquah5F chohpe9G yoiM2noh iePh9iej aij7uXu7 Phoophi8
Bei5iLah uR3aicer oagh2OVo uThox9Xa Gu4reeOv shohNe2a weReth7A Vae4ga3b
Jee9jieX kohjoR6o Zimaish2 ut9mahJ8 ephu8Ray IepOeiTh ooB3joom Rai1ohzu
emOEeruv Tu7Phoh1 bohOIFee roh6Phae tauT3ohh LieFiuOa Voo9uvah pahpuiJ1
ohSiaN9a ooBahnu9 Uo2DahSO oor6Huwe ahs6Och3 aeCai1oo ahw2Lawi oCaeboo8
oshahB8e Xu3iyohx NoX4ohCi oa5aiLih uLah7noo Thopie2a ua6iuQuo ooYab5ai
Gae5ahsh Eech1re7 feeDah4v wou7Oek4 iefoo9AJ zei4ahVi uMiel7sh jae3eiVo
zahC3Tue Eiphei6E ke6GiaJ8 oquieBaO chi8Ohba ooZ9OC3e deiV7pae sieCho6W
nu1oba1D aiYoh2oo OoluaZ7u Ahg5pee7 Teepha6E oochOMod ThaiPui5 Ehui9ioF
ekuina3Z Oafaivi1 Pusuef9g aChoh2Eb Cio7aebe eoPOiepu seGh2kie fiax4Cha
```

Pipe the data stream through the sort command. The result is a single password. Whenever the data stream of pwgen with no options is sent through a pipe, only a single password is generated. This is ideal for automation scripts.

```
[student@studentvm1 ~]$ pwgen | sort
Eaphui7K
```

This behavior can be overridden using the -N option to specify the desired number of passwords.

```
[student@studentvm1 ~]$ pwgen -N 6 | sort
boot6Ahr
Die2thah
nohSoh1T
reob9eiR
shahXoL6
Wai6aiph
```

Instead of using options, **pwgen** recognizes two arguments in the syntax, **pwgen pw_length number_of_passwds**.

```
[student@studentvm1 ~]$ pwgen 25 10
EetahchOhiuvaedodu1iPh5Oh Ahvoosoh5Eifei8eiyahWee1s Hout4ichoh9eiBeip5ChaiRe2
aGuuquaexiet8epao1phiOthu yuwoo3pei5nooQua7koo9kube wa6ahcho8Aey1ahthaegaeB9w
ahg5oo8xeivo6fahw6shila1C een9eeGOquoov3Iegheixahde hae6IeBe1eiZoh2laa9phivae
naengeiHohshaikahghie4aer
```

Now use the -s and -y options in various combinations.

```
[student@studentvm1 ~]$ pwgen -s
[student@studentvm1 ~]$ pwgen -y
[student@studentvm1 ~]$ pwgen -sy
[student@studentvm1 ~]$ pwgen -s 25 90
```

Read the man page for **pwgen** and check out some of its other options. Of particular interest is the option to remove ambiguous characters, those which can be confused for each other such as l (eye) and 1 (one) or 0 (zero) and O (oh).

## Password quality

There is a configuration file that is used to define the quality requirements of new passwords, /etc/security/pwquality.conf. The system defaults are defined to us humans as commented lines in this file. Lines that need to be changed to improve security can be uncommented and the default value for the variable changed to whatever is desired.

For one example, you might wish to change the default password length from eight to ten. So uncomment the `# minlen = 8` line and change 8 to 10.

One trick many users try when required to change their passwords is to simply change one character in the existing one. So they change the old password, password3, to password4. The default setting of the `difok = 1` allows this, but it is a bad idea.

## EXPERIMENT 16-7

This experiment will be performed partly as root and partly as the student1 user. First, read the man page for pwquality.conf to get an idea of the variables available that can be changed. Then as root, set the password for student1 to "password1234". Log in to the student1 account using that password.

As student1, change the password to "password9876".

Edit the /etc/security/pwquality.conf file and change the line

```
# difok = 1
```

to

```
difok = 5
```

Now try to change the password for student1 to "password4567". Note that a reboot is not required. These changes are activated immediately.

```
[student1@studentvm1 ~]$ passwd
Changing password for user student1.
Current password: password9876
New password: password4567
BAD PASSWORD: The password is too similar to the old one
passwd: Authentication token manipulation error
```

Another trick users try is to use all one class of characters for their passwords, such as all lowercase or all numbers. The `minclass = 0` can be changed to require as many as four classes of characters, uppercase, lowercase, numbers, and special characters. It is a good practice to require at least three different classes of characters.

Change `minclass` to 3 and try to change the password for student1 to one with one or two classes of characters. Then change the password to one with three classes. This setting does not specify which character classes are required.

Change the altered lines back to their original values and set the password for student1 to your preferred one.

# Managing user accounts

Managing user accounts is a very common task for system administrators in many environments. This can include adding new accounts, deleting accounts that are no longer needed, and modifying existing accounts.

There are at least three methods that can be used to create, delete, and modify user accounts. Most desktop environments have some form of GUI tool. There are command-line tools **useradd** and **adduser**; the latter is simply a symlink to **useradd** but is retained for backward compatibility as well as commands to delete and to modify user accounts. And there is also the very retro method of editing the required files by hand.

In the next few experiments, we will manage user accounts using the command line and by manually editing the various files. I liked editing these files by hand when I was first learning because it provided me with knowledge of what the commands actually do as well as all of the files required and their structures.

## Creating new accounts

Creating new accounts is easy no matter which method is used.

### The useradd command

We start with the easy method. The useradd command is flexible and can allow the SysAdmin to just take the defaults for things like password expiration, the default shell, and more.

**EXPERIMENT 16-8**

Perform this experiment as the root user.

Start by checking one of the files that defines some defaults for new users, /etc/defaults/useradd. These can be set permanently here or overridden at the command line.

```
[root@studentvm1 ~]# cat /etc/default/useradd
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
```

```
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

Now add a user using the default settings. The -c (comment) option adds the text in quotes to the GECOS comment field in /etc/passwd.

```
[root@studentvm1 ~]# useradd -c "Test User 1" tuser1
```

Set a password for this new user. Then look at the lines for this user in /etc/passwd, /etc/shadow, and /etc/group. Look at the contents of the home directory for tuser1, including hidden files.

Now let's create a user account with a different shell, zsh, and home directory location, /TestFS/tuser2. We added a number of different shells in Chapter 7 of Volume 1 so the Z Shell should already be installed. The -d option specifies the complete path of the home directory. If the base directory, in this case /TestFS, does not exist, neither it nor the home directory will be created. The -s option specifies the user's default shell.

```
[root@studentvm1 etc]# useradd -c "Test User 2" -d /TestFS/tuser2 -s /usr/
bin/zsh tuser2
```

Set the password for this user. Check the lines that were added to the passwd, shadow, and group files. Log in or **su -** as this user to verify that zsh is the default shell. The command prompt will be a bit different. Notice the difference in command-line editing and the lack of tab completion. Log out from this user account.

Let's add a user with a password expiration date of today. The -e option specifies the expiration date in YYYY-MM-DD format and does the conversion to days elapsed since Jan 1, 1970, 00:00 UTC. Note that "today" for you will be different from "today" as I write this, so be sure to use your today.

```
[root@studentvm1 etc]# useradd -c "Test User 3" -e 2019-05-28  tuser3
```

Set a password for tuser3. As the student user, use **su** to log in as tuser3 from a terminal session. You could also use a virtual console to log in as tuser3.

```
[student@studentvm1 ~]$ su - tuser3
Password: <Enter password for tuser3>
Your account has expired; please contact your system administrator
su: User account has expired
```

Now let's use the **usermod** command to set the expiration date to a few days in the future – your future.

```
[root@studentvm1 etc]# usermod -e 2019-06-05 tuser3
```

As the student user again, use **su** to log in as tuser3.

```
[student@studentvm1 ~]$ su - tuser3
Password: <Enter password for tuser3>
Running /etc/profile
Running /etc/profile.d/myBashConfig.sh
Running /etc/bashrc
Running /etc/bashrc
[student@studentvm1 ~]$
```

Setting the account expiration date is not tied to the password expiration date. For example, you might hire a contractor to work on a project that will take 6 months. So you can set the account expiration date to 6 months, while the password expiration is set to 30 days. The password will expire every 30 days during that 6 months, but the account will expire at the specified future date. Even if the password has not expired, the user will not be able to log in to an expired account.

Log out of the tuser3 session.

Although the password expiration date can be set using the -K option of the **useradd** command, these should be system-wide settings and belong in the /etc/login.defs file to ensure consistency in its application.

It is possible to set an initial password, already encrypted, while adding a new account. We can use the **openssl** command to encrypt the password into a string called a hash, then use the result as an option in the **useradd** command.

Now we get to adding a new user with an encrypted password.

## EXPERIMENT 16-9

Perform this experiment as the root user. Add a new user using a password hash to set the password with the **useradd** command. Enter this command all on one line. Be sure to place the back tics (`) around the **openssl passwd** command as shown.

```
[root@studentvm1 ~]# useradd -c "Test User 4" -p `openssl passwd -salt 123456
-6 mypassword` tuser4
```

Now look at the shadow file and verify that it looks like this. Because we used the same salt and the same password to generate the password hash, the resulting hash should be identical.

```
tuser4:$6$123456$KKcK3jDXxN5TVYNLbMdEIjnfRjaSlbqj5X9bBgryaa4qLDO4lrM9ksw
CpAZL27/WXlbsDQcJ8kBxPjcpips781:18044:0:99999:7:::
```

Now log in as tuser4 using the password "mypassword" to verify that this has produced the desired result. Log out as tuser4.

## Creating new accounts by editing the files

Although it is a bit more involved and takes a little more time than using the useradd command, editing the files to add a new user can be a helpful learning experience. It is really not all that difficult. It just takes time and knowledge.

Although we would never expect to add a new user in this way, I have had the need to repair a user account. Knowing all of the files involved and how to edit them safely has been useful to me. So this experiment is more about gaining a knowledge of the files that comprise a user account rather than a need to ever add a new user using this method.

### EXPERIMENT 16-10

This experiment must be performed as the root user. We will add a new user, Test User 5, tuser5, by editing the appropriate configuration files and creating a home directory.

Start by adding the following line to the /etc/passwd file. I just used the Vim editor to copy the line for tuser4 and made the appropriate changes. The account for tuser4 is UID and GID 1006 so the UID and GID for tuser5 will be 1007.

```
tuser5:x:1007:1007:Test User 5:/home/tuser5:/bin/bash
```

Edit the /etc/group file by adding the following line. Once again I just copied the line for tuser4 and made the needed changes.

```
tuser5:x:1007:
```

Edit the /etc/shadow file and add the following line. Again, I just copied the line for tuser4 and changed the user account name to tuser5. This results in tuser5 having the same password as tuser4.

```
tuser5:$6$123456$KKcK3jDXxN5TVYNLbMdEIjnfRjaSlbqj5X9bBgryaa4qLDO4lrM9ksw
CpAZL27/WXlbsDQcJ8kBxPjcpips781:18044:0:99999:7:::
```

Note that when you attempt to save this change, Vim displays an error, "E45: 'readonly' option is set (add ! to override)". So instead of simply using :wq, you need to add an exclamation mark (!) to force Vim to save the changes, **:wq!**.

Create the new home directory and set its permissions to 700.

```
[root@studentvm1 home]# cd /home ; mkdir tuser5 ; ll
total 52
drwxrws---   2 root     dev       4096 Apr  2 09:19 dev
drwx------.  2 root     root     16384 Dec 22 11:01 lost+found
drwxrws---   2 root     dev       4096 Apr  2 12:34 shared
drwx------. 27 student  student   4096 May 28 16:09 student
drwx------   4 student1 student1  4096 Apr  2 09:20 student1
drwx------   3 student2 student2  4096 Apr  1 10:41 student2
drwx------   3 tuser1   tuser1    4096 May 27 21:48 tuser1
drwx------   3 tuser3   tuser3    4096 May 28 08:27 tuser3
drwx------   3 tuser4   tuser4    4096 May 28 17:35 tuser4
drwxr-xr-x   2 root     root      4096 May 28 21:50 tuser5
[root@studentvm1 home]# chmod 700 tuser5 ; ll
total 60
<snip>
drwx------   2 root     root     12288 May 28 21:53 tuser5
```

Copy the files from /etc/skel into the new home directory.

```
[root@studentvm1 home]# cp -r /etc/skel/.[a-z]* /home/tuser5 ; ll -a tuser5
total 40
drwx------   3 root root 12288 May 29 07:45 .
drwxr-xr-x. 12 root root  4096 May 28 21:50 ..
-rw-r--r--   1 root root    18 May 29 07:45 .bash_logout
-rw-r--r--   1 root root   141 May 29 07:45 .bash_profile
-rw-r--r--   1 root root   376 May 29 07:45 .bashrc
-rw-r--r--   1 root root   172 May 29 07:45 .kshrc
```

```
drwxr-xr-x   4 root root  4096 May 29 07:45 .mozilla
-rw-r--r--   1 root root   658 May 29 07:45 .zshrc
```

Set the ownership of the directory and its contents to the new user.

```
[root@studentvm1 home]# chown -R tuser5.tuser5 tuser5 ; ll -a tuser5
total 32
drwx------   3 tuser5 tuser5 4096 May 29 07:51 .
drwxr-xr-x. 12 root   root   4096 May 29 07:50 ..
-rw-r--r--   1 tuser5 tuser5   18 May 29 07:51 .bash_logout
-rw-r--r--   1 tuser5 tuser5  141 May 29 07:51 .bash_profile
-rw-r--r--   1 tuser5 tuser5  376 May 29 07:51 .bashrc
-rw-r--r--   1 tuser5 tuser5  172 May 29 07:51 .kshrc
drwxr-xr-x   4 tuser5 tuser5 4096 May 29 07:51 .mozilla
-rw-r--r--   1 tuser5 tuser5  658 May 29 07:51 .zshrc
```

To test this new user with a login, you can use either a virtual console or use **su - tuser5** from a student user terminal session. Or do both. Remember that tuser5 has the same password as tuser4, mypassword.

```
[student@studentvm1 ~]$ su - tuser5
Password: mypassword
Running /etc/profile
Running /etc/profile.d/myBashConfig.sh
Running /etc/bashrc
Running /etc/bashrc
[tuser5@studentvm1 ~]$
```

That was fairly easy, but using the useradd command is even easier most of the time. Log out of the tuser5 account.

---

You have now performed all of the steps necessary and modified all of the files involved in creating a new user. This knowledge has been useful in my SysAdmin work. I have sometimes found it easier to simply edit a group or passwd file than to use an appropriate CLI command like **usermod**. It is also easier to spot a damaged entry in one of these files having edited them by hand.

## Locking the password

Is one of your users going on extended vacation? Or is a user leaving the organization or transferring to another location? Sometimes you want to secure an account so that no one – except for you, the SysAdmin, as root – has access, but you do not want to delete it. The need to do this could also be for forensic reasons.

A locked password prevents the user – or anyone else – from logging into the account, but it does not change the password hash. The password can be easily unlocked when necessary. The user can then log in with the original password.

**EXPERIMENT 16-11**

Parts of this experiment will be performed as root and parts as tuser5. Be sure to log out of the tuser5 account. Now lock the tuser5 account. The -l (lowercase ell) option locks the account.

```
[root@studentvm1 ~]# passwd -l tuser5
Locking password for user tuser5.
passwd: Success
```

Look at the line for tuser5 in the /etc/shadow file.

```
[root@studentvm1 ~]# grep tuser5 /etc/shadow
tuser5:!!$6$123456$KKcK3jDXxN5TVYNLbMdEIjnfRjaSlbqj5X9bBgryaa4qLDO4lrM9
kswCpAZL27/WXlbsDQcJ8kBxPjcpips781:18044:0:99999:7:::
```

Notice the two exclamation points (!!) at the beginning of the password hash. Now log in to the tuser5 account from a virtual console or using **su** from a student account terminal session.

```
[student@studentvm1 ~]$ su - tuser5
Password:
su: Authentication failure
```

Now unlock the account.

```
[root@studentvm1 ~]# passwd -u tuser5
Unlocking password for user tuser5.
passwd: Success
```

Log in to it again to verify that it is unlocked.

We could also have used Vim to edit the /etc/shadow file to add and remove the "!!" from the beginning of the password field.

## Deleting user accounts

User accounts are very easy to delete. Depending upon the reason for deletion, you may wish to retain the user's home directory or to delete it along with the rest of the account.

### EXPERIMENT 16-12

Perform this experiment as the root user. Use the **userdel** command to delete tuser3 but leave the home directory.

```
[root@studentvm1 ~]# userdel tuser3 ; ll /home
total 52
drwxrws---   2 root     dev       4096 Apr  2 09:19 dev
drwx------.  2 root     root     16384 Dec 22 11:01 lost+found
drwxrws---   2 root     dev       4096 Apr  2 12:34 shared
drwx------. 27 student  student   4096 May 28 16:09 student
drwx------   4 student1 student1  4096 Apr  2 09:20 student1
drwx------   3 student2 student2  4096 Apr  1 10:41 student2
drwx------   3 tuser1   tuser1    4096 May 27 21:48 tuser1
drwx------   3     1005     1005  4096 May 29 07:53 tuser3
drwx------   3 tuser4   tuser4    4096 May 28 17:35 tuser4
drwx------   3 tuser5   tuser5    4096 May 29 07:53 tuser5
```

Because the tuser3 account no longer exists, the account name is no longer shown in the long listing of the /home directory. The directory and its contents still have UID and GID of 1005 and that is what we see here.

Now delete the account for tuser4 along with its home directory. The -r option removes the home directory for an account.

```
[root@studentvm1 ~]# userdel -r tuser4 ; ll /home
total 48
drwxrws---   2 root     dev       4096 Apr  2 09:19 dev
drwx------.  2 root     root     16384 Dec 22 11:01 lost+found
drwxrws---   2 root     dev       4096 Apr  2 12:34 shared
drwx------. 27 student  student   4096 May 28 16:09 student
drwx------   4 student1 student1  4096 Apr  2 09:20 student1
drwx------   3 student2 student2  4096 Apr  1 10:41 student2
drwx------   3 tuser1   tuser1    4096 May 27 21:48 tuser1
```

```
drwx------   3     1005     1005  4096 May 29 07:53 tuser3
drwx------   3 tuser5   tuser5    4096 May 29 07:53 tuser5
```

It is also possible to delete user accounts by editing the appropriate files to remove the lines pertaining to the account we want to remove.

# Forcing account logoff

Sometimes it may become necessary to force a user account to log off. This may be required because a user has left for the day and the system needs to be updated with no users logged in, the user has left the organization, or there may be one or more runaway processes.

Whatever the reason, there is no "logout other user" command, but there is a simple way to deal with this need to force the termination of all processes belonging to a user. This is, in fact, even more effective than simply forcing a logout.

**EXPERIMENT 16-13**

This experiment will be performed as root and the student1 user. First let's see if the student1 user is logged in. The **pgrep** command can be used as root to locate the PIDs of running processes that belong to a specific user account.

```
[root@studentvm1 ~]# pgrep -U student1
[root@studentvm1 ~]#
```

In this case the student user is not logged in. If you find that there are no processes belonging to the student1 user or you find only three or four processes, open a terminal session on your desktop and su to the student1 user.

```
[student@studentvm1 ~]$ su - student1
Password:
Running /etc/profile
Running /etc/profile.d/myBashConfig.sh
Running /etc/bashrc
Running /etc/bashrc
[student1@studentvm1 ~]$
```

Also log in at a virtual console as student1. As root, look for processes belonging to student1. The specific PIDs you see will be different from those listed here from my VM.

```
[root@studentvm1 ~]# pgrep -U student1
30774
30775
30780
30781
30785
30831
30840
```

Now we can kill all of these processes to force the user off the system. This is not a logout; it kills all of the user's processes, even ones that might be left behind after a logout.

```
[root@studentvm1 ~]# pkill -KILL -U student1
[root@studentvm1 ~]# pgrep -U student1
[root@studentvm1 ~]#
```

This will also show "Killed" in the terminal window you had opened.

## Setting resource limits

You might think that with huge amounts of disk space, high CPU counts and speeds, gigabytes of RAM, and terabytes of disk space that modern Linux hosts do not have limited resources. That is not true, and I have seen huge systems overburdened by programs in test or even fully released programs that were poorly written. More frequently, I have caused myself problems where a shell script I have written sucks up most or all of some critical resource. You saw how easy that was when we did it intentionally earlier in this course. It seems to be even easier when it is unintentional.

So imagine for a moment that you are the system administrator for a Linux box that has a large number of users. And that one of those user accounts is running resource hogs – large jobs that suck up CPU time and memory which slows the tasks being run by other users to a crawl. They all come complaining to you. What do you do?

These issues can be corrected immediately by killing all of the processes – or at least the offending ones – belonging to the user owning them.

It is even better to prevent these problems by setting limits on the amount of system resources that a user, users, or groups can consume. The /etc/security/limits.conf file can be used to set limits for system resources to levels that will affect other users not at all or at least only marginally. A related option is to add a local config file to the /etc/security/limits.d directory. This prevents making changes to the main configuration file such as might occur during updates or complete reinstallations.

There are soft and hard limits that can be set. A soft limit sends a message when it is met or exceeded, while a hard limit prevents completion of the command that caused it. Both hard and soft limits can be set on a resource so that the user first receives a message and then is prevented from continuing. The limits set in this file are monitored and enforced by the pluggable authentication module (PAM) system which we will encounter again in Chapter 17 of this volume.

Limits can be applied to individual users, to groups such as dev or accounting, and to lists of users or groups, as well as to all users or groups.

## EXPERIMENT 16-14

This experiment is performed in part as the root user and in part as the student1 user.

First, as the root user, open /etc/security/limits.conf with **less** and just view its contents for now. Read the comment sections of this file to understand the system resources that can be restricted using this file and its overall structure as well as the syntax used in creating a limit specification. It is fine to leave this file open for reference.

Some of the configurable resources need a bit additional clarification. So, Figure 16-4 lists all of the resources over which we can exert some control and the descriptions as found in the limits.conf file. In some cases I have added comments of my own.

| Resource | Description | Comments |
|---|---|---|
| core | Limits the core file size (KB) | This refers to the core dump files in the event of a kernel or other system error. |
| data | max data size (KB) | |
| fsize | maximum filesize (KB) | |
| memlock | max locked-in-memory address space (KB) | This type of memory is locked into RAM and cannot be swapped out. This limits the maximum amount of RAM that can be locked into RAM by a user's processes. |
| nofile | max number of open file descriptors | Each open file has a file descriptor. Effectively limits the total number of open files. |
| rss | max resident set size (KB) | The maximum virtual memory, RAM + swap, that can be allocated. |
| stack | max stack size (KB) | The stack is a temporary storage location for programs to store some types of data during operation. This limits the total stack space available to the user or group. |
| cpu | max CPU time (MIN) | The maximum amount of CPU time in minutes per day that can be allocated to a user. NO additional CPU time can be allotted to the user after a hard limit is reached so they would be done for the day. |
| nproc | max number of processes | |
| as | address space limit (KB) | The total amount of address space that can be allocated including, program, data, and stack. |
| maxlogins | max number of logins for this user | Limits the total number of concurrent logins by individual users. |
| maxsyslogins | max number of logins on the system | Limits the total number of concurrent logins on the system as a whole. |
| priority | the priority to run user process with | Allows setting low (or high) priorities on user processes. Usually used to lower priorities for users that do not run critical processes, thus allowing the more critical users to have more CPU time if it is required. |

***Figure 16-4.*** *The system resources that can be controlled with the limits.conf file*

| Resource | Description | Comments |
|---|---|---|
| locks | max number of file locks the user can hold | File locks allow multiple users to access a single file while preventing all but one – the one with the lock – from altering the file. This limits the number of files a user can have open for modification. |
| sigpending | max number of pending signals | We looked at signals in Chapter 23. It is possible to use Ctrl-C to exit from a process ( or other signals) multiple times but if the process is not responding, the signals are queued. This limits the size of that queue. |
| msgqueue | max memory used by POSIX message queues (bytes) | |
| nice | max nice priority allowed to raise to values: [-20, 19] | |
| rtprio | max realtime priority | |

***Figure 16-4.*** *(continued)*

Rather than change the default file, we will add a new file local.conf to the /etc/security/limits.d/ directory. Create the local.conf file there and open it with Vim. Add the following line to set the number of logins for student1 to 3. The settings in this local file override the default settings.

```
student1        -       maxlogins       3
```

A reboot is not required. However, if a user already has the specified number or more of logins already open, the existing logins are not affected but no additional logins will be allowed.

Use virtual consoles 2 through 5 to log in as the student1 user. If you have those consoles logged in as another user, log out and then log in as student1. When you try to log in on VC5, you will get the following error indicating that the number of logins has been exceeded.

```
student1@studentvm1's password:
Too many logins for 'student1'.
Permission denied
```

Look at the number of logins.

```
[root@studentvm1 limits.d]# w -u student1
 14:53:19 up 2 days, 22:12, 13 users,  load average: 0.04, 0.03, 0.01
```

```
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
student1 tty2      14:34   19:01   0.02s  0.02s -bash
student1 tty3      14:34   18:47   0.02s  0.02s -bash
student1 tty4      14:34   18:36   0.01s  0.01s -bash
```

This is the maximum number of logins allowed for this user.

Resource limitations must be considered with care, especially when applying them to groups and multiple users. This is an excellent way to ensure that the desires of the few do not outweigh the needs to the many.

# Chapter summary

This chapter has taken us through the processes of creation, modification, and deletion of user accounts. We have also explored the various configuration files that support user accounts and their creation. We have created user accounts using the command-line tools and have also used Vim to add a new user by direct editing of the passwd, group, and shadow files.

We have also looked at user account security in general and password security in particular. This included exploring configuration tools that can force users to change their passwords on a regular basis as well as to enforce some aspects of password security policy.

We have also explored multiple methods to deal with resource allocation problems and how to kick users off the system who are abusing resources.

# Exercises

Perform these exercises to complete this chapter:

1. What are the file modes for /etc/passwd and /etc/shadow?
2. Why are the user passwords stored in /etc/shadow?
3. Why might a SysAdmin want to generate a password using the pwgen command and pipe it directly to the useradd command, thus creating an initial password that is not known to anyone?

4. The command **`cp -r /etc/skel/.[a-z]* /home/tuser5 ; ll -a tuser5`** was used to copy the files from /etc/skel to the home directory of tuser5 in Experiment 16-9. Why is the syntax **`/etc/skel/.[a-z]*`** used instead of just **`/etc/skel/.*`**?
5. Remove the tuser5 account by editing the user account files. Also remove the home directory for the account.
6. The tuser3 account was removed in Experiment 16-12 without also removing the account's home directory. Remove that home directory.
7. Set a hard limit of 5 minutes of CPU time on a user other than the student user. You could use the student1 user. Then start a cpuHog program and watch what happens.
8. Devise and run an experiment to verify that setting the number of allowed logins on a user to a number smaller than the current number that are open does not affect the open ones.
9. Set the shell on the student1 account as /sbin/nologin. What happens when you try to log in as that user?

CHAPTER 17

# Security

## Objectives

In this chapter you will learn

- To define the full scope of Linux security
- To identify and implement additional aspects of password security
- To use the firewalld firewall to manage access to the host on SSH and Telnet ports
- To install and configure the Telnet server
- To use tcpdump to monitor the plain text data of a Telnet conversation
- To enable the SSHD server and use tcpdump to verify that the conversation is encrypted
- To describe PAM authentication
- To install and configure Fail2Ban dynamic firewall software to block attacks
- To take some basic steps to improve security of any Linux host

## Introduction

Linux is a very secure operating system. It provides a secure environment in which to work and store files. However, good security, by its very nature, can be a bit obtrusive. We have already had discussions about security in many sections of this course. This is because security must not be an afterthought; it must be an integral component of everything we do as SysAdmins. The security of our systems and the data on them must be a major consideration if not the prime consideration in everything we do.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_17

Any device connected to the Internet is subject to attack and no operating system, even Linux, is completely free of exploitable flaws. Linux just happens to be more secure than other operating systems and, when it is breached, is less vulnerable to widespread damage.

There are only four rules required to achieve complete and unbreakable security:

1. The computer must be locked in a blast-proof room to prevent both unauthorized access and destruction of the computer and its contents.
2. It must be inside a Faraday cage to prevent its own radio frequency emissions from escaping to be captured by "The Bad People." This would also protect the host against an EMP blast but assumes a power source completely within the confines of the cage.
3. It must have a 100% air gap – that is, it must not be connected to any network and especially not to the Internet. This includes all hard wired and wireless connections such as wireless, Bluetooth, infrared, and anything else that transmits data outside of the computer.
4. It must be turned off and the rest of the rules don't count.

Unfortunately, these rules mean the computer is unusable even for its intended purpose. That, in turn, means that any computer that is turned on is vulnerable to cracking. A cracker is the correct name for a hacker with evil intent. Hackers – the good people who hack, that is, work on, hardware and code – are the ones who gave us things like free and open source tools like Linux and the many applications that run on it. So we must use our computers in an imperfect and unsafe environment at all times.

Security is not an afterthought that is appended to the end of a book – it is something that must be considered as part of all we do as SysAdmins. We have already enabled and used various types of security throughout this book. In this chapter we will look at some additional security considerations.

## Security by obscurity

Most computers are connected to the Internet through first a wireless or wired router and then the router/modem supplied by the Internet Service Provider, or ISP. This provides a couple layers of obscurity between the computer and the Internet. You may also think that your Small Office/Home Office (SOHO) or other small organization or even your personal home computers are too small and insignificant to attract the attention of the crackers. You are counting on your relative obscurity to protect you from the crackers because there are surely more lucrative and much larger and important targets available.

***NOT!***

"Security by obscurity" cannot be counted on to protect your computer. In fact, the worst assumption you can make is that some level of obscurity can protect your computers. Some small businesses I have worked with, including my own systems, are constantly subjected to attempts to crack into their firewall servers, hundreds of attempts per day and thousands per month. And that is *after* I have instituted measures to reduce the total number of attacks. Every computer that is connected to any network – and especially the Internet – is a target.

There are a number of measures that can be instituted to protect your computer, but it will never be impervious. And you must help by exercising care in the way you use the Web and deal with spam email.

There are a number of good web sites with information on how to protect yourself online. Although I cannot verify all of the information on them, there is one I especially like Get Safe Online.[1] It even has a section on safe Linux use.

## What is security?

Security is about far more than simply preventing unwanted people from logging in to our Linux computers. Although good passwords and other security measures are helpful and can help to prevent that type of security vulnerability, they are the response to only one part of the security problem. There are many aspects to security, and it is important to understand that fact as well as to know the things that we are using various security protocols to protect.

[1]Get Safe Online, www.getsafeonline.org/protecting-yourself/

So what are the things we are trying to protect? Not surprisingly, a large part of security is designed to protect our data but perhaps in ways and for reasons you have not previously considered.

## Data protection

There are three major considerations to data protection and different tools and strategies applied to each.

First we want to protect our data from loss in the sense that it needs to be available to us. We need to be able to have access to it. So this is about ensuring that the data will not be destroyed or lost so that it is no longer available. As a business owner, loss of my financial records in a fire or natural disaster would be a disaster for my company and that might be impossible to recover from. This is about the accessibility of our data in order to ensure the continuity of our business.

Second we want to protect our data from unauthorized access. We need to ensure that our company and personal data is not available to someone who might use it to steal our identity or to steal money from us. In the case of many organizations, that data might contain information that a competitor might use to gain advantage over us. This is about the confidentiality of our data.

Third, we want to ensure that our data is safe from unauthorized changes or corruption, perhaps by malware or a disgruntled employee or one who is simply a thief. And we also want to ensure that we are not blocked from access to our own data by malware such as ransomware which would encrypt the data and keep us from accessing it until the ransom is paid. This is about ensuring the integrity of our data.

It's not just about keeping the data safe, but knowing for a fact that the data was kept safe from espionage/destruction/corruption.

## Security vectors

Security attack and danger vectors are many and varied. These vectors are all classifiable into five major categories, self-inflicted, environmental, physical, network, and software vulnerabilities. Although we will list a number of common security vulnerabilities of different types, I have no intention of exploring all the responses to these problems here. There are some things that can be done, in addition to some of the obvious ones hinted at in this section, and we will explore those later in this chapter.

In case you have not figured it out yet, this is where I try to scare you enough to apply some common sense and freely available open source tools to improving the security of your Linux systems.

## Self-inflicted problems

Self-inflicted data loss comes in many forms. The most common form is the semi-intentional erasure of one or more important files or directories.

Sometimes erasing needed files is accidental. I just erase a bunch of old files in a directory, and it turns out later that one or two are still needed. More often, for me at least, I actually look at the files and decide they are no longer needed. A day, or two, or a week after I delete them, it turns out that I still need at least some of the files I just deleted. I have also made significant changes to a file and saved it. Once again I find at some time later I made changes and especially deletions that I should not have. Clearly it is necessary to pay attention when deleting files or making changes to them. That still won't keep us from deleting data we may need later.

On other occasions I have been working on the back side of a computer rack and accidentally pulled the power plug – or plugs if there were redundant power supplies – from the wrong computer. Although hard drives and journaling filesystems can generally withstand a power loss, it still happens. In a somewhat less stringent environment than a data center, I have managed to kick a power cable out of the wall.

This category also includes things like using poor passwords that can be easily cracked and leaving a USB drive with critical data stored on it in an accessible location. Leaving a laptop unattended in a public place like a coffee shop or using unencrypted wireless links are also common points of data loss.

## Environmental problems

Environmental issues that can affect the security of computers systems are usually not considered as potential problems or at least misunderstood by most people. When we use the word environmental, we tend to think in terms of electrical power backup units, cooling the data center so the computers will be cool, and so on. But there is so much more that many of us don't think about. I was fortunate that I learned about environmental issues early in my career at IBM.

**Power failures** can occur for many reasons. This includes momentary power failures that can shut down the computer just as irrevocably as longer ones. Regardless of the reason for the power failure, there is the danger of losing data especially from documents that have not been saved. Modern hard drives and filesystems employ strategies that help to minimize the probabilities of data loss, but it still happens.

**Grounding** – actually the lack thereof or improper grounding – can be a serious issue. Good grounding is essential for the proper electronic operation and stability of computers.

**Electromagnetic interference, EMI**, is various types of electromagnetic radiation from many different sources. This radiation can interfere with the correct operation of any electronic device including computers. Lightning, static electricity, microwaves, old CRT displays, military radar systems, and radio frequency bursts on a ground line all of these and more can cause problems. Good grounding can reduce the effects of all of these types of EMI. But that does not make our computers completely immune to the effects of strong EMI fields.

**Hard drive failures** also cause data loss. The most common failures in today's computers are devices that have moving mechanical components. Leading the frequency list are cooling fans and hard drives are a close second. Modern hard drives have SMART capabilities that enable predictive failure analysis. Linux can monitor these drives and send an email to root indicating that failure is imminent. Do not ignore those emails because replacing a hard drive before it fails is less trouble than replacing one after it fails and then hoping the backups are up to date.

Modern computers are well protected against many aspects of environmental problems. All we need to do is to ensure that we use battery backup units, which are also known as Uninterruptible Power Supplies (UPS), and that they are plugged into properly grounded outlets. Things can still happen, but this will minimize the possibilities.

## Physical attacks

Physical security is about protecting the hardware from various types of harm. Although we tend to think in terms of keeping bad people away from the hardware on which we run our systems, we also need to consider disaster scenarios as a major part of our planning.

**Disgruntled employees** can maliciously destroy data. Proper security procedures can mitigate this type of threat, but backups are still handy.

**Common theft** is also a way to lose data. Soon after we moved to Raleigh, NC, in 1993, there was a series of articles in the local paper and TV that covered the tribulations of a scientist at one of our better known universities. This scientist kept all of his data on a single computer. He did have a backup – to another hard drive on that same computer. When the computer was stolen from his office, all of his experimental data went missing as well and it was never recovered.

**Natural disasters** occur. Fire, flood, hurricanes, tornadoes, mud slides, tsunamis, and so many more kinds of disasters can destroy computers and locally stored backups as well. I can guarantee that, even if I have a good backup, I will never take the time during a fire, tornado, or natural disaster that places me in imminent danger to save the backups.

Back up everything – frequently. And keep the most recent backups someplace off-site. I store my backups in a safe deposit box. If my home office is devastated by a disaster, I can rebuild from my backups.

## Network attacks

Attacks via networks, both local and the Internet, are common and can be extremely dangerous. These attacks can take many forms ranging from direct attacks from the Internet against firewalls and servers to indirect attacks in which malware is introduced into a host by some stealthy means such as hiding in a downloaded file, an email, or a click-bait link on a web site.

This type of attack does not require direct physical access to your computers. Rather, they come through your connections to the outside world.

**Scripted attacks** are generally used by so-called script kiddies. This derogatory term is used because they are not usually smart or determined enough to create the attack scripts themselves so they download them from those who are. The scripts they use are simple brute force remote login attempts. Their malicious attacks are useless against today's well-protected Linux hosts because most distributions are well hardened at installation and do not have the SSH server up and running so this type of attack is fruitless when SSH is not available for a connection.

These attacks usually consist of automated dictionary probes against a large number of remote hosts, usually those on a specific network range, rather than against a specific single host or organization.

**Malware** is a very generic term for software that can be used for various malicious purposes including destroying or deleting your data.

**Ransomware** is a specific form of malware that encrypts your data and holds it for ransom. If you pay the ransom, you may get the key that will allow you to decrypt your data – if you are lucky.

**Drive-by malware** is a malicious link in an apparently innocuous advertisement on an otherwise legitimate web page. You do not even need to click this link for your computer to be affected.

**Targeted login attempts** are aimed directly at your organization. These are like script kiddie attacks but with you as the main target. These attacks are usually carried out by someone or some group with a specific reason to target you. If someone targets you specifically and really wants to crack into your system, they will be able to do so, given enough time and even just a little bit of carelessness on your part.

Always keep systems up to date so that the latest security patches are installed. Ensure that good firewalls are in place and properly configured. And check frequently for evidence of break-ins.

## Software vulnerabilities

Many attacks on connected computers are aided and abetted by vulnerabilities in the host's software. These vulnerabilities are exploited by the attackers and can be leveraged to install malware of various types. However, just because a vulnerability exists does not mean that an exploit is available to take advantage of it.

Always install the latest updates to ensure that the software is as secure as possible.

# Linux and security

So – did that scare you? It should have.

But the good news is that Linux in general is very secure and so is Fedora especially when SELinux is set to enforcing. Linux is very secure immediately upon installation. There are only a couple minor services running that do not need to run, but none provide external access from the Internet. The remaining ones can be easily turned off. Unneeded services can be an access point for a cracker. In Chapter 13 of this volume, we turned off the pcscd service for this very reason.

Fedora has an excellent firewall in place and the one service I use in all of my Linux hosts, the SSH server for secure logins to and from remote hosts is configured only for outbound connections. The inbound SSHD server is disabled.

# Login security

Login security – ensuring that only authorized users have access to log in and use the system's resources – is the first line of defense. Generating and using secure passwords is the main tool we have to provide this security, whether a local or remote login. But it seems a bit silly for me to write an entire section on password security when the passwd(1) man page already has an excellent section on just that. So here it is, directly from the passwd man page.

"**Remember the following two principles**

**Protect your password.**

Don't write down your password - memorize it. In particular, don't write it down and leave it anywhere, and don't place it in an unencrypted file! Use unrelated passwords for systems controlled by different organizations. Don't give or share your password, in particular to someone claiming to be from computer support or a vendor. Don't let anyone watch you enter your password. Don't enter your password to a computer you don't trust or if things "look funny"; someone may be trying to hijack your password. Use the password for a limited time and change it periodically.

**Choose a hard-to-guess password.**

passwd, through the calls to the pam_cracklib PAM module, will try to prevent you from choosing a really bad password, but it isn't foolproof; create your password wisely. Don't use something you'd find in a dictionary (in any language or jargon). Don't use a name (including that of a spouse, parent, child, pet, fantasy character, famous person, and location) or any variation of your personal or account name. Don't use accessible information about you (such as your phone number, license plate, or social security number) or your environment. Don't use a birthday or a simple pattern (such as "qwerty", "abc", or "aaa"). Don't use any of those backward, followed by a digit, or preceded by a digit. Instead, use a mixture of upper- and lowercase letters, as well as digits or punctuation. When choosing a new password, make sure it's unrelated to any previous password. Use long passwords

```
(say at least eight characters long). You might use a word pair with
punctuation inserted, a passphrase (an understandable sequence of
words), or the first letter of each word in a passphrase."
```

We explored setting passwords and generating good ones in Chapter 16 of this volume, along with locking accounts to prevent any login access at all.

# Checking logins

Another tool we have is the list of user logins. This is made easy with the **last** and **lastb** commands which prevent us from having to scan the /var/log/secure log files. The **last** command displays all successful logins.

The **lastb** command displays a list of failed logins. These can be your own or other users' failed logins due to fumble fingers as I sometimes have, or it could be the result of an attack on your system.

## EXPERIMENT 17-1

Perform this experiment as the root user. The **last** command can be run by a non-root user, but **lastb** cannot.

Let's start with a view of successful logins. I have piped the result through the **head** command because this list can be very long and **head** truncates the data stream for me. You can do this both with and without piping it through **head**. The specifics of your data will be different from mine, but it should look very much similar to this.

```
[root@studentvm1 ~]# last | less
student  pts/4        192.168.0.1      Tue Jun  4 07:59   still logged in
root     pts/0        192.168.0.1      Mon Jun  3 21:37   still logged in
reboot   system boot  5.0.7-200.fc29.x Mon Jun  3 17:36   still running
student  pts/4        :pts/3:S.0       Sun Jun  2 14:43 - 21:35 (1+06:52)
student  pts/3        192.168.0.1      Sun Jun  2 14:42 - 21:35 (1+06:52)
root     pts/0        192.168.0.1      Sun Jun  2 14:41 - 21:35 (1+06:53)
reboot   system boot  5.0.7-200.fc29.x Sun Jun  2 10:26 - 21:35 (1+11:09)
root     pts/0        192.168.0.1      Sat Jun  1 14:06 - 14:06  (00:00)
reboot   system boot  5.0.7-200.fc29.x Fri May 31 17:20 - 14:06  (20:45)
student1 pts/10       192.168.0.1      Thu May 30 14:39 - 14:39  (00:00)
<snip>
```

```
You have new mail in /var/spool/mail/root
[root@studentvm1 ~]#
```

Here we can see a number of reboots, the student and root logins, and whether they are still logged in or not. The login times and durations are also recorded here. We also looked previously at ways to list all of the system boots using **journalctl**.

```
[root@studentvm1 etc]# journalctl --list-boots
```

This information can be useful in a forensic investigation into who was logged in at the time a specific file was changed or accessed, or some specific event took place. Really experienced crackers can cover their tracks, but this information could still have some value, especially if you are looking for someone who is not a pro.

So now let's look at failed logins. You may not have had any so this result will be empty as it is for my instance of StudentVM1.

```
[root@studentvm1 ~]# lastb

btmp begins Tue Jun  4 08:27:54 2019
```

So let's create a few bad logins. Using virtual console 2, try to log in as the user root, jhgd, !@#$%^, news, chrony, rpcuser, student, james, henry, and alice. Be sure to use completely random and bogus passwords. Look again at the list of failed logins.

```
[root@studentvm1 ~]# lastb
(unknown tty2                         Tue Jun  4 08:53 - 08:53  (00:00)
(unknown tty2                         Tue Jun  4 08:53 - 08:53  (00:00)
(unknown tty2                         Tue Jun  4 08:53 - 08:53  (00:00)
student  tty2                         Tue Jun  4 08:53 - 08:53  (00:00)
(unknown tty2                         Tue Jun  4 08:51 - 08:51  (00:00)
rpcuser  tty2                         Tue Jun  4 08:50 - 08:50  (00:00)
chrony   tty2                         Tue Jun  4 08:49 - 08:49  (00:00)
(unknown tty2                         Tue Jun  4 08:30 - 08:30  (00:00)
(unknown tty2                         Tue Jun  4 08:30 - 08:30  (00:00)
(unknown tty2                         Tue Jun  4 08:30 - 08:30  (00:00)
(unknown tty2                         Tue Jun  4 08:28 - 08:28  (00:00)
root     tty2                         Tue Jun  4 08:28 - 08:28  (00:00)
root     tty2                         Tue Jun  4 08:27 - 08:27  (00:00)

btmp begins Tue Jun  4 08:27:54 2019
```

The list of failed logins shows the user account that the attempt was targeted at, if it exists. If the account does not exist, the account name is shown as (Unknown) which has been truncated.

Now let's look at a small bit of the **lastb** data stream from my firewall system. I use a Linux host as a firewall and router. These login attempts are via the Internet over SSH. I use SSH to remotely log in to my network through this firewall so I have the SSHD server running and accepting connections.

The leftmost column of this output shows the user account name that the attack targeted. The second column shows the attack against SSH and "notty" means no tty was assigned. Column three is the IP address from which the attack generated, although that can be spoofed. The rest of the columns list the data and time the attack occurred. The "(00:00)" (MM:SS) column just means that the connection lasted zero time.

```
karika   ssh:notty    91.134.241.32    Tue Jun  4 08:12 - 08:12  (00:00)
karika   ssh:notty    91.134.241.32    Tue Jun  4 08:12 - 08:12  (00:00)
gfa      ssh:notty    79.6.34.129      Tue Jun  4 08:11 - 08:11  (00:00)
gfa      ssh:notty    79.6.34.129      Tue Jun  4 08:11 - 08:11  (00:00)
mjestel  ssh:notty    91.134.241.32    Tue Jun  4 08:09 - 08:09  (00:00)
mjestel  ssh:notty    91.134.241.32    Tue Jun  4 08:09 - 08:09  (00:00)
redmine  ssh:notty    128.199.170.177  Tue Jun  4 08:09 - 08:09  (00:00)
redmine  ssh:notty    128.199.170.177  Tue Jun  4 08:09 - 08:09  (00:00)
cow      ssh:notty    79.6.34.129      Tue Jun  4 08:08 - 08:08  (00:00)
cow      ssh:notty    79.6.34.129      Tue Jun  4 08:08 - 08:08  (00:00)
alberta  ssh:notty    51.75.124.76     Tue Jun  4 08:08 - 08:08  (00:00)
alberta  ssh:notty    51.75.124.76     Tue Jun  4 08:08 - 08:08  (00:00)
<snip>
ec2-user ssh:notty    181.114.209.13   Sat Jun  1 00:23 - 00:23  (00:00)
ec2-user ssh:notty    181.114.209.13   Sat Jun  1 00:23 - 00:23  (00:00)
!@#$%^   ssh:notty    180.76.108.110   Sat Jun  1 00:22 - 00:22  (00:00)
!@#$%^   ssh:notty    180.76.108.110   Sat Jun  1 00:22 - 00:22  (00:00)
performe ssh:notty    180.76.108.110   Sat Jun  1 00:19 - 00:19  (00:00)
performe ssh:notty    180.76.108.110   Sat Jun  1 00:19 - 00:19  (00:00)
zhuang   ssh:notty    129.204.46.170   Sat Jun  1 00:18 - 00:18  (00:00)
zhuang   ssh:notty    129.204.46.170   Sat Jun  1 00:18 - 00:18  (00:00)
usp      ssh:notty    181.114.209.13   Sat Jun  1 00:16 - 00:16  (00:00)
usp      ssh:notty    181.114.209.13   Sat Jun  1 00:16 - 00:16  (00:00)
geminroo ssh:notty    140.143.93.31    Sat Jun  1 00:16 - 00:16  (00:00)
```

```
geminroo ssh:notty    140.143.93.31    Sat Jun  1 00:16 - 00:16  (00:00)
trinity  ssh:notty    218.75.102.110   Sat Jun  1 00:12 - 00:12  (00:00)
trinity  ssh:notty    218.75.102.110   Sat Jun  1 00:12 - 00:12  (00:00)
fv       ssh:notty    140.143.93.31    Sat Jun  1 00:12 - 00:12  (00:00)
fv       ssh:notty    140.143.93.31    Sat Jun  1 00:12 - 00:12  (00:00)
script   ssh:notty    129.204.46.170   Sat Jun  1 00:12 - 00:12  (00:00)
script   ssh:notty    129.204.46.170   Sat Jun  1 00:12 - 00:12  (00:00)
mongo    ssh:notty    5.39.88.4        Sat Jun  1 00:11 - 00:11  (00:00)
mongo    ssh:notty    5.39.88.4        Sat Jun  1 00:11 - 00:11  (00:00)
zi       ssh:notty    5.39.88.4        Sat Jun  1 00:08 - 00:08  (00:00)
zi       ssh:notty    5.39.88.4        Sat Jun  1 00:08 - 00:08  (00:00)

btmp begins Sat Jun  1 00:08:40 2019
```

This output had 4600 lines in it and the last two do not count. There are 4598 lines of failed login attempts. Look at the dates. These data run from midnight on June 1 of 2019 through 08:12 AM June 4. So we have almost 4600 failed login attempts in 3 days and a bit over 8 hours.

Note the different and sometimes strange user account names used in the attacks. Some are those belonging to Linux system services, some are apparently legitimate user account names, but others are clearly random and concocted. I particularly like !@#$%^ which consists of the special characters above the 1, 2, 3, 4, 5, and 6 keys on the US keyboard. "Cow" is also kind of strange.

You can download the complete results from my firewall host. As the student user, download the Chapter-17.tgz file (tarball) from the Apress GitHub repository into your ~/Downloads directory. Then untar the file.

```
[student@studentvm1 ~]$ cd ~/Downloads/
[student@studentvm1 Downloads]$ wget https://github.com/Apress/using-and-
administering-linux-volume-2/blob/master/Chapter-17.tgz
```

A quick explanation of the **tar** command is that it can be used to create an archive type called a tarball with a file extension of .tar. The files can be compressed using another Linux tool, **gzip**. The file extension for a gzip'ed tarball is .tgz. We will extract the file from this tarball with the **tar** command.

The -x option means to extract the files contained in the tarball. -z specifies decompression with **gunzip**, the reverse of **gzip**. The -v option means verbose so that the command will list the files as it extracts them. And -f is used to specify the name of the tarball from which

we are going to extract the files. We will explore more aspects of the tar command, including using it to create tarballs for backup purposes, in Chapter 18 of this volume.

```
[student@studentvm1 Downloads]$ tar -xzvf Chapter-36.tgz ; ll
lastb.txt
total 388
-rw-rw-r-- 1 student student  39514 Jun  4 12:07 Chapter36.tgz
-rw-r--r-- 1 student student 335692 Jun  4 08:21 lastb.txt
[student@studentvm1 Downloads]$
```

Look through the lastb.txt file which is quite long but which has an interesting array of targeted account names and source IP addresses.

The full results from my firewall host in Experiment 17-1 show only some of the many failed login attempts against my firewall. That is only partially because I snipped most of them out of what you see here. It is also because after four failed attempts logged from any single IP address within a period of 24 minutes (1440 seconds), I block any further attempts from that IP for 24 minutes. Any connection attempts from blocked IP addresses do not even get logged, they just get dropped. I use an open source tool called Fail2Ban for this and we will explore SSH, and firewalls, and Fail2Ban in the next sections.

The point here is that there are constant and large numbers of attacks against every host connected to the Internet.

# Telnet

Telnet is an old and well-known terminal emulator that provided a very easy way to connect to remote hosts. Telnet was developed in a time before the advent of the Internet for everyone, when the only Internet was the ARPANET connecting large universities with each other and the Department of Defense (DOD). There were few connections and everyone was collaborative. There was no malware and it was a safe place.

As a result, Telnet was not developed with security as a prime consideration. All communications between hosts were in plain ASCII text with no encryption – including the user ID and password. As the Internet grew and the cast of players on the dark side also grew, this lack of security became a problem.

However, a short exploration of Telnet can provide some interesting insight into how easily anyone with just a bit of knowledge can eavesdrop on an unencrypted connection. Think – the typical wireless connections in public places that do not require passwords.

Before we look at Telnet, however, we need a tool that will let us explore TCP packets and their contents. The **tcpdump** utility allows us do that and it is already installed.

## EXPERIMENT 17-2

Perform this experiment as the root user. Let's start with a bit of preparation. Previous experiments may have launched some connections that have not been dropped and we want to start this experiment with as few connections as possible. So reboot StudentVM1.

Now dump the headers of all the packets on the enp0s3 interface. The headers tell us what kind of packet it is and a bit of other information about the packet. The source and destination IP addresses that are contained in the packet are also displayed.

We need some packets to dump so let's start a ping to the virtual router in our virtual network. The ip route command displays the IP address of the default gateway router. Do this in a terminal session as root. Your default route should be the same as mine, 10.0.2.1, but there is a possibility that it might be different. Regardless, the IP address of the router is given on the "default via" line of the results.

```
[root@studentvm1 ~]# ip route
default via 10.0.2.1 dev enp0s3 proto dhcp metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.5 metric 100
```

In another terminal session as root, start **tcpdump**. The -i option specifies the interface we want to view. Without the -i option tcpdump will display packets from all interfaces including lo; specifying an interface makes it easier for us to see the packets we want.

```
[root@studentvm1 ~]# tcpdump -i enp0s3
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), capture size 262144 bytes
```

At this point you should see only the following couple lines to indicate that **tcpdump** is listening on enp0s3. Be sure to have the terminal session with **tcpdump** running in a location where you can see it. Now let's generate a bit of traffic in a different terminal session as root.

```
[root@studentvm1 ~]# ping 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
```

```
64 bytes from 10.0.2.1: icmp_seq=1 ttl=255 time=0.545 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=255 time=0.225 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=255 time=0.363 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=255 time=0.128 ms
<snip>
```

You should see results like these in the **tcpdump** session. First – on my system – we see some DNS requests looking for the NTP server. You probably will not see that unless you watch the data stream for some time.

```
08:56:14.212503 IP studentvm1.34113 > ntp1.glypnod.com.ntp: NTPv4, Client, length 48
08:56:14.213202 IP studentvm1.59682 > router.domain: 12976+ PTR? 175.155.131.104.
in-addr.arpa. (46)
08:56:14.247782 IP router.domain > studentvm1.59682: 12976 ServFail 0/0/0 (46)
08:56:14.247996 IP studentvm1.35991 > google-public-dns-a.google.com.domain: 12976+
PTR? 175.155.131.104.in-addr.arpa. (46)
08:56:14.282835 IP google-public-dns-a.google.com.domain > studentvm1.35991: 12976
1/0/0 PTR ntp1.glypnod.com. (76)
08:56:14.305939 IP studentvm1.54924 > router.domain: 33356+ PTR? 8.8.8.8.in-addr.
arpa. (38)
08:56:14.306377 IP ntp1.glypnod.com.ntp > studentvm1.34113: NTPv4, Server, length 48
08:56:15.050090 IP router.domain > studentvm1.54924: 33356 1/4/8 PTR google-public-
dns-a.google.com. (330)
08:56:19.498446 ARP, Request who-has router tell studentvm1, length 28
08:56:19.498635 ARP, Reply router is-at 52:54:00:12:35:00 (oui Unknown), length 46
08:57:23.737060 IP studentvm1 > router: ICMP echo request, id 18956, seq 1, length 64
08:57:23.737207 IP router > studentvm1: ICMP echo reply, id 18956, seq 1, length 64
08:57:24.778523 IP studentvm1 > router: ICMP echo request, id 18956, seq 2, length 64
08:57:24.778708 IP router > studentvm1: ICMP echo reply, id 18956, seq 2, length 64
08:57:25.802521 IP studentvm1 > router: ICMP echo request, id 18956, seq 3, length 64
08:57:25.802707 IP router > studentvm1: ICMP echo reply, id 18956, seq 3, length 64
08:57:26.826517 IP studentvm1 > router: ICMP echo request, id 18956, seq 4, length 64
08:57:26.826673 IP router > studentvm1: ICMP echo reply, id 18956, seq 4, length 64
08:57:27.850495 IP studentvm1 > router: ICMP echo request, id 18956, seq 5, length 64
08:57:27.850683 IP router > studentvm1: ICMP echo reply, id 18956, seq 5, length 64
08:57:28.874486 IP studentvm1 > router: ICMP echo request, id 18956, seq 6, length 64
08:57:28.874660 IP router > studentvm1: ICMP echo reply, id 18956, seq 6, length 64
08:57:29.130418 ARP, Request who-has router tell studentvm1, length 28
08:57:29.130756 ARP, Reply router is-at 52:54:00:12:35:00 (oui Unknown), length 46
```

Then we see an ARP[2] request that indicates that our host is looking for the MAC address of the router and the corresponding response. This is those two devices making their layer 1 physical connection. It is a request to the router, using the router's known IP address, that seeks the MAC address of the router and it also sends the MAC address for the enp0s3 NIC. The router then responds with a message targeted to the MAC address of enp0s3 on the studentvm1 host. This is the layer at which hosts actually communicate with each other when they are on the same physical (or virtual) network segment.

After that, there are several ping (ICMP[3]) requests from StudentVM1 and the corresponding responses from the router.

Terminate the ping events with Ctrl-C and continue to watch the data stream for several minutes. You will eventually see some additional traffic which should mostly be NTP traffic and related ARP traffic. You may also see some DHCP traffic when the IP address given to the studentvm1 host expires and it requests a new lease from the DHCP server.

This is a very basic introduction to a powerful and complex tool. All we have seen so far are the packet headers – information about the packets – not the contents. Exit **tcpdump** with Ctrl-C.

---

Opensource.com has an excellent introductory article[4] about **tcpdump** on their web site. There are any number of tools, both CLI and GUI, as well as free open source and for a fee commercial, that provide many or all of these functions. The **tcpdump** tool is free and open source and has been around for a long time.

Now we can proceed with installing Telnet and xinetd. The xinetd package is used to manage a number of older server types, including Telnet. It is not required on most modern Linux hosts. It is configured with a main file, /etc/xinetd.conf, and with individual service files in the /etc/xinetd.d directory.

[2]ARP, Address Resolution Protocol, a protocol that enables discovery of the MAC (hardware) address of a network interface on a remote host using the host's IP address

[3]ICMP, Free On-line Dictionary of Computing (FOLDOC), an extension to the Internet Protocol (IP) that allows for the generation of error messages, test packets, and informational messages related to IP. It is defined in STD 5, RFC 792.

[4]Ricardo Gerardi, *An introduction to using tcpdump at the Linux command line*, https://opensource.com/article/18/10/introduction-tcpdump

## EXPERIMENT 17-3

Perform this experiment as root. Install the telnet-server and xinetd packages.

```
[root@studentvm1 etc]# dnf -y install telnet-server xinetd
```

Configuring the xinetd server is a bit of a challenge because the file needed to configure xinetd to manage the Telnet service is not installed by default. So we must create it ourselves. Create the file /etc/xinetd.d/telnet and add the content shown in the following data. The file should have the permissions 600; having other permissions will not prevent Telnet from working, but it would be a security issue because we do not want non-privileged users to read most configuration files including this one.

```
# default: on
# description: The telnet server serves telnet sessions; it uses \
#       unencrypted username/password pairs for authentication.
service telnet
{
         flags           = REUSE
         socket_type     = stream
         wait            = no
         user            = root
         server          = /usr/sbin/in.telnetd
         log_on_failure  += USERID
         disable         = no
}
```

We need to allow Telnet through our firewall on port 23. How do we know the port number? The /etc/services file contains a listing of all assigned and commonly recognized ports and the services assigned to them.

```
[root@studentvm1 ~]# grep -i telnet /etc/services
telnet          23/tcp
telnet          23/udp
rtelnet         107/tcp                         # Remote Telnet
rtelnet         107/udp
```

```
telnets         992/tcp
telnets         992/udp
su-mit-tg       89/tcp                  # SU/MIT Telnet Gateway
su-mit-tg       89/udp                  # SU/MIT Telnet Gateway
```

The data stream from this command shows many Telnet variants, many dating from early days of computer communications. We want the first one, plain Telnet, which is port 23.

Now add the firewall rule. The first command adds port 23 for TCP (not UDP) to the firewall permanently and the second reloads the firewall configuration to activate the new rule.

```
[root@studentvm1 ~]# firewall-cmd --permanent --add-port=23/tcp
success
[root@studentvm1 ~]# firewall-cmd --reload
success
[root@studentvm1 ~]#
```

Check the firewall status.

```
[root@studentvm1 ~]# firewall-cmd --list-ports
23/tcp
[root@studentvm1 ~]#
```

The xinetd service is managed by systemd and is enabled by default so that it will start on boot. Ensure that it is up and running.

```
[root@studentvm1 xinetd.d]# systemctl status xinetd
```

If the xinetd service is not active and running, start it.

```
[root@studentvm1 xinetd.d]# systemctl start xinetd
```

We will explore firewalld in more detail later in this chapter.

---

We are now ready to explore Telnet.

## EXPERIMENT 17-4

Perform this experiment as root. In one root terminal session that should remain visible on the desktop for the rest of this experiment, start **tcpdump** and monitor the lo (local) interface for Telnet communications on port 23. Start a Telnet session with the localhost.

```
[root@studentvm1 ~]# telnet localhost
Trying ::1...
Connected to localhost.
Escape character is '^]'.

Kernel 5.0.7-200.fc29.x86_64 on an x86_64 (7)
studentvm1 login: root
Password: <enter the root password>
Last login: Mon Jun 10 21:33:49 from 192.168.0.1
Running /etc/profile
Running /etc/profile.d/myBashConfig.sh
Running /etc/bashrc
Running /root/.bash_profile
Running /root/.bashrc
Running /etc/bashrc
[root@studentvm1 ~]#
```

The tcpdump terminal session should now contain a data stream that looks similar to this. I have only reproduced a few lines here to save space.

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
14:02:59.472466 IP6 localhost.39640 > localhost.telnet: Flags [S], seq
612902319, win 65476, options [mss 65476,sackOK,TS val 1287054799 ecr
0,nop,wscale 7], length 0
14:02:59.472496 IP6 localhost.telnet > localhost.39640: Flags [S.], seq
3258934320, ack 612902320, win 65464, options [mss 65476,sackOK,TS val
1287054799 ecr 1287054799,nop,wscale 7], length 0
14:02:59.472512 IP6 localhost.39640 > localhost.telnet: Flags [.], ack 1, win
512, options [nop,nop,TS val 1287054799 ecr 1287054799], length 0
14:02:59.473651 IP6 localhost.39640 > localhost.telnet: Flags [P.], seq 1:25,
ack 1, win 512, options [nop,nop,TS val 1287054800 ecr 1287054799], length
```

```
24 [telnet DO SUPPRESS GO AHEAD, WILL TERMINAL TYPE, WILL NAWS, WILL TSPEED,
WILL LFLOW, WILL LINEMODE, WILL NEW-ENVIRON, DO STATUS [|telnet]
14:02:59.473674 IP6 localhost.telnet > localhost.39640: Flags [.], ack 25,
win 512, options [nop,nop,TS val 1287054800 ecr 1287054800], length 0
14:02:59.478708 IP6 localhost.telnet > localhost.39640: Flags [P.], seq 1:13,
ack 25, win 512, options [nop,nop,TS val 1287054805 ecr 1287054800], length
12 [telnet DO TERMINAL TYPE, DO TSPEED, DO XDISPLOC, DO NEW-ENVIRON [|telnet]
14:02:59.478736 IP6 localhost.39640 > localhost.telnet: Flags [.], ack 13,
win 512, options [nop,nop,TS val 1287054805 ecr 1287054805], length 0
```

Now run the **ll** command while watching the tcpdump data stream.

So far, we have only looked at the packet headers for this Telnet session. Terminate the tcpdump command and restart it using the -A option which dumps the data contents of the packets in ASCII format.

```
[root@studentvm1 ~]# tcpdump -i lo port 23 -A
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
```

One packet you should find will look something like this one from my StudentVM1 host. You can see the data contained in the packet. This is quite insecure and could result in the exposure of private data to anyone on the Internet who cares enough to listen in.

```
14:26:09.919600 IP6 localhost.telnet > localhost.39642: Flags [P.], seq
623:1175, ack 8, win 512, options [nop,nop,TS val 1288445246 ecr 1288445246],
length 552
`....H.@.....................................J..c\e.......P.....
L..>L..>-rw-------. 1 root root 2118 Dec 22 11:07 anaconda-ks.cfg
drwxr-xr-x  2 root root 4096 Apr 16 17:24 .[0m.[01;34mbin.[0m
-rwxrwx---  1 root root 3318 Apr 16 08:17 .[01;32mdoUpdates.[0m
-rw-r--r--. 1 root root  308 Dec 22 11:06 ifcfg-enp0s3
-rw-r--r--. 1 root root 2196 Dec 22 12:47 initial-setup-ks.cfg
-rw-r--r--. 1 root root  308 Dec 22 11:06 original.ifcfg-enp0s8w
-rw-r--r--  1 root root    0 May 14 15:17 .[01;35msystemd.svg.[0m
-rw-r--r--  1 root root  101 May 14 08:57 TestFS.automount
-rw-r--r--  1 root root  284 Apr 18 14:59 test.log
```

Use Ctrl-C to terminate the tcpdump session. Close the Telnet session using the **exit** command. Stop the Telnet server service.

Despite its lack of security, Telnet is an excellent tool for learning about network communications. The **tcpdump** command has an extensive man page that provides an excellent reference.

It is worth noting at this point that Telnet, the venerable but totally insecure remote terminal utility, can still have its uses as we will see in Volume 3 of this series.

# SSH

We touched very briefly on SSH in Chapter 7 of Volume 1. SSH stands for Secure Shell, but SSH is not really a shell. The **ssh** command starts a secure terminal link between itself as the client and another host with the SSHD server running on it. The actual command shell used at the server end is whatever the default shell set for that user account on the server side, such as the Bash shell. SSH is a network protocol that creates a secure communications tunnel between two Linux hosts. It is like a software virtual private network (VPN).[5]

The function of SSH, which is implemented in Linux by the OpenSSH package, is to enable secure, encrypted login connections to remote hosts using the command-line interface and a standard shell such as Bash. This prevents Internet skulkers from reading your password and the entire connected session in plain text. The SSH client and server work together to provide a session that is safely encrypted from start to finish.

## The SSH server

Our intent in exploring SSH is to understand the client side of this tool. However, we do need a server to which the client can connect. I did look on the Internet for a public server that could be used to test our clients, but I could not find one. So the next best thing is to use our own SSH server on our StudentVM1 host. Yes, it is possible and easily done so that we can use both client and server on the same host.

[5]Wikipedia, *Virtual Private Network*, https://en.wikipedia.org/wiki/Virtual_private_network

The SSH server is always installed during a Fedora installation. Configuration of an SSH server is fairly easy. In fact, the default configuration is fine for many environments. The default configuration of SSHD (the SSH server daemon) allows root login, but this is easily changed if necessary. We will use this root login when we do backups using **rsync** in Chapter 18 of this volume. It is also important to note that the default configuration of the firewalld firewall allows incoming connections to the SSH server so we do not need to change the firewall.

We will keep this very simple for now but will go into more detail about SSH and SSHD server configuration in Volume 3 of this series.

## EXPERIMENT 17-5

Begin this experiment as the root user. Perform a listing of all files (-a) in the root home directory of your student host. You should not find a directory named ~/.ssh in the list. This directory is where local SSH configuration files for the user are located, but it does not get created until the first time the user connects to a remote (or local) host with SSH.

We will begin by starting the SSHD server daemon and enabling it so it will start on boot.

```
[root@studentvm1 ~]# systemctl start sshd ; systemctl enable sshd
Created symlink /etc/systemd/system/multi-user.target.wants/sshd.service → /
usr/lib/systemd/system/sshd.service.
[root@studentvm1 ~]#
```

Our StudentVM1 host is now ready for us to try out an SSH connection. We use localhost for this just as we did for our Telnet connection.

```
[root@studentvm1 ~]# ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is SHA256:NDM/B5L3eRJaalex6IOUdnJsE1smOSiQNWgaI8BwcVs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
root@localhost's password: <Enter Password>
Last login: Wed Jun  5 07:21:28 2019 from david.both.org
Running /etc/profile
Running /etc/profile.d/myBashConfig.sh
Running /etc/bashrc
Running /root/.bash_profile
```

```
Running /root/.bashrc
Running /etc/bashrc
[root@studentvm1 ~]#
```

The first time an SSH connection is made to any host, the authenticity message is displayed along with the fingerprint of the private key of the remote (in this case local) host. In a very security conscious environment, we would have already received a copy of the remote host's key fingerprint. This allows comparison so that we know we are connecting to the correct remote host. This is not the security key; it is a fingerprint that is unique to that private key. It is impossible to reconstruct the original private key from which the fingerprint was generated.

You must type "yes" – the full word – in order for the full public key to be transmitted from the remote host to the local one. Then you must enter the password for the remote host.
You should also connect to the localhost as the student user via SSH.

Now let's look at the /root/.ssh directory. Then look at the contents of the ~/.ssh/known_hosts file. You should see the public host key for the remote host. This file is created in the localhost, the one we are connecting from, and not on the remote host, the one we are connecting to.

```
[root@studentvm1 ~]# cat .ssh/known_hosts
localhost ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMDg3AOuakzj1P14aJge0
HCRSJpsxOAlU6fXiVRlc/RwQRvFkMbl05/t7wSFcwOG8tRSiNaktVs4dxpAoMbrT3c=
```

After accepting this key during the first connection to the remote host, the connections initialize a little faster because the two computers now know each other and can identify themselves via the keys.

Type **exit** to disconnect the SSH connection. Exit from all SSH connections if there are more than just one.

If this is just a bit confusing, that may be due to the fact that we are using the same host for the originator (client) and the receiver (server) of the SSH connection.

---

Now let's look at the content of the SSH TCP/IP packets.

## EXPERIMENT 17-6

Perform this experiment as the root user. In one terminal session that is located where it can be seen from other terminal sessions, start tcpdump listening to SSH on port 22 of the lo interface. I use the tee command so that we can view the data stream on the terminal screen as well as store it in a file for later use.

```
[root@studentvm1 ~]# tcpdump -i lo port 22 -A | tee /root/tcpdump-ssh.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
```

In another terminal session, ssh to the localhost and use the ll command like we did for the Telnet connection in Experiment 17-4.

```
[root@studentvm1 ~]# ssh localhost
root@localhost's password:
Last login: Wed Jun 12 16:02:48 2019 from ::1
Running /etc/profile
Running /etc/profile.d/myBashConfig.sh
Running /etc/bashrc
Running /root/.bash_profile
Running /root/.bashrc
Running /etc/bashrc
[root@studentvm1 ~]#  ll
total 44
-rw-------. 1 root root  2118 Dec 22 11:07 anaconda-ks.cfg
drwxr-xr-x  2 root root  4096 Apr 16 17:24 bin
-rwxrwx---  1 root root  3318 Apr 16 08:17 doUpdates
-rw-r--r--. 1 root root   308 Dec 22 11:06 ifcfg-enp0s3
-rw-r--r--. 1 root root  2196 Dec 22 12:47 initial-setup-ks.cfg
-rw-r--r--. 1 root root   308 Dec 22 11:06 original.ifcfg-enp0s8w
-rw-r--r--  1 root root     0 May 14 15:17 systemd.svg
-rw-r--r--  1 root root 12288 Jun 12 16:07 tcpdump-ssh.txt
-rw-r--r--  1 root root   101 May 14 08:57 TestFS.automount
-rw-r--r--  1 root root   284 Apr 18 14:59 test.log
[root@studentvm1 ~]
```

The data stream passed very quickly on the terminal screen. Exit from the SSH session and terminate the **tcpdump** collection as well and use **less** to view the content of the file we created, /root/tcpdump-ssh.txt. Near the top of this file, you will see some ASCII plain text data which are lists of security protocols that the client and the server communicate to each other as they negotiate the best protocol to use for the connection. After that you will not find the results of the **ll** command as we did in the Telnet data stream, but merely encoded data.

The whole point to this series of experiments is that SSH is the secure communication protocol and is the only choice to ensure secure terminal connections to remote hosts. We will explore SSH communications in more detail in the next course in this series, *Advanced Linux System and Server Administration*.

# Firewalls

Firewalls are a very important part of computer and network security. Firewalls can block any and all attempts to access our Linux hosts by way of the networks to which we are connected. firewalld is the default firewall management tool used by current releases of Fedora. It replaced IPTables which had been around for many years. However, firewalld and IPTables are both just management wrappers around the netfilter[6] functions that are an intrinsic component of the Linux kernel.

All Linux firewalls are based on netfilter. The tools we use with many different names simply allow us to add, modify, and remove rules that are used by netfilter to examine each data packet and determine how to handle it.

By default, firewalld and IPTables both have rule sets that block all incoming packets unless explicitly allowed. This, plus the fact that most services that are not needed are not installed or not enabled, means that Linux is very secure right from the initial installation. Outbound packets are not blocked so that we don't need to add rules for protocols like email, SSH, and web browsers just to access these services from our client hosts.

The flow of packets as they enter the Linux host is generally from start to finish through the rule set. If a packet matches one of the rules, the action defined in the rule is taken. Ultimately, each packet will be accepted, rejected, or dropped. When a packet matches a rule that has one of these three actions, that action is taken and the packet travels no further through the rules. These actions require just a bit of explanation:

[6]Wikipedia, *netfilter*, https://en.wikipedia.org/wiki/netfilter

- **Accept**: The packet is accepted and passed to the port and a server such as a web server, Telnet, or SSH to which it is addressed. The dport is the destination port.
- **Reject**: The packet is rejected and sent back to the originator with a message. This message allows the host on the other end to know what happened and try again if need be or to terminate the connection.
- **Drop**: The packet is dropped and proceeds no further through the rules. No message is sent back to the originator. This action maintains the connection for the timeout period specified on the sender's end. This is useful when blocking IP addresses of known spammers, so that when they attempt a connection, their sending host must wait through the timeout to try again, thus slowing down their attacks significantly.

## firewalld

The firewalld service provides a complex and intricate set of rules for a firewall. It uses the concept of zones to collect related rules in such a way as to create levels of trust. Each zone represents a level of trust that can be independently modified without affecting other zones. firewalld has several zones predefined.

These zones are arbitrary constructs developed to meet a specific perceived set of needs in a firewall. The firewalld tool has been designed to manage the firewall using these zones. That is, in and of itself, neither bad nor good.

Let's take a quick look at the firewall rules on our VMs.

### EXPERIMENT 17-7

Perform this experiment as the root user. We can use the iptables-save command to view the current rule set. This command does not actually save the rules; it only prints them to the screen. Your rules should look like mine although I have pruned the following data stream by quite a bit.

```
[root@studentvm1 ~]# iptables-save | less
# Generated by iptables-save v1.8.0 on Thu Jun 13 08:04:33 2019
*nat
```

```
:PREROUTING ACCEPT [104:8914]
:INPUT ACCEPT [81:5794]
:OUTPUT ACCEPT [13539:1741469]
:POSTROUTING ACCEPT [13539:1741469]
:OUTPUT_direct - [0:0]
:POSTROUTING_ZONES - [0:0]
:POSTROUTING_ZONES_SOURCE - [0:0]
:POSTROUTING_direct - [0:0]
:POST_public - [0:0]
:POST_public_allow - [0:0]
:POST_public_deny - [0:0]
:POST_public_log - [0:0]
:PREROUTING_ZONES - [0:0]
:PREROUTING_ZONES_SOURCE - [0:0]
:PREROUTING_direct - [0:0]
:PRE_public - [0:0]
:PRE_public_allow - [0:0]
:PRE_public_deny - [0:0]
:PRE_public_log - [0:0]
-A PREROUTING -j PREROUTING_direct
-A PREROUTING -j PREROUTING_ZONES_SOURCE
-A PREROUTING -j PREROUTING_ZONES
-A OUTPUT -j OUTPUT_direct
-A POSTROUTING -j POSTROUTING_direct
-A POSTROUTING -j POSTROUTING_ZONES_SOURCE
-A POSTROUTING -j POSTROUTING_ZONES
-A POSTROUTING_ZONES -o enp0s8 -g POST_public
-A POSTROUTING_ZONES -o enp0s3 -g POST_public
-A POSTROUTING_ZONES -g POST_public
-A POST_public -j POST_public_log
-A POST_public -j POST_public_deny
-A POST_public -j POST_public_allow
-A PREROUTING_ZONES -i enp0s8 -g PRE_public
-A PREROUTING_ZONES -i enp0s3 -g PRE_public
-A PREROUTING_ZONES -g PRE_public
-A PRE_public -j PRE_public_log
-A PRE_public -j PRE_public_deny
```

```
-A PRE_public -j PRE_public_allow
COMMIT
# Completed on Thu Jun 13 08:04:33 2019
# Generated by iptables-save v1.8.0 on Thu Jun 13 08:04:33 2019

<snip>

-A INPUT_ZONES -i enp0s8 -g IN_public
-A INPUT_ZONES -i enp0s3 -g IN_public
-A INPUT_ZONES -g IN_public
-A IN_public -j IN_public_log
-A IN_public -j IN_public_deny
-A IN_public -j IN_public_allow
-A IN_public -p icmp -j ACCEPT
-A IN_public_allow -p tcp -m tcp --dport 22 -m conntrack --ctstate
NEW,UNTRACKED -j ACCEPT
-A IN_public_allow -d 224.0.0.251/32 -p udp -m udp --dport 5353 -m conntrack
--ctstate NEW,UNTRACKED -j ACCEPT
-A IN_public_allow -p tcp -m tcp --dport 23 -m conntrack --ctstate
NEW,UNTRACKED -j ACCEPT
COMMIT
# Completed on Thu Jun 13 08:06:57 2019
```

The data from Experiment 17-7 shows a set of firewall rules that is way more complex than needed for most Linux hosts unless they are being used as edge routers and firewalls to protect the internal network from the Internet crackers. Most workstations and servers would never require firewall rules of this complexity. The entire function of this rule set is to reject all incoming packets except for SSH and the rule we added for Telnet.

One of the tenets I discuss in Chapter 18 of *The Linux Philosophy for SysAdmins* is "Find the Simplicity." The bottom line here is that a simple set of easy to understand and change is better than a complex set of rules that perform the same task.

However, regardless of its complexity, it is easy to add and delete firewalld rules. So it makes little sense to revert to IPTables except for illustrative purposes, as part of a simplification effort, or because you will be making changes to the rule sets, or just because you value simplicity.

## Understanding the rules

We will convert firewalld to iptables a bit further on, but there is one key secret here. All of the rules in the firewalld firewall are standard netfilter rules, just as they are with IPTables. In fact, the iptables-save command creates a series of rules that, when prefaced with the iptables command, become a script to create the firewall.

So the IPTables command **`iptables -A IN_public_allow -p tcp -m tcp --dport 23 -m conntrack --ctstate NEW,UNTRACKED -j ACCEPT`** -A appends this rule to the IN_public_allow chain. It -m matches for TCP protocol for destination port 23 (Telnet) and specifies that connection tracking is NOT being used on these packets. If the packet is a match for Telnet/TCP, it -j jumps to the ACCEPT target which is a condition not a rule. Specifying the TCP protocol prevents access attempts using UDP.

The COMMIT lines in the script commit the new rules to the netfilter rules in the kernel.

The other rules can be parsed in much the same manner.

All of these high-level firewall tools like IPTables, firewalld, and others are just wrappers around the netfilter functions of the Linux kernel. They provide us SysAdmins with methods to manage the rules that netfilter uses to check each packet and determine what to do with it.

## Deleting and adding rules

As an old – er, mature – SysAdmin, I tend to think in terms of ports when working with firewalls and network services. Sometimes I need to look up a port number associated with a particular service, but that is no big deal because they are all defined in /etc/services, as we have already seen.

The firewalld firewall works perfectly well with port numbers, but it has many services and their respective ports predefined, and Telnet is one of those predefined services. So let's remove the rule that we added earlier which allows Telnet access using the port number 23 and add the equivalent rule using the service name instead of the port number.

## EXPERIMENT 17-8

Perform this experiment as root. Start by verifying the existence of the rule. You can use **iptables-save**, as we did previously, or we can be a little more targeted.

```
[root@studentvm1 ~]# firewall-cmd --list-ports
23/tcp
[root@studentvm1 ~]# firewall-cmd --list-ports --permanent
23/tcp
```

Note that we list both runtime ports which are temporary and the permanent configuration which will last through a reboot. Changes made without making them permanent will be lost at a reboot. We need to change both the runtime rules

```
[root@studentvm1 ~]# firewall-cmd --remove-port=23/tcp
success
[root@studentvm1 ~]# firewall-cmd --list-ports

[root@studentvm1 ~]# firewall-cmd --list-ports --permanent
23/tcp
[root@studentvm1 ~]#
```

and the permanent rules.

```
<pre>[root@studentvm1 ~]# firewall-cmd --remove-port=23/tcp --permanent
success
[root@studentvm1 ~]# firewall-cmd --list-ports

[root@studentvm1 ~]# firewall-cmd --list-ports --permanent

[root@studentvm1 ~]#
```

Add the Telnet service to the runtime firewall and verify that it has been added there but not to the permanent group of services.

```
[root@studentvm1 ~]# firewall-cmd --add-service=telnet
success
[root@studentvm1 ~]# firewall-cmd --list-services
dhcpv6-client mdns ssh telnet
[root@studentvm1 ~]# firewall-cmd --list-services --permanent
dhcpv6-client mdns ssh
```

We can use the following command to make the runtime configuration permanent. Using the --runtime-to-permanent option is easier if you have made multiple changes to the runtime configuration and need to make them all permanent.

```
[root@studentvm1 ~]# firewall-cmd --runtime-to-permanent
success
[root@studentvm1 ~]# firewall-cmd --list-services
dhcpv6-client mdns ssh telnet
[root@studentvm1 ~]# firewall-cmd --list-services --permanent
dhcpv6-client mdns ssh telnet
[root@studentvm1 ~]#
```

Log in to the localhost using Telnet to verify that this new configuration works as it should. You might also try rebooting to ensure that the new configuration really is permanent.

You probably noticed that a service named MDNS is open on our virtual machine; MDNS is Multicast DNS. It is used as a hostname resolver only in small networks where there is no explicitly defined name server. Our virtual network uses the name server built in to the virtual router for external hostname services, and we have already added the router and its IP address of 10.0.2.1 to the only host on our virtual network. The MDNS service is not required so let's remove the rule that allows access to that port from our firewall.

## EXPERIMENT 17-9

Perform this experiment as root. Remove the MDNS service from the firewall, make it permanent, and verify that it has been removed from both permanent and runtime configuration rule sets.

```
[root@studentvm1 ~]# firewall-cmd --remove-service=mdns
success
[root@studentvm1 ~]# firewall-cmd --list-services --permanent
dhcpv6-client mdns ssh telnet
[root@studentvm1 ~]# firewall-cmd --list-services
dhcpv6-client ssh telnet
[root@studentvm1 ~]# firewall-cmd --runtime-to-permanent
```

```
success
[root@studentvm1 ~]# firewall-cmd --list-services --permanent
dhcpv6-client ssh telnet
[root@studentvm1 ~]#
```

When using firewalld for firewall services, it is always a good idea to be consistent and add new rules using the service name rather than the port number. It may still be necessary to use the port number for a service that is not predefined although it is also possible to add a new service file to /etc/firewalld/services with a name of the form <servicename>.service. These files are all in XML and should be easily understandable. Most of us will never need to modify or even look at these files. That is, unless you are running a Linux box as a router and play a lot of 90s games with their own weird, wild expectations for how networking should work.

# iptables

I personally prefer IPTables to manage my firewalls because of the relative simplicity of its starting rule sets. The netfilter rule types and syntax are identical to those used with firewalld, and I could use IPTables to create a complete set of rules identical to the firewalld defaults.

The advantage of firewalld is that it has already created the complex rule sets that implement the arbitrary concept of zones for use in Linux hosts used as firewalls and routers with multiple network connections. The advantage of IPTables is that it uses a much simpler set of rules to accomplish the same objectives in less complex environments such as workstations with a single connection to a local network. The end result in both cases is just as secure.

## Converting to IPTables

To convert to IPTables from firewalld is simple. We need to install one package, iptables-services, create a short set of rules, deactivate firewalld, and activate iptables.

## EXPERIMENT 17-10

Perform this experiment as root. Start by installing the iptables-services package. This package also includes a very brief and extremely secure set of rules for IPTables.

```
[root@studentvm1 ~]# dnf -y install iptables-services
```

Look at the contents of /etc/sysconfig/iptables.

```
[root@studentvm1 ~]# cat /etc/sysconfig/iptables
# sample configuration for iptables service
# you can edit this manually or use system-config-firewall
# please do not ask us to add additional ports/services to this default
configuration
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Note that this set of rules is very simple and it does allow SSH access. As a standard security step, we disable the network interface to prevent crackers from accessing our host while the firewall is down.

```
[root@studentvm1 ~]# ip link set enp0s3 down
```

Stop and disable firewalld, then start and enable iptables.

```
[root@studentvm1 ~]# systemctl stop firewalld ; systemctl disable firewalld
Removed /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
[root@studentvm1 ~]# systemctl start iptables ; systemctl enable iptables
```

```
Created symlink /etc/systemd/system/basic.target.wants/iptables.service → /
usr/lib/systemd/system/iptables.service.
[root@studentvm1 ~]#
```

Now check the current rule set.

```
[root@studentvm1 ~]# iptables-save
# Generated by iptables-save v1.8.0 on Thu Jun 13 16:01:53 2019
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [50:5676]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
# Completed on Thu Jun 13 16:01:53 2019
[root@studentvm1 ~]#
```

Connect to the localhost using SSH to verify that this works. Try connecting to localhost with Telnet.

## Understanding the rule set

IPTables rules are organized in chains. There are five predefined tables consisting of multiple chains of rules, but SysAdmins can also define their own chains. Each table has a specific purpose:

- **Filter**: The filter table is the one chain defined in the tiny default rule set. It is used to filter packets and to discard them or accept them. This chain is the one most commonly used in very simple firewalls.

- **NAT**: The NAT table is used for Network Address Translation. Internal private addresses like the 10.0.0.0/8 addresses used by our virtual router or the 192.168.0.0/16 range are not routable through the Internet. So outbound request packets to, for example, a web site must have the return IP address of the router at the edge of the internal network and the Internet. NAT substitutes the routable IP address in place of the non-routable one for outbound packets and the non-routable IP in place of the routable one for the return packets.
- **Mangle**: The mangle table is used to change – mangle – various portions of a packet. One example is to redefine the source IP address of the packet. Although such mangling does have legitimate uses, it can also be used by crackers to spoof the source address of packets as part of distributed denial of service (DDOS) attacks.
- **Raw**: This table would be used to configure exemptions to packet tracking rules.
- **Security**: This table would be used to implement Mandatory Access Control rules. It is generally used in conjunction with SELinux to enhance security.

In Figure 17-1, we have a breakdown of the lines in the default iptables file. We skip the comment lines which are ignored by IPTables.

| # | IPTables Line | Description |
|---|---|---|
| 1 | `*filter` | The first meaningful line not a comment simply states that the following rules are inserted into the filter table. |
| 2 | `:INPUT ACCEPT [0:0]` | This is a policy rule which accepts all packets on the input chain of the table. The [0:0] are counters for the numbers of transmitted and received packets for this chain. The other rules in the INPUT chain modify the default policy. |
| 3 | `:FORWARD ACCEPT [0:0]` | The forward chain is used in routers for forwarding packets to the correct network interface. This rule sets the default policy to accept. |
| 4 | `:OUTPUT ACCEPT [0:0]` | This policy rule accepts all packets outbound from the host. Should we desire to block outbound packets of a specific type, we can do that with the OUTPUT chain. |
| 5 | `-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT` | This rule works with stateful connections. It accepts all packets after the first one has been accepted by other rules. That is packets that belong to an already established connection and that are related to an existing connection.<br><br>All the rest of the rules in the INPUT chain are matched only on the first packet to initialize the connection. The rest of the packets are matched in any allowed connection are matched by this rule. |
| 6 | `-A INPUT -p icmp -j ACCEPT` | This entry accept all ICMP (Ping) requests thus allowing a response. |
| 7 | `-A INPUT -i lo -j ACCEPT` | Accepts packets from the localhost on interface lo. Without this we would not be able to SSH or Telnet to the localhost from the localhost. |
| 8 | `-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT` | This entry accepts the first packet of a new connection on port 22, SSH. This sets up a stateful connection all further packets of which can then be accepted by rule #5. |

***Figure 17-1.*** *A description of the rules in the IPTables firewall*

| # | IPTables Line | Description |
|---|---|---|
| 9 | `-A INPUT -j REJECT --reject-with icmp-host-prohibited` | This rule rejects all packets that don't match other rules. Essentially this rejects everything except ICMP and SSH packets because they have already matched other rules. |
| 10 | `-A FORWARD -j REJECT --reject-with icmp-host-prohibited` | This rule rejects all packets sent to the FORWARD chain of the filter table. |
| 11 | `COMMIT` | This line is not a rule. It is the last line of the filter table and causes the previous rules for this table to be committed to the active firewall rule set. |

***Figure 17-1.*** *(continued)*

Although IPv6 rules are contained in a different configuration file, ip6tables, the rules have the same syntax and uses. The primary difference is that IP addresses would be specified with IPv6 formats.

## Managing rules with IPTables

My preferred method for interacting with iptables is by using my favorite editor to make the changes I need to /etc/sysconfig/iptables. It is fast and easy this way. Part of the problem with using the **`iptables`** command is the need to specify the location of the new rule in the set of existing rules. That means a bit of counting to ensure the correct positioning. It is not really a major problem but just provides opportunities for errors to creep in.

Let's start with a simple change, adding a rule to allow Telnet connections.

### EXPERIMENT 17-11

Perform this experiment as root. In this experiment we append a rule to the INPUT chain that allows inbound Telnet packets. We will start by making a backup copy of the iptables file and then using the CLI command to append the new rule to the existing INPUT chain of the filter table.

```
[root@studentvm1 ~]# cp /etc/sysconfig/iptables /root
[root@studentvm1 ~]# iptables -A INPUT -p tcp -m state --state NEW -m tcp
--dport 23 -j ACCEPT
```

Verify the addition.

```
[root@studentvm1 ~]# iptables-save
# Generated by iptables-save v1.8.0 on Fri Jun 14 08:31:49 2019
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [32:3588]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A INPUT -p tcp -m state --state NEW -m tcp --dport 23 -j ACCEPT
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
# Completed on Fri Jun 14 08:31:49 2019
```

We could also list just the INPUT chain. The iptables command provides a bit different view of the running firewall rather than the rule sets.

```
[root@studentvm1 ~]# iptables -L INPUT
```

We can see that the new rule has been added, but will it work? Consider that for a moment and then read on.

It will not work. The reason is that the INPUT rule rejecting all unmatched packets would be encountered before the new rule which would match the first packet of the Telnet connection on port 23. The rule must appear before the general rejection rule.

Let's delete the new rule and then add it in the right place. First, we need to know the rule number in the INPUT chain. We count, starting with 1, and I come up with number 6. Do not count the INPUT chain policy rule. Your number for this rule may be different.

```
[root@studentvm1 ~]# iptables -D INPUT 6 ; iptables-save
# Generated by iptables-save v1.8.0 on Fri Jun 14 09:07:24 2019
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [1:88]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

```
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
# Completed on Fri Jun 14 09:07:24 2019
```

Now let's insert this rule as rule number 5. This will insert the new rule after rule number 4 and before the current rule number 5.

```
[root@studentvm1 ~]# iptables -I INPUT 5 -p tcp -m state --state NEW -m tcp
--dport 23 -j ACCEPT ; iptables-save
# Generated by iptables-save v1.8.0 on Fri Jun 14 09:09:59 2019
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [22:2400]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 23 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
# Completed on Fri Jun 14 09:09:59 2019
```

Now connect to localhost using Telnet to verify that this new rule is working. After testing, we need to save this new rule set. Adding the rules using the iptables command merely adds them to the running set in memory. You can view the /etc/sysconfig/iptables file to verify this. Save the current active rule set.

```
[root@studentvm1 ~]# iptables-save > /etc/sysconfig/iptables
```

IPTables provides the same functionality as firewalld but with a much more elegant simplicity. I find it easy to use and modify the rules.

---

We will explore firewalls, specifically IPTables, in more detail in the next book of this series.

# Fail2Ban

A dynamic firewall is one that can adapt as the threats change. I needed something like this to stem the large number of attacks via SSH I had been experiencing a few years ago. After a good bit of exploring and research, I found fail2ban, open source software which automates what I was previously doing manually.

Fail2Ban has a complex series of configurable matching rules and separate actions that can be taken when attempts are made to crack into a system. It has rules for many types of attacks that include Web, email, and many other services that might have vulnerabilities. Fail2Ban works by detecting attacks and then adding a rule to the firewall that will block further attempts from that specific, single IP address for a specified and configurable amount of time. After the time has expired, it removes the blocking rule.

Let's install Fail2Ban and see how it works.

**EXPERIMENT 17-12**

Perform this experiment as the root user. First, install Fail2Ban. This only takes a minute or so and does not require a reboot. The gamin package is a library of tools that monitor files for changes and can notify other programs, such as Fail2Ban, when a log file changes.

```
[root@studentvm1 ~]# dnf -y install fail2ban gamin
```

Fail2Ban is not started by the installation so we will need to do so after we do a bit of configuration. Make /etc/fail2ban the PWD and list the files there. The jail.conf file is the main configuration file, but it is not used for most configuration because it might get overwritten during an update. We will create a jail.local file in the same directory. Any settings defined in jail.local will override ones set in jail.conf.

Copy jail.conf to jail.local. Edit the jail.local file and delete the comment near the beginning that tells you not to modify this file. It is, after all, the one we will be modifying.

Find the line **# ignoreself = true**, remove the comment hash, and change it to **ignoreself = false**. We do this so that Fail2Ban will not ignore failed login attempts from the localhost.

Scroll down to the line **bantime = 10m** and change that to 1 minute. Since we have no other hosts to test from, we will test using localhost. We do not want the localhost banned for long so that we can resume experiments quickly. In the real world, I would set this to several hours so that the crackers cannot get more attempts for a long time.

Change **maxretry =** 5 to 2. This is the maximum number of retries allowed after any type of failed attempt. Two retries is a good number for experimental purposes. I normally set this to three because anyone failing three retries to get into my system using SSH does not belong there.

We could also change both of these configuration options in the [sshd] filter section which would limit them to sshd, while the original settings would apply to all other filters.

Read the comments for the other miscellaneous options in this section of the file, then scroll down to the [sshd] section in JAILS.

Add the highlighted line. The documentation is not clear about needing to add this line. In previous versions the line was **enabled = false** so it was clear that changing false to true would enable the sshd jail.

```
[sshd]

# To use more aggressive sshd modes set filter parameter "mode" in jail.
local:
# normal (default), ddos, extra or aggressive (combines all).
# See "tests/files/logs/sshd" or "filter.d/sshd.conf" for usage example and
details.
enabled = true
#mode   = normal
port    = ssh
logpath = %(sshd_log)s
backend = %(sshd_backend)s
```

Do not enable fail2ban, but start it.

```
[root@studentvm1 ~]# systemctl start fail2ban
```

Now ssh to localhost and log in using a bad user account or password on a good user account. It takes three failed attempts to log in, not three failed password entries. After three failed login attempts, the following error message is displayed.

```
[student@studentvm1 ~]$ ssh localhost
ssh: connect to host localhost port 22: Connection refused
```

This means that the sshd jail is working. Look at the active firewall rules. Remember that these fail2ban rules are stored in memory and are not added to the /etc/sysconfig/iptables file. There is a line in the following output that rejects connections from 192.168.0.1 which is

my physical host system, from which I also tried this. The IPTables rejection lines are removed after 1 minute so if you don't see that line, force the failed logins again.

```
[root@studentvm1 ~]# iptables-save
# Generated by iptables-save v1.8.0 on Sun Jun 16 21:24:59 2019
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [4:296]
:f2b-sshd - [0:0]
-A INPUT -p tcp -m multiport --dports 22 -j f2b-sshd
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 23 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
-A f2b-sshd -s 127.0.0.1/32 -j REJECT --reject-with icmp-port-unreachable
-A f2b-sshd -s 192.168.0.1/32 -j REJECT --reject-with icmp-port-unreachable
-A f2b-sshd -j RETURN
COMMIT
# Completed on Sun Jun 16 21:24:59 2019
[root@studentvm1 ~]#
```

Now let's look at a couple log files. In /var/log, first look at /var/log/secure. You should see a number of entries indicating failed passwords. These are the log entries checked by Fail2Ban for failures.

Look at the /var/log/fail2ban.log file. This log file shows the times that triggering entries were found in the secure log and the ban and unban actions taken to protect the system.

Be aware that the f2b-sshd chain entries do not appear in the IPTables rule set until the first time a ban is triggered. Once there, the first and last lines of the chain are not deleted, but the lines rejecting specific IP addresses are removed as they time out. It took me a bit of work to figure out this bit.

The installation of Fail2Ban installs the configuration files needed for logwatch to report on Fail2Ban activity. It is possible to create your own filters and actions for Fail2Ban, but that is beyond the scope of this course.

# PAM

PAM stands for pluggable authentication modules. It is a key component of security on Linux hosts and provides a dynamic and flexible means to manage user access to their accounts and resources.

PAM divides the function of authentication into four parts:

1. Account management determines things like whether the user's password is expired or locked and whether the user is authorized to access a particular service.
2. Authentication management is the task of authenticating the user, as in verifying that the password and user ID are correct. This can be extended to include biometric authentication and smart card hardware and methods.
3. Password management is used in the process of password updates.
4. Session management is used to enable user access to services such as their home directory, resource allocation, and deals with logging for audit trails.

The PAM man page has a good explanation of PAM and references to other resources. The man page indicates that it is not necessary for a SysAdmin to understand the internal workings of the PAM libraries that implement this tool. The reason for this is that the configuration file /etc/pam.conf (if it exists, which it does not in Fedora 29 and 30) and the files located in the /etc/pam.d directory are the tools used to configure PAM.

The PAM configuration items most likely to be used are mostly related to resource management such as specifying limits on the CPU time, memory, and number of processes that specific users or groups may consume. This capability can be used to aid in the allocation of limited resources to those who have more need or are authorized such access.

Of course many of today's Linux hosts have huge amounts of all of the resources required on a modern computer system. Even then certain environments such as development, test, large database systems, high-performance computer (HPC), high-traffic web sites, and others may have contention among users and their running tasks for one or more system resources. And, of course, users in many organizations don't have access to the huge systems that many of us do, which again leads to contention for scarce resource.

There are also the individual users who manage to suck up as much of any resources as they are allowed. This seems to be especially true on systems with resources already strained to the limit. We have already explored setting resource limits and password quality restrictions in Chapter 16 but did not link the /etc/security/limits.conf file to PAM. It is, in fact, PAM that deals with enforcing any limits or other configurations we set with the files in the /etc/security directory.

## Some basic steps

There are some steps that can be taken for any Linux host to harden it against attacks of many different types. These steps range from easy to difficult, and, as mentioned previously, the ones you choose to put in place depend upon the amount of pain you would be in should the defenses of your systems be breached. The cost trade-off is a judgment that must be made by those at each installation, but I recommend taking as many of these steps as possible.

Some of these steps are included here for the sake of completeness but will not be covered here. Some of those will be covered in Volume 3:

1. Limit physical access to prevent unauthorized passersby from inserting malware via a USB thumb drive or just pocketing one that is already sitting there. It can also prevent curious fingers from pushing reset buttons.
2. Strong passwords are a simple security measure and easily enforced as we have seen. This makes brute force cracking of passwords much more difficult.
3. Change passwords frequently to ensure that any that are cracked are not usable for more than a short period of time. Password aging can be used to enforce this.

4. Do not share user accounts. When multiple users have access to a common account, it becomes more difficult to determine the user responsible for security problems. If users must collaborate on shared documents, create a shared directory separate from their own home directories and use a separate group to allow access to only the people who need it. We covered shared directories and files in Chapter 18 of Volume 1.
5. Deleting old user accounts is important in keeping a system secure. Old, supposedly unused, accounts can be used to gain access to a system. Cleaning out the cruft is a good security practice.
6. Strong firewalls are always an important part of any security regimen.
7. Use public/private keypairs (PPKP) with SSH. These are stronger than passwords and cannot be memorized so cannot be divulged under duress. (Yes, that happens!)
8. Do not store sensitive data on computers that are firewalls or routers that are directly accessible to the Internet.
9. In larger organizations data should not be stored on any host in the DMZ.[7]
10. Intrusion detection can be used to detect when an intrusion has occurred, hopefully before and damage has been done.
11. Verify open ports with tools like nmap. There should be no open ports that you are not expecting and that are not consistent with the services that you want exposed to the outside world.
12. Use a BIOS password to prevent changes to the hardware boot sequence.

[7]DMZ – A network segment that contains servers that respond to external requests for web pages and so on but in which no data is stored. All data is stored in a more secure network with another set of firewalls between it and the DMZ.

13. Use a GRUB password to prevent changes to Linux initialization and startup.
14. Turn off or remove unused services to prevent attacks against any possible known vulnerabilities in those services.
15. Use firewalls to limit in and outbound traffic to only what would be expected on a given host.
16. Use SELinux to prevent crackers from making changes even if they do gain access to a host. This is an advanced tool when used to do more than warn of potential problems and can be a bit of a nuisance to work around when doing updates or adding new software. It does provide very strong protection by preventing potential system alteration rather than just reporting the changes after the fact.
17. Use intrusion detection software like Tripwire to report altered files and other signs of a successful or attempted intrusion.
18. Disable ZEROCONF (Zero Configuration) a network self-configuration program when static configuration has not been performed and DHCP is not available. It is on by default in earlier Fedora releases. This service is sometimes known as Avahi. I always remove the avahi package and its dependencies.
19. Sync all system times using NTP to make it easier to compare log files.
20. Only allow root to run cron jobs.
21. Enable only ssh2 protocol which is the default in Fedora and other Red Hat-based distros.
22. Do not allow root logins, especially remote ones. Log in as non-root user and then su to root.

23. Real SysAdmins don't use sudo. Don't use sudo yourself as the SysAdmin. I discuss this in Chapter 11, in my book, *The Linux Philosophy for SysAdmins*,[8] and in an excerpt[9] from that book on my web site.

24. If a non-root user really does need access to a command that requires root privilege, configure sudo for that one user to use that one command.

25. Back up everything – frequently. This is so important that it has a chapter of its own and we explore it in Chapter 18.

# Chapter summary

Security is a big part of our job as SysAdmins.

Realistic security is a cost/benefit trade-off between the owner of the computer network and the cracker. The question is, how much is our data worth? Security should be a pain for users and they will likely complain, but it should not be enough to engender bad behavior such as writing down passwords and sticking them under the keyboard.

A very high level of security can be achieved quite easily with Linux. Using good passwords, a simple firewall, and the Fail2Ban program can create a very secure environment with little trouble and no software costs.

We will look at more active security measures, such as SELinux, root kit hunters, and intrusion detection, in Volume 3.

# Exercises

Perform the following exercises to complete this chapter:

1. What is the best way to protect against any type of login attacks such as dictionary-based attacks?

2. What is the second best way to protect against any type of login attacks such as dictionary-based attacks?

[8]Both, David, *The Linux Philosophy for SysAdmins*, Apress, 2018, 375

[9]Both, David, *Real SysAdmins don't sudo – Book excerpt*, www.both.org/?p=960

3. In Experiment 17-2, the ping and tcpdump commands both display the hostname “router” as part of their output, instead of or in addition to the IP address of the router, 10.0.2.1. Why does this happen?
4. Use tcpdump to monitor the network traffic your studentvm1 host generates when you use a browser to connect to a remote web page such as www.example.com. Look at the content as well as the headers.
5. What is the result of setting the default policy of the INPUT chain of the filter table to REJECT?
6. What is the function of Avahi?
7. Is the Avahi daemon running on your StudentVM1 host? If so, remove it.
8. Use the logwatch program to view report on Fail2Ban activity.
9. Are there any easy steps left that you can take to improve security on your StudentVM1 instance? If so, take them.

CHAPTER 18

# Backup Everything – Frequently

In this chapter you will learn

- Why backups are important
- How to use S.M.A.R.T. to predict hard drive failures before they occur
- How to create simple backups using the **tar** command
- How to devise a simple backup strategy

## Introduction

Nothing can ever go wrong with my computer and I will never lose my data. <sarcasm>Right</sarcasm>.

I have experienced data loss for a myriad of reasons, many of them my own fault. Keeping decent backups has always enabled me to continue with minimal interruption. In Chapter 17 of this volume, we looked at some of the many ways in which data can be lost, compromised, or corrupted. This chapter discusses one method for preventing catastrophic data loss and facilitating easy recovery.

## Backups to the rescue

Recently, very recently – while I was working on this book, actually – I encountered a problem in the form of a hard drive crash that destroyed the data in my home directory. I had been expecting this for some time so it came as no surprise.




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4_18

# The problem

The first indication I had that something was wrong was a series of emails from the S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology) enabled hard drive on which my home directory resided.[1] Each of these emails indicated that one or more sectors had become defective and that the defective sectors had been taken offline and reserved sectors allocated in their place. This is normal operation; hard drives are designed intentionally with reserved sectors for just this reason.

I first used the **`smartctl`** command to view the internal statistics for the hard drive in question. The original, defective hard drive has been replaced but, yes, I keep some old, defective devices for teachable moments like this. I installed this damaged hard drive in my docking station to demonstrate what the results of a defective hard drive look like.

You can perform this experiment along with me, but your results will be different – hopefully healthier than my defective drive.

The S.M.A.R.T. reports used in Experiment 18-1 can be a bit confusing. The web page "Understanding S.M.A.R.T. Reports[2]" can help somewhat with that. Wikipedia also has an interesting page on this technology.[3] I recommend reading those documents before attempting to interpret the S.M.A.R.T. results; they can be very confusing.

**EXPERIMENT 18-1**

This experiment must be performed as root. It will work better on a physical Linux host, but you can follow along just the same.

After installing the drive in the docking station and turning it on, the dmesg command showed the drive to be assigned as device special file /dev/sdi. Be sure to use the correct device special file for your hard drive. You can use any physical hard drive installed in your host, even if it is in use.

I have divided the results of the command into sections for easier reference during the discussion, and I have removed a large amount of irrelevant data. You should use /dev/sda for the hard drive on your VM.

[1]Your host must have a mail transfer agent (MTA) such as SendMail installed and running. The /etc/aliases file must have an entry to send root's email to your email address.

[2]*Understanding SMART Reports*, https://lime-technology.com/wiki/Understanding_SMART_Reports

[3]Wikipedia, *SMART*, https://en.wikipedia.org/wiki/SMART

```
[root@david ~]# smartctl -x /dev/sd1 | less
smartctl 6.5 2016-05-07 r4318 [x86_64-linux-4.15.6-300.fc27.x86_64] (local
build)
Copyright (C) 2002-16, Bruce Allen, Christian Franke, www.smartmontools.org

=== START OF INFORMATION SECTION ===
Model Family:     Seagate Barracuda 7200.11
Device Model:     ST31500341AS
Serial Number:    9VS2F303
LU WWN Device Id: 5 000c50 01572aacc
Firmware Version: CC1H
User Capacity:    1,500,301,910,016 bytes [1.50 TB]
Sector Size:      512 bytes logical/physical
Rotation Rate:    7200 rpm
Device is:        In smartctl database [for details use: -P show]
ATA Version is:   ATA8-ACS T13/1699-D revision 4
SATA Version is:  SATA 2.6, 3.0 Gb/s
Local Time is:    Wed Mar 14 14:19:03 2018 EDT
SMART support is: Available - device has SMART capability.
SMART support is: Enabled
AAM level is:     0 (vendor specific), recommended: 254
APM feature is:   Unavailable
Rd look-ahead is: Enabled
Write cache is:   Enabled
ATA Security is:  Disabled, NOT FROZEN [SEC1]
Wt Cache Reorder: Unknown
=== START OF READ SMART DATA SECTION ===
SMART Status not supported: Incomplete response, ATA output registers missing
SMART overall-health self-assessment test result: PASSED
Warning: This result is based on an Attribute check.
```

The first section of results, shown just earlier, provides basic information about the hard drive capabilities and attributes such as brand, model, and serial number. This is interesting and good information to have, but it is all you will see on your VM.

This section shows that this S.M.A.R.T. data report must be taken with a bit of skepticism. Notice that my known defective drive has passed the self-assessment test. That appears to mean that the drive is not about to fail catastrophically even though it already has.

The data we are most interested in at present is in the next two sections. Notice that I have trimmed out a great deal of the information not essential to this experiment.

```
=== START OF READ SMART DATA SECTION ===
<snip – removed list of SMART capabilities.>

SMART Attributes Data Structure revision number: 10
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAGS    VALUE WORST THRESH FAIL RAW_VALUE
  1 Raw_Read_Error_Rate     POSR--   116   086   006    -    107067871
  3 Spin_Up_Time            PO----   099   099   000    -    0
  4 Start_Stop_Count        -O--CK   100   100   020    -    279
  5 Reallocated_Sector_Ct   PO--CK   048   048   036    -    2143
  7 Seek_Error_Rate         POSR--   085   060   030    -    365075805
  9 Power_On_Hours          -O--CK   019   019   000    -    71783
 10 Spin_Retry_Count        PO--C-   100   100   097    -    0
 12 Power_Cycle_Count       -O--CK   100   100   020    -    279
184 End-to-End_Error        -O--CK   100   100   099    -    0
187 Reported_Uncorrect      -O--CK   001   001   000    -    1358
188 Command_Timeout         -O--CK   100   098   000    -    12885622796
189 High_Fly_Writes         -O-RCK   001   001   000    -    154
190 Airflow_Temperature_Cel -O---K   071   052   045    -    29 (Min/Max
22/29)
194 Temperature_Celsius     -O---K   029   048   000    -    29 (0 22 0 0 0)
195 Hardware_ECC_Recovered  -O-RC-   039   014   000    -    107067871
197 Current_Pending_Sector  -O--C-   100   100   000    -    0
198 Offline_Uncorrectable   ----C-   100   100   000    -    0
199 UDMA_CRC_Error_Count    -OSRCK   200   200   000    -    20
240 Head_Flying_Hours       ------   100   253   000    -    71781 (50 96 0)
241 Total_LBAs_Written      ------   100   253   000    -    2059064490
242 Total_LBAs_Read         ------   100   253   000    -    260980229
                            ||||||_ K auto-keep
                            |||||__ C event count
                            ||||___ R error rate
                            |||____ S speed/performance
                            ||_____ O updated online
                            |______ P prefailure warning
```

The preceding section of results from the **smartctl** command displays raw data accumulated in the hardware registers on the drive. The raw values are not particularly helpful for some of the error rates; as you can see, some of the numbers are clearly bogus. The "Value" column is usually more helpful. Read the referenced web pages to understand a bit about why. In general, numbers like 100 in the Value column mean 100% good and low numbers like 001 mean close to failure – sort of 99% of the useful life is used up. It is really very strange.

In this case, 048 in the Value column for Reallocated_Sector_Ct – Reallocated Sector Count – sort of might mean that about half of the sectors allocated for reallocation have been used up.

The number 001 for Reported_Uncorrect – reported defective sectors that are not correctable – and High_Fly_Writes (writes in which the heads were flying further off the recording surface of the hard drive than is optimal) means that the life of this hard drive is effectively over. This has been shown to be the case with empirical evidence.

This next section actually lists errors and information about them when they occur. This is the most helpful part of the output. I do not try to analyze every error; I simply look to see if there are multiple errors. The number 1350, in the first line of the following code, is the total number of errors detected on this hard drive.

```
<Snip>

Error 1350 [9] occurred at disk power-on lifetime: 2257 hours (94 days + 1 hours)
 When the command that caused the error occurred, the device was active or idle.

 After command completion occurred, registers were:
 ER -- ST COUNT  LBA_48  LH LM LL DV DC
 -- -- -- == -- == == == -- -- -- -- --
 40 -- 51 00 00 00 04 ed 00 14 59 00 00  Error: UNC at LBA = 0x4ed001459 =
21156074585

 Commands leading to the command that caused the error were:
 CR FEATR COUNT  LBA_48  LH LM LL DV DC  Powered_Up_Time  Command/Feature_Name
 -- == -- == -- == == == -- -- -- -- --  ---------------  --------------------
 60 00 00 00 08 00 04 ed 00 14 58 40 00 11d+10:44:56.878  READ FPDMA QUEUED
 27 00 00 00 00 00 00 00 00 00 00 e0 00 11d+10:44:56.851  READ NATIVE MAX ADDRESS
                                                          EXT [OBS-ACS-3]
 ec 00 00 00 00 00 00 00 00 00 00 a0 00 11d+10:44:56.849  IDENTIFY DEVICE
 ef 00 03 00 46 00 00 00 00 00 00 a0 00 11d+10:44:56.836  SET FEATURES [Set
                                                          transfer mode]
```

```
 27 00 00 00 00 00 00 00 00 00 00 e0 00 11d+10:44:56.809  READ NATIVE MAX
ADDRESS EXT [OBS-ACS-3]

Error 1349 [8] occurred at disk power-on lifetime: 2257 hours (94 days + 1 hours)
 When the command that caused the error occurred, the device was active or idle.

 After command completion occurred, registers were:
 ER -- ST COUNT  LBA_48  LH LM LL DV DC
 -- -- -- == -- == == == -- -- -- -- --
 40 -- 51 00 00 00 04 ed 00 14 59 00 00  Error: UNC at LBA = 0x4ed001459 =
21156074585

 Commands leading to the command that caused the error were:
 CR FEATR COUNT  LBA_48  LH LM LL DV DC  Powered_Up_Time  Command/Feature_Name
 -- == -- == -- == == == -- -- -- -- --  ---------------  --------------------
 60 00 00 00 08 00 04 ed 00 14 58 40 00 11d+10:44:53.953  READ FPDMA QUEUED
 60 00 00 00 08 00 04 f4 00 14 10 40 00 11d+10:44:53.890  READ FPDMA QUEUED
 60 00 00 00 10 00 04 f4 00 14 00 40 00 11d+10:44:53.887  READ FPDMA QUEUED
 60 00 00 00 10 00 04 f3 00 14 f0 40 00 11d+10:44:53.886  READ FPDMA QUEUED
 60 00 00 00 10 00 04 f3 00 14 e0 40 00 11d+10:44:53.886  READ FPDMA QUEUED

Error 1348 [7] occurred at disk power-on lifetime: 2257 hours (94 days + 1 hours)
 When the command that caused the error occurred, the device was active or idle.

 After command completion occurred, registers were:
 ER -- ST COUNT  LBA_48  LH LM LL DV DC
 -- -- -- == -- == == == -- -- -- -- --
 40 -- 51 00 00 00 04 ed 00 14 59 00 00  Error: UNC at LBA = 0x4ed001459 =
21156074585

 Commands leading to the command that caused the error were:
 CR FEATR COUNT  LBA_48  LH LM LL DV DC  Powered_Up_Time  Command/Feature_Name
 -- == -- == -- == == == -- -- -- -- --  ---------------  --------------------
 60 00 00 00 08 00 04 ed 00 14 58 40 00 11d+10:44:50.892  READ FPDMA QUEUED
 27 00 00 00 00 00 00 00 00 00 00 e0 00 11d+10:44:50.865  READ NATIVE MAX
                                                          ADDRESS EXT [OBS-
                                                          ACS-3]
 ec 00 00 00 00 00 00 00 00 00 00 a0 00 11d+10:44:50.863  IDENTIFY DEVICE
 ef 00 03 00 46 00 00 00 00 00 00 a0 00 11d+10:44:50.850  SET FEATURES [Set
                                                          transfer mode]
```

```
  27 00 00 00 00 00 00 00 00 00 00 e0 00 11d+10:44:50.823  READ NATIVE MAX
                                                           ADDRESS EXT [OBS-
                                                           ACS-3]

Error 1347 [6] occurred at disk power-on lifetime: 2257 hours (94 days + 1 hours)
 When the command that caused the error occurred, the device was active or idle.

<Snip - removed many redundant error listings>
```

These errors are indicative that something really is wrong with the disk. Hopefully you won't have any errors on your virtual disk.

Because I am naturally very curious, I decided I would wait to see what else occurred before I replaced the hard drive. The failure numbers were not bad in the beginning. The error count rose to 1350 at the time of the catastrophic failure.

Some testing of over 67,800 S.M.A.R.T. drives[4] by a cloud company named Backblaze provides some statistically based insight into failure rates of hard drives that experienced various numbers of reported errors. This web page is the first I have found that demonstrates a statistically relevant correlation between reported S.M.A.R.T. errors and actual failure rates. Their web page also helped improve my understanding of the five S.M.A.R.T. attributes that they found should be closely monitored.

In my opinion, the bottom line of the Backblaze analysis is that hard drives should be replaced as soon as possible after they begin to experience error reports in any of the five statistics they recommend monitoring. My experience seems to confirm that although it was not even close to being statistically significant. My drive failed within a couple months of the first indications that there was a problem. The number of errors my drive experienced before failing beyond recovery is very high, and I had been very lucky to have been able to recover from several errors that caused the /home filesystem to switch to read-only (ro) mode. This only occurs when Linux determines that the filesystem is unstable and cannot be trusted.

[4]BackBlaze, Web site, *What SMART Stats Tell Us About Hard Drives*, www.backblaze.com/blog/what-SMART-stats-indicate-hard-drive-failures/

# Backup options

There are many options for performing backups. In addition to old favorites like **tar**, most Linux distributions are provided with one or more additional open source programs especially designed to perform backups. There are many commercial options available as well. However, fancy and expensive backup programs are not really necessary to design and implement a viable backup program.

# tar

The **tar** command is used to make archives, more commonly referred to today as backups. The name "tar" stands for Tape ARchive, but it can be used with any type of recording media such as tape, hard drives, thumb drives, and more.

The tar command is simple and easy, but it does have a few things to watch for. It can be used quite effectively by non-root users to create their own backups.

**EXPERIMENT 18-2**

Perform this experiment as the student user. We will use tar to create a backup of the student home directory on the localhost to the student.tar file – tar files and their compressed versions are also called tarballs – in the /tmp directory. The -c option creates a new tarball, and the -v option indicates verbose mode which prints the name and path of every file placed in the tarball. The -f option specifies the file name and path for the tarball being created. The final dot is the directory being archived and the dot (.) is our shortcut for specifying the current directory. The tar command archives all subdirectories of the specified directory.

```
[student@studentvm1 ~]$ tar -cvf /tmp/student.tar .
```

This command created a tar file named student.tar in the /tmp directory. That file is a backup of everything in the home directory. Well, that's nice, but also not very interesting because it is very common.

Let's look in the tarball we just created. There are a couple ways to do this. First we can use tar to list the table of contents (TOC) of the tarball. This just lists the file names and their attributes, not their contents. The contents of your directory will be different.

The -t option displays the table of contents for the tarball, and -f is the input file name. The -v option again specifies verbose.

```
[student@studentvm1 ~]$ tar -tvf /tmp/student.tar
drwx------ student/student   0 2019-06-13 15:41 ./
-rw-rw-r-- student/student 41936 2019-01-16 14:08 ./dmesg1.txt
drwxrwxr-x student/student     0 2018-12-30 16:36 ./testdir6/
drwxrwxr-x student/student     0 2019-03-14 16:00 ./.fvwm/
-rw------- student/student    60 2019-03-14 16:00 ./.fvwm/.FvwmConsole-History
-rw-r--r-- student/student    18 2018-10-08 09:41 ./.bash_logout
drwxrwxr-x student/student     0 2019-03-02 08:21 ./chapter6/
-rw-rw-r-- student/student   614 2019-02-23 13:10 ./chapter6/Experiment_6-1.txt
-rw-rw-r-- student/student 177878 2019-03-01 08:54 ./chapter6/Experiment_6-3.txt
-rw-r--r-- student/student    839 2019-06-13 15:41 ./.bashrc
drwxr-xr-x student/student      0 2018-12-22 13:15 ./Public/
<snip>
drwxrwxr-x student/student        0 2018-12-30 16:36 ./testdir1/testdir2/
testdir3/testdir4/testdir5/
-rw------- student/student    33523 2019-06-13 15:41 ./.bash_history
-rw-r----- student/student        5 2019-06-13 15:36 ./.vboxclient-seamless.pid
drwxr-xr-x student/student        0 2018-12-22 13:15 ./Music/
-rw-r----- student/student        5 2019-06-13 15:36 ./.vboxclient-clipboard.pid
-rw-rw-r-- student/student        0 2019-04-02 08:50 ./umask.test
-rwxr-xr-x student/student       92 2019-03-21 08:34 ./cpuHog.Linux
lrwxrwxrwx student/student        0 2018-12-30 16:48 ./softlink1 -> link1
drwxr-xr-x student/student        0 2018-12-22 13:15 ./Videos/
-rw-rw-r-- student/student    41876 2018-12-30 16:37 ./dmesg3.txt
-rw------- student/student     2935 2019-06-13 15:36 ./.xsession-errors
drwxr-xr-x student/student        0 2018-12-22 13:15 ./Templates/
-rw-rw-r-- student/student        0 2019-04-06 21:26 ./Downtown
-rw-rw-r-- student/student    41876 2018-12-30 16:37 ./dmesg2.txt
```

Now we have a simple backup and have verified its contents. Let's delete a single file and restore it from the tarball we just created. Let's just do a simple restore. Delete and restore the ~./cpuHog.Linux file.

```
[student@studentvm1 ~]$ rm cpuHog.Linux
```

Verify that the file is no longer present before continuing. Now let's verify that the cpuHog. Linux file is present in the tarball.

```
[student@studentvm1 ~]$ tar -tvf /tmp/student.tar | grep cpuHog
-rwxr-xr-x student/student      97 2019-03-20 15:58 ./cpuHog.dos
-rwxr-xr-x student/student      92 2019-03-21 09:19 ./cpuHog.mac
-rwxr-xr-x student/student       92 2019-03-20 15:53 ./cpuHog
-rw------- student/student    20169 2019-03-21 15:27 ./chapter26/cpuHog.pdf
-rwxr-xr-x student/student       92 2019-03-21 08:34 ./cpuHog.Linux
```

We see all of the cpuHog files that match the pattern. Note that we must specify the path, in this case the path relative to the directory in which the file was located.

We can also use the -d option to get a list of differences between the tarball and the filesystem.

```
[student@studentvm1 ~]$ tar -df /tmp/student.tar .
tar: ./cpuHog.Linux: Warning: Cannot stat: No such file or directory
```

Now let's restore the file. The -x option extracts the desired file or files from the tarball.

```
[student@studentvm1 ~]$ tar -xvf /tmp/student.tar ./cpuHog.Linux
./cpuHog.Linux
```

List the contents of the home directory to ensure that the file has been restored. Make ~/tmp the PWD. Then extract the file again using the same command to illustrate a problem to consider when restoring files.

```
[student@studentvm1 ~]$ cd ~/tmp ; tar -xvf /tmp/student.tar ./cpuHog.Linux ; ll
./cpuHog.Linux
total 12
-rwxr-xr-x 1 student student 92 Mar 21 08:34 cpuHog.Linux
-rw-rw-r-- 4 student student 12 Apr  2 12:32 file2.txt
-rw-rw-r-- 4 student student 12 Apr  2 12:32 file.txt
[student@studentvm1 tmp]$
```

Note that the file is extracted to the current directory. Let's restore a file in a subdirectory of the student's home directory, ./Documents/file09, but let's keep ~/tmp as the PWD.

```
[student@studentvm1 tmp]$ tar -tvf /tmp/student.tar | grep file09
-rw-rw-r-- student/student       13 2018-12-30 16:33 ./Documents/file09
-rw-rw-r-- student/student    41876 2018-12-30 16:32 ./Documents/testfile09
```

```
[student@studentvm1 tmp]$ tar -xvf /tmp/student.tar file09
tar: file09: Not found in archive
tar: Exiting with failure status due to previous errors
```

Note that we must specify the exact file we want including the path as it appears in the tarball.

```
[student@studentvm1 tmp]$ tar -xvf /tmp/student.tar ./Documents/file09
./Documents/file09
[student@studentvm1 tmp]$ ll
total 16
-rwxr-xr-x 1 student student   92 Mar 21 08:34 cpuHog.Linux
drwxrwxr-x 2 student student 4096 Jun 17 13:40 Documents
-rw-rw-r-- 4 student student   12 Apr  2 12:32 file2.txt
-rw-rw-r-- 4 student student   12 Apr  2 12:32 file.txt
```

The Documents directory has been created here along with file09 contained in it. One of the important lessons to learn about using the **tar** command is that the files specified to be restored are extracted into the PWD. In order to restore any file to its proper location, the PWD during extraction must be the directory specified in the original command that created the tarball.

If the target output file is not specified using the -f option, the output of the tar command is sent directly to STDOUT so we can redirect it to a file.

```
[student@studentvm1 ~]$ cd ; tar -cv . > /tmp/tarball2.tar
```

This command performs the same function as the first tar command in this section, just in a somewhat different and more interesting manner.

---

So far we have done this as the student user. There are files such as configuration files that need to be backed up too. Non-root users do not have the access required to archive most system configuration files such as those in /tmp.

Part of our job as SysAdmins is to ensure that we back up – archive – everything that needs to be preserved in case of disaster. That includes the entire /home filesystem and configuration files in /etc/. Although we could just target specific files to archive from /etc and other configuration files, I think it is best to archive the entire /etc directory structure and then I have all possible files I might ever need to restore. I can just restore the ones I want if they are all there, and there is no danger of not having selected the file I need.

Let's now do a backup as the root user and back up a bit more.

## EXPERIMENT 18-3

Perform this experiment as root. We will back up the /home and /etc directories to a tarball in /tmp. Remember when we made it 5GB in size. That space will be useful for this.

Be sure to use the -p option to preserve file permissions and ownership. This should be done for both creating the archive and extracting the files from it. Archiving multiple directories just requires listing each one as part of the final arguments of the **tar** command. Also use the time utility to get a feel for how long these backups take on our VMs.

```
[root@studentvm1 ~]# time tar -cvpf /tmp/backup.tar /etc /home /root
```

The resulting tarball which took about 1 second to create on my VM was about 100MB in size. One of the things we can do is to compress the data to save space. So let's do that and compare the results.

```
[root@studentvm1 ~]# time tar -czvpf /tmp/backup.tgz /etc /home /root
```

Using compression took about 4.5 seconds on my VM, but it reduced the file size to about 32 MB, a reduction of two-thirds of the original.

Now use one pane of Midnight Commander (MC) to view the contents of one or both of the tarballs. Just highlight the tarball with the Midnight Commander cursor and press **Enter**. You can then navigate around the contents of the tarball just as if it were a filesystem on your VM – and in one sense it is because it is just contained inside a single archive file.

In the other pane of MC, navigate to the /tmp directory if it is not already the PWD for that pane. Locate the archived file /etc/sysconfig/iptables and copy it to /tmp using the F5 key.

It is very easy to navigate through archive files using MC and many other file managers in order to find and extract a single file.

In MC, return to root's home directory in both panes and exit from MC.

## Off-site backups

Creating good backups is an important first step in a backup strategy. Keeping the resulting backup media in the same physical location as your original data is a mistake although we have done that in the previous experiments for experimental purposes.

We have seen that theft of a computer that has all its backups on an internal drive can result in the complete and irrecoverable loss of important data. Fire and other disasters can also result in the loss of original data and the backup data if it is stored in the same location. Fireproof safes are one option that can reduce the threat from both theft and disaster like fire. Such safes are usually rated in minutes at specified temperatures for which they are supposed to protect their contents. I guess my personal concern here is that I have no idea how long or hot a fire will burn. Perhaps the safe will hold out long enough, but what if it does not?

I prefer to do for my own backups what the large companies do. I keep current off-site backups. For me this is in the safe deposit box at my bank. For others this might be "in the cloud" somewhere. I like the end-to-end control I have with my safe deposit box solution. I know it is well protected. If my little home office is destroyed, the bank is likely far enough away that it will not be affected by whatever disaster occurred.

For large companies there are services that store your backups in a remote, high-security location with climate-controlled vaults. Most of these services will even send armored trucks to your facilities to pick up and transport your backup media. Some provide high-speed network connections so that backups can be made directly onto their own storage media at their remote locations.

Many people and organizations are making backups to the cloud these days. I have serious reservations about the so-called cloud. First, "cloud" is just another word for someone else's computer. Second, considering the number of hacks into allegedly secure computing facilities that I have been reading about, I am not likely to trust my data to any external organization that maintains online backups accessible from the Internet. I would much prefer my remote backup data to be offline until I need it.

The concern I have with the cloud is that, aside from the marketing information the providers put on their web sites, there is no way for me to actually know whether their security measures are better than I can do for myself. Perhaps they can, but as a SysAdmin, I would like some proof of this. I have no doubt that a good portion of the cloud providers can do a better job of managing the security of the data entrusted to them than many businesses and individuals do. How do I know which ones those are? Remember that we are talking about cloud-based backup solutions, not application or web presence solutions.

What I think I can say with some level of confidence is that the established and recognized cloud providers, such as Amazon, Azure, Google, and others, are certainly more trustworthy when it comes to security than are many small or medium organizations. I am thinking about the ones don't have a full-time SysAdmin or outsource IT to small, local companies that are not especially reputable. I also think that many less experienced SysAdmins are not ready to deal with the high level of security required on the Internet in today's world of constant cyberattack.

So for many organizations, the cloud may be a viable option. For others, an experienced and knowledgeable SysAdmin may be the best choice. As with many IT decisions, it is a matter of weighing the risk factors and determining how much you are willing to accept.

## Disaster recovery services

Taking backups a step further, some of the places I have worked maintained a contract with one or more disaster recovery services. This type of service is paid to maintain a complete computer and network environment that can replace your own on a moment's notice. This usually includes everything from mainframes down to Intel-based servers and workstations. This is, of course, in addition to keeping massive amounts of data in off-site backup storage.

At one of the places I worked, we had quarterly assessments of our disaster recovery plan. We shut down all of the computers from the mainframes through the Intel servers. We notified the disaster recovery company that we were conducting a test, and they prepared their site with the various computers we would require to get back up and operational. We had the backup storage service transport the latest backup media from their secure facility in Raleigh, NC, to the recovery site in Philadelphia.

A group of folks from our offices traveled up to the recovery site and restored all of the data from our backup media, brought everything online, and tested to ensure that everything was working properly.

There were always problems. Always. But that was the whole point of the exercise – to find the problems with our strategies and procedures. And then to fix them.

## Options

Not everyone needs a disaster recovery service or huge amounts of backup data storage. For some individuals and very small businesses with only a single computer, a couple USB thumb drives and a manual backup to one of those drives is more than sufficient. For others, a relatively small external USB hard drive works well.

For my own needs, I use several 4TB external USB 3.0 hard drives and rotate them each week. The most current backup goes to my safe deposit box, and the one in the box comes home and goes back into the rotation. I also have a 4TB SATA hard drive in my main workstation that I also back up to every night. This means I always have the most recent backup right on line where and when I need it most. Two complete sets of backups every day works best for me. Of course I have seen so many ways to lose data that I am quite paranoid about it.

It is all in what you need for your circumstances.

## What about the "frequently" part?

What does this really mean? Because it actually opens up a wide range of questions:

- What does "frequently" really mean?
- What does "full" mean?
- If I have 24TB of movies on my NAS,[5] do I make a full backup of those every day or just a diff?[6]
- What should I think about in terms of setting up a cron job for a full backup?
- What if my system is off when it's supposed to run?

[5]Network Attached Storage

[6]Changes to files between one backup and the next. The difference

## How frequent is "frequently?"

Let's start with this question because it is part of the chapter title, after all. Always make a backup at least once every day. No matter what.

We are talking about the absolute minimum requirement for any system administered by a Linux SysAdmin. This means that the most work that could be lost is 24 hours or less. This frequency will be sufficient for many office and even development environments.

In many other cases, a once daily frequency will not be nearly enough. Think about banks, stock markets, high-intensity agile development environments, and scientific data collection and processing such as weather prediction. All of these environments require almost near-instantaneous duplication of huge amounts of data in case of hardware failures and, even more, minute-to-minute backups of the already duplicated data.

These high-performance requirements can be met, at least partially, using various forms and combinations of RAID arrays, high-availability network storage devices, cloud storage, and remote storage. Those solutions are outside the scope of this course.

## What does "full" really mean?

This should include all of the files you would need to rapidly recover from a major disaster. A full backup will obviously cover all data files, but that should not be all. A complete and quick recovery also means making backups of system configuration files and other system-level data.

So where are your data files? I back up the entire /home directory, thus ensuring I can restore everything including the user application data as well as user-level configuration files.

For the system, I also make backups of the entire /etc directory which contains configuration files for almost every system-level tool. I have used this backup data many times to recover from various self-inflicted forms of data loss. I do the entire /etc directory because one never knows what data will need to such be restored after an oopsy.

When creating a backup strategy for servers, we also want to make sure we include the appropriate configuration and user data. For example, the MariaDB database stores some configuration data in etc and user data such as WordPress content in /var.

The specific directories to be backed up will differ between organizations and even hosts within an organization. It will be necessary to determine what needs to be backed up for each host.

I never back up the operating system itself, such as /boot, /usr, /bin, /sbin, and /lib, if they exist. Some of the bin and lib files are being reconfigured into a single directory in any event. A reinstallation can easily and quickly recover the operating system. And snapshots make recovery of VMs particularly easy – so long as you are making them.

Some organizations use what is called bare-metal restores which means take an empty hard drive and restore everything as a complete disk image. This takes "full" to the ultimate level. Such a bare-metal restoration is beyond the scope of this course, but I bet you are already thinking about how to do it with **dd**.

## All vs. diff

When using the **tar** utility as we did here in this chapter, we simply made a complete backup of everything specified. The **tar** utility can also add a diff – a complete replacement of an altered file – to the end of a tarball, but the diff consists the entirety of the files that are altered and not just the individual portions that have changed.

Advanced tools like **rsync** can be used to alter the changed portions of a file in the target backup. However, the structure of an rsync backup is significantly different as we will see in Chapter 16 of Volume 3.

The answer in this case is another resounding, "it depends." I have always found that it is easier to create and recover using complete backups when utilizing **tar**. With **rsync** a diff is better because it is faster and makes no difference in the final backup.

## Considerations for automation of backups

Backups are one of those tasks best automated to ensure consistency in the timing of their runs. It also prevents someone forgetting to start it off.

Consideration needs to be given to things such as which automation tools to use, for example, cron vs. systemd timers, or something else. Also, do we want to consider commercial backup systems, advanced open source backup tools, and locally created tools such as scripts?

Timing of the backups also needs some consideration. For example, do we start the backups in the evening or the early morning? If they are too close together, will they overlap and create problems?

I had one instance where a backup took so long that the next iteration started before the first was finished. This caused those two instances to lock each other out as well as the next several ones. Once discovered, it was simple to terminate all instances of the backup programs, reset the timing of backups, and start over. A situation like this highlights the need for warning messages in case of problems as well as the need to check the backups on a regular basis.

## Dealing with offline hosts

This can be a problem and I have encountered it myself. Most backup systems, including locally written scripts, will simply time out and proceed to the next remote host. This is usually due to the attempted SSH connection timing out.

A simple system will ignore the missing backup on the next iteration and make a new one. It could also take the more sophisticated approach in which the backup software determines the latest successful backup, creates a new instance, and proceeds with the current backup while transferring only altered portions of files. The latter is the approach taken by the script I wrote for my own backups.

## Advanced backups

In Chapter 16 of Volume 3 of this course, we will explore an advanced backup technique using a common Linux tool, **`rsync`**. This tool allows us to create a script for an automated backup system that can perform repeated and frequent complete backups while only copying the altered portions of files whether on the localhost or across the network. It handles dealing with missing backups. Restores are easy for the SysAdmin and users alike because they are in the form of copies of the original files.

The **`rsync`** utility is part of the Linux core utilities so is already installed on almost all Linux hosts. It has some interesting and powerful features that enable those capabilities in our script. We will explore those features in some detail in Volume 3.

# Chapter summary

Backups are an incredibly important part of our jobs as SysAdmins. I have experienced many instances where backups have enabled rapid operational recovery for places I have worked as well as for my own business and personal data.

There are many options for performing and maintaining data backups. I do what works for me and have never had a situation where I lost more than a few hours worth of data.

Like everything else, backups are all about what you need. Whatever you do – do something! Figure out how much pain you would have if you lost everything – data, computers, hard copy records – everything. The pain includes the cost of replacing the hardware and the cost of the time required to restore data that was backed up and to recover data that was not backed up. Then plan and implement your backup systems and procedures accordingly.

We will explore backups in more depth in Volume 3 of this course.

# Exercises

Perform the following exercises to complete this chapter:

1. If you are running your VM on a Linux host, you will of necessity have access to root. As root, run **`smartctl -x /dev/sda`** on that physical host. Explore the output for signs of disk failure.
2. If you are running your VM on a Windows host, create a live USB thumb drive from the downloaded ISO image used to install Fedora on your VM. Then boot that host using the live USB device. As root, run **`smartctl -x /dev/sda`** on that physical host. Explore the output for signs of disk failure.
3. List three advantages of using the **`tar`** command for backups.
4. Determine how much space you need to back up the /home, /root, and /etc directories. Locate a USB drive large enough to contain that much data and make it available to your VM. Make a backup of those three directories on the USB drive.
5. What advantages might there be to use “the cloud” for backups?

6. Write a simple script to automate backups of /home, /root, and /etc, and configure a systemd timer to run the script every day at 02:00 AM. Create a new filesystem from the existing space on your volume group, fedora_studentvm1, for storage of the backup; use this as a substitute for an external device. Test it to verify that it works.
7. Wipe out the /home/student home directory by deleting all of the files in it and deleting the directory as well. Restore /home/student from the backup you made in the previous exercise.
8. Can the Thunar GUI file manager be used to access the archive tarballs and extract files from them?

# Bibliography

## Books

Binnie, Chris, *Practical Linux Topics*, Apress 2016, ISBN 978-1-4842-1772-6

Both, David, The Linux Philosophy for SysAdmins, Apress, 2018, ISBN 978-1-4842-3729-8

Gancarz, Mike, *Linux and the Unix Philosophy*, Digital Press – an imprint of Elsevier Science, 2003, ISBN 1-55558-273-7

Kernighan, Brian W.; Pike, Rob (1984), *The UNIX Programming Environment*, Prentice Hall, Inc., ISBN 0-13-937699-2

Libes, Don, *Exploring Expect*, O'Reilly, 2010, ISBN 978-1565920903

Nemeth, Evi [et al.], *The Unix and Linux System Administration Handbook*, Pearson Education, Inc., ISBN 978-0-13-148005-6

Matotek, Dennis, Turnbull, James, Lieverdink, Peter; *Pro Linux System Administration*, Apress, ISBN 978-1-4842-2008-5

Raymond, Eric S., *The Art of Unix Programming*, Addison-Wesley, September 17, 2003, ISBN 0-13-142901-9

Siever, Figgins, Love & Robbins, *Linux in a Nutshell 6th Edition* (O'Reilly, 2009), ISBN 978-0-596-15448-6

Sobell, Mark G., *A Practical Guide to Linux Commands, Editors, and Shell Programming Third Edition,* Prentice Hall; ISBN 978-0-13-308504-4

van Vugt, Sander, *Beginning the Linux Command Line,* Apress, ISBN 978-1-4302-6829-1

Whitehurst, Jim, *The Open Organization*, Harvard Business Review Press (June 2, 2015), ISBN 978-1625275271

Torvalds, Linus and Diamond, David, Just for Fun, HarperCollins, 2001, ISBN 0-06-662072-4




D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4

# Web sites

BackBlaze, Web site, *What SMART Stats Tell Us About Hard Drives*, www.backblaze.com/blog/what-smart-stats-indicate-hard-drive-failures/

Both, David, *8 reasons to use LXDE*, https://opensource.com/article/17/3/8-reasons-use-lxde

Both, David, *9 reasons to use KDE*, https://opensource.com/life/15/4/9-reasons-to-use-kde

Both, David, *10 reasons to use Cinnamon as your Linux desktop environment*, https://opensource.com/article/17/1/cinnamon-desktop-environment

Both, David, *11 reasons to use the GNOME 3 desktop environment for Linux*, https://opensource.com/article/17/5/reasons-gnome

Both, David, *An introduction to Linux network routing*, https://opensource.com/business/16/8/introduction-linux-network-routing

Both, David, *Complete Kickstart*, www.linux-databook.info/?page_id=9

Both, David, *Making your Linux Box Into a Router*, www.linux-databook.info/?page_id=697

Both, David, *Network Interface Card (NIC) name assignments*, www.linux-databook.info/?page_id=4243

Both, David, *Using hard and soft links in the Linux filesystem*, www.linux-databook.info/?page_id=5087

Both, David, *Using rsync to back up your Linux system*, https://opensource.com/article/17/1/rsync-backup-linux

Bowen, Rich, *RTFM? How to write a manual worth reading*, https://opensource.com/business/15/5/write-better-docs

Charity, *Ops: It's everyone's job now*, https://opensource.com/article/17/7/state-systems-administration

Dartmouth University, *Biography of Douglas McIlroy*, www.cs.dartmouth.edu/~doug/biography

DataBook for Linux, www.linux-databook.info/

Digital Ocean, *How To Use journalctl to View and Manipulate Systemd Logs*, www.digitalocean.com/community/tutorials/how-to-use-journalctl-to-view-and-manipulate-systemd-logs

Edwards, Darvin, Electronic Design, *PCB Design And Its Impact On Device Reliability*, www.electronicdesign.com/boards/pcb-design-and-its-impact-device-reliability

Engineering and Technology Wiki, *IBM 1800*, http://ethw.org/IBM_1800

Fedora Magazine, *Tilix*, https://fedoramagazine.org/try-tilix-new-terminal-emulator-fedora/

Fogel, Kark, *Producing Open Source Software*, https://producingoss.com/en/index.html

Free On-line Dictionary of Computing, *Instruction Set*, http://foldoc.org/instruction+set

Free Software Foundation, *Free Software Licensing Resources*, www.fsf.org/licensing/education

gnu.org, *Bash Reference Manual – Command Line Editing*, www.gnu.org/software/bash/manual/html_node/Command-Line-Editing.html

Harris, William, *How the Scientific Method Works*, https://science.howstuffworks.com/innovation/scientific-experiments/scientific-method6.htm

Heartbleed web site, http://heartbleed.com/

How-two Forge, *Linux Basics: How To Create and Install SSH Keys on the Shell*, www.howtoforge.com/linux-basics-how-to-install-ssh-keys-on-the-shell

Kroah-Hartman, Greg, Linux Journal, *Kernel Korner – udev – Persistent Naming in User Space*, www.linuxjournal.com/article/7316

Krumins, Peter, *Bash emacs editing*, www.catonmat.net/blog/bash-emacs-editing-mode-cheat-sheet/

Krumins, Peter, *Bash history*, www.catonmat.net/blog/the-definitive-guide-to-bash-command-line-history/

Krumins, Peter, *Bash vi editing*, www.catonmat.net/blog/bash-vi-editing-mode-cheat-sheet/

Kernel.org, *Linux allocated devices (4.x+ version)*, www.kernel.org/doc/html/v4.11/admin-guide/devices.html

Linux Foundation, *Filesystem Hierarchical Standard (3.0)*, http://refspecs.linuxfoundation.org/fhs.shtml

Linux Foundation, *MIT License*, https://spdx.org/licenses/MIT

The Linux Information Project, *GCC Definition*, www.linfo.org/gcc.html

Linuxtopia, *Basics of the Unix Philosophy*, www.linuxtopia.org/online_books/programming_books/art_of_unix_programming/ch01s06.html

LSB Work group - The Linux Foundation, *Filesystem Hierarchical Standard V3.0, 3*, https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf

Opensource.com, https://opensource.com/

Opensource.com, *Appreciating the full power of open*, https://opensource.com/open-organization/16/5/appreciating-full-power-open

Opensource.com, David Both, *SpamAssassin, MIMEDefang, and Procmail: Best Trio of 2017*, Opensource.com, https://opensource.com/article/17/11/spamassassin-mimedefang-and-procmail

Opensource.org, *Licenses*, https://opensource.org/licenses

opensource.org, *The Open Source Definition (Annotated)*, https://opensource.org/osd-annotated

OSnews, *Editorial: Thoughts on Systemd and the Freedom to Choose*, www.osnews.com/story/28026/Editorial_Thoughts_on_Systemd_and_the_Freedom_to_Choose

Peterson, Christine, Opensource.com, *How I coined the term 'open source,'* https://opensource.com/article/18/2/coining-term-open-source-software

Petyerson, Scott K, *The source code is the license*, Opensource.com, https://opensource.com/article/17/12/source-code-license

Princeton University, *Interview with Douglas McIlroy*, www.princeton.edu/~hos/frs122/precis/mcilroy.htm

Raspberry Pi Foundation, www.raspberrypi.org/

Raymond, Eric S., *The Art of Unix Programming*, www.catb.org/esr/writings/taoup/html/index.html/

Wikipedia, *The Unix Philosophy, Section: Eric Raymond's 17 Unix Rules*, https://en.wikipedia.org/wiki/Unix_philosophy#Eric_Raymond%E2%80%99s_17_Unix_Rules

Raymond, Eric S., *The Art of Unix Programming, Section The Rule of Separation*, www.catb.org/~esr/writings/taoup/html/ch01s06.html#id2877777

*Understanding SMART Reports*, https://lime-technology.com/wiki/Understanding_SMART_Reports

Unnikrishnan A, Linux.com, *Udev: Introduction to Device Management In Modern Linux System*, www.linux.com/news/udev-introduction-device-management-modern-linux-system

Venezia, Paul, *Nine traits of the veteran Unix admin*, InfoWorld, Feb 14, 2011, www.infoworld.com/t/unix/nine-traits-the-veteran-unix-admin-276?page=0,0&source=fssr

Wikipedia, *Alan Perlis*, https://en.wikipedia.org/wiki/Alan_Perlis

Wikipedia, *Christine Peterson*, https://en.wikipedia.org/wiki/Christine_Peterson

Wikipedia, *Command Line Completion*, https://en.wikipedia.org/wiki/Command-line_completion

Wikipedia, *Comparison of command shells*, https://en.wikipedia.org/wiki/Comparison_of_command_shells

Wikipedia, *Dennis Ritchie*, https://en.wikipedia.org/wiki/Dennis_Ritchie

Wikipedia, *Device File*, https://en.wikipedia.org/wiki/Device_file

Wikipedia, *Gnome-terminal*, https://en.wikipedia.org/wiki/Gnome-terminal

Wikipedia, *Hard Links*, https://en.wikipedia.org/wiki/Hard_link

Wikipedia, *Heartbleed*, https://en.wikipedia.org/wiki/Heartbleed

Wikipedia, *Initial ramdisk*, https://en.wikipedia.org/wiki/Initial_ramdisk

Wikipedia, *Ken Thompson*, https://en.wikipedia.org/wiki/Ken_Thompson

Wikipedia, *Konsole*, https://en.wikipedia.org/wiki/Konsole

Wikipedia, *Linux console*, https://en.wikipedia.org/wiki/Linux_console

Wikipedia, *List of Linux-supported computer architectures*, https://en.wikipedia.org/wiki/List_of_Linux-supported_computer_architectures

Wikipedia, *Maslow's hierarchy of needs*, https://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs

Wikipedia, *Open Data*, https://en.wikipedia.org/wiki/Open_data

Wikipedia, *PHP*, https://en.wikipedia.org/wiki/PHP

Wikipedia, *PL/I*, https://en.wikipedia.org/wiki/PL/I

Wikipedia, *Programma 101*, https://en.wikipedia.org/wiki/Programma_101

Wikipedia, *Richard M. Stallman*, https://en.wikipedia.org/wiki/Richard_Stallman

Wikipedia, *Rob Pike*, https://en.wikipedia.org/wiki/Rob_Pike

Wikipedia, *rsync*, https://en.wikipedia.org/wiki/Rsync

Wikipedia, *Rxvt*, https://en.wikipedia.org/wiki/Rxvt

Wikipedia, *SMART*, https://en.wikipedia.org/wiki/SMART

Wikipedia, *Software testing*, https://en.wikipedia.org/wiki/Software_testing

Wikipedia, *Terminator*, https://en.wikipedia.org/wiki/Terminator_(terminal_emulator)

Wikipedia, *Tony Hoare*, https://en.wikipedia.org/wiki/Tony_Hoare

Wikipedia, *Unit Record Equipment*, https://en.wikipedia.org/wiki/Unit_record_equipment

Wikipedia, *Unix*, https://en.wikipedia.org/wiki/Unix

Wikipedia, *Windows Registry*, https://en.wikipedia.org/wiki/Windows_Registry

Wikipedia, *Xterm*, https://en.wikipedia.org/wiki/Xterm

WikiQuote, *C._A._R._Hoare*, https://en.wikiquote.org/wiki/C._A._R._Hoare

WordPress, *Home page*, https://wordpress.org/

# Index

## A



## B






D. Both, *Using and Administering Linux: Volume 2*, https://doi.org/10.1007/978-1-4842-5455-4

## C

## D

## E



## F

## G



## H



## I, J

## K



## L

## M



## N



## O



## P, Q

## R

## S

## T



## U

## V



## W



## X, Y, Z



## Command list

## List of operators