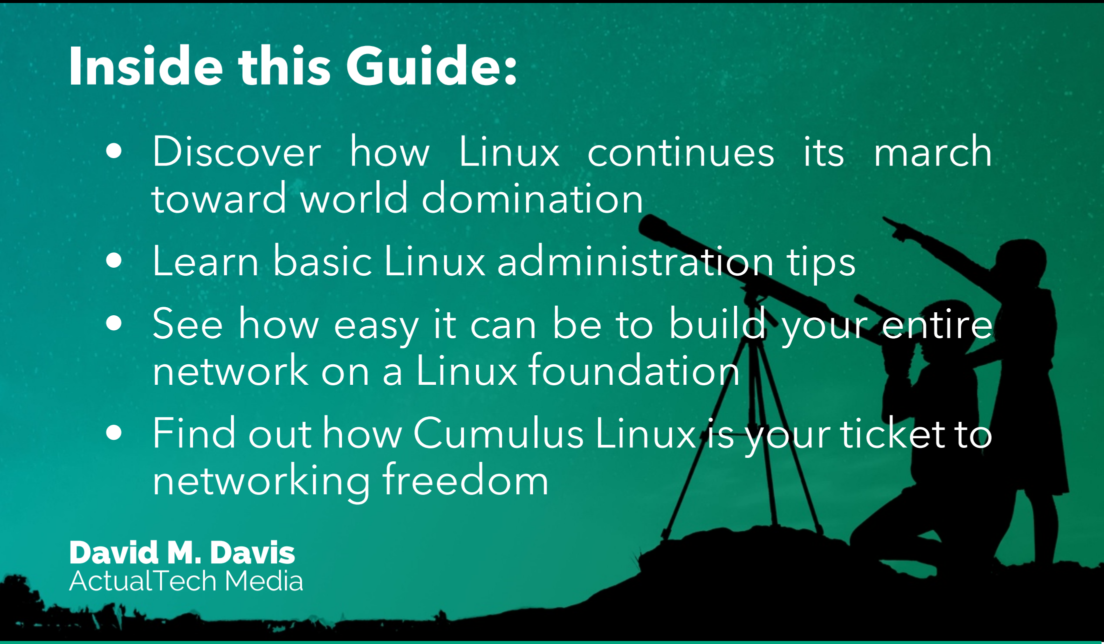
THE GORILLA GUIDE TO...®

Linux Networking 101



Inside this Guide:

- Discover how Linux continues its march toward world domination
- Learn basic Linux administration tips
- See how easy it can be to build your entire network on a Linux foundation
- Find out how Cumulus Linux is your ticket to networking freedom

David M. Davis
ActualTech Media

HELPING YOU NAVIGATE
THE TECHNOLOGY JUNGLE!

ActualTech Media
www.actualtechmedia.com

In Partnership With
CUMULUS

The Gorilla Guide To...

Linux Networking 101

Author
David M. Davis, ActualTech Media

Editors
Hilary Kirchner, Dream Write Creative, LLC
Christina Guthrie, Guthrie Writing & Editorial, LLC
Madison Emery, Cumulus Networks

Layout and Design
Scott D. Lowe, ActualTech Media

The information provided within this eBook is for general informational purposes only. While we try to keep the information up-to-date and correct, there are no representations or warranties, express or implied, about the completeness, accuracy, reliability, suitability or availability with respect to the information, products, services, or related graphics contained in this book for any purpose. Any use of this information is at your own risk.

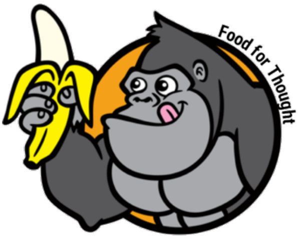
ActualTech Media
Okatie Village Ste 103-157
Bluffton, SC 29909
www.actualtechmedia.com

Entering the Jungle

Callouts Used in This Book

School House

The Gorilla is the professorial sort that enjoys helping people learn. In the Schoolhouse callout, you'll gain insight into topics that may be outside the main subject but that are still important.



This is a special place where readers can learn a bit more about ancillary topics presented in the book.

Bright Idea

When we have a great thought, we express them through a series of grunts in the Bright Idea section.

Takes readers into the deep, dark depths of a particular topic.

Icons Used in This Book

Definition. Defines a word, phrase, or concept.

Knowledge Check. Tests your knowledge of what you've read.

Pay attention. We want to make sure you see this!

GPS. We'll help you navigate your knowledge to the right place.

Watch out! Make sure you read this so you don't make a critical error!

Introduction

Modern data centers are vastly different from legacy ones, and with good reason. In the past, companies typically supported a handful of critical monolithic applications, and the network was put in place primarily to support just those applications. Once installed, the network was left mostly untouched in many organizations. It consisted of dedicated hardware-based routers and switches that, for the times, performed their tasks of routing and switching packets quite well. The routers and switches favored by many enterprises typically came from one of the "big 3" networking vendors, but their products generally included costly appliances made up of custom hardware and highly proprietary software. That network gear was so specialized that an entire ecosystem sprang up around it to provide training, education, certification, consulting, software and support maintenance, and more.

Over time, the data center landscape has changed — and for the better, particularly given that the application landscape has also morphed into something radically different from what was seen in the past. The number of business-critical applications is on the rise, and, unlike their older stay-at-home cousins, modern applications are distributed between on-premises infrastructure, between partner networks, and across the public cloud. End user and company data moves around the globe at light speed, and it's happening constantly. New applications are being built today and torn down tomorrow in favor of even newer applications. Change is happening fast, and the network is adapting to support these changes.

Thankfully, the specialized hardware that characterized legacy data centers isn't so necessary anymore. Today, networking needs are being met using industry-standard switching/routing silicon, off-the-shelf hardware, Intel CPUs, and the Linux operating system. This combination makes networking far more affordable, more scalable, easier to learn, and more adaptable to the constantly changing needs of the business. After all, the network's sole purpose is to connect the users with their applications and data, so it should do it as reliably, securely, efficiently, and affordably as possible.

The key piece of the previous paragraph and the focus of this book is this: **Linux networking is the future for almost every use case.** But to leverage a Linux-based networking solution, you need to understand Linux, and that's where this book comes in.

Definitions Abound!

If you don't know what some of these words mean, don't worry! We'll define them during your Linux 101 journey. By the end of this book, you'll be using these phrases in casual conversation!

Six Reasons You Need to Learn Linux

What if you don't know Linux and are asking yourself, "Is this book really worth my time?" The short answer is a resounding YES, but to back that up, let me give you six good reasons why you should invest some of your time to learn Linux.

1. Linux is the future

Although Linux has been around for over 25 years, it has enjoyed a continuous rise in business-critical usage, and many see Linux as being the most popular operating system for the future. The reason as to why Linux is the lingua franca of the modern data center relates to the points below.

2. Linux is on everything

Linux runs more than *two-thirds* of the servers on the Internet, all Android phones, most consumer network gear, such as NetGear and Linksys devices, 99% of the top supercomputers in the world, many Internet of Things (IoT) devices, Tesla cars, and even PlayStation gaming consoles.

3. Linux is adaptable

The very reason everything is on Linux is because it's such an adaptable operating system. Thanks to Linux's modularity and open-source nature, you can choose the pieces you need for your product or service and develop any pieces that may not already exist. You can install tiny versions of Linux for specialized use cases (such as operating water sprinklers in the gorilla exhibit at the zoo), modify it to work on appliances that route packets across a large enterprise network, or use it as your desktop operating system. Your choices are practically endless.

4. Linux has a strong community and ecosystem

Linux has been so successful mainly because of the strong community and ecosystem that surrounds it. There are Linux contributors (developers who write code to make the product better); Linux forums and communities; Linux instructors; Linux training options; Linux blogs; Linux third-party tools; Linux distributions; Linux conferences; and even Linux books such as this one!

5. Linux is fun!

Linux is a lot of fun because you can do just about anything with it. Linux is commonly used in Internet of Things (IoT) projects; it runs on tiny Raspberry Pi computers commonly used by hobbyists, and it even makes a great operating system on your laptop or desktop computer. More examples of the many uses of Linux are found throughout the book.

6. Linux is open-source and sometimes free

Linux is open-source, meaning that the original source code is made freely available and may be redistributed and modified. That said, there are paid and fully supported commercial editions available, too. The open nature of Linux has made it the adaptable OS of the future, allowing it to run on everything, and has resulted in the creation of a strong ecosystem.

Ready to start learning Linux?

Head to the first chapter in this Gorilla Guide and find out the answer to the burning question: **What is Linux?**

Chapter 1

What Is Linux?

As you get started learning about Linux, you'll likely have many of the same questions that thousands of other people have had since the beginning of Linux time. For that reason, we'll start this chapter by answering the most common questions about Linux.

By reading this chapter, you'll find the answers to these questions:

1. What is an operating system?
2. What makes up the Linux OS?
3. What makes Linux unique?
4. What are the b enefits of using Linux?

"Hello everybody out there using minix I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu)…"

Photo by Krd (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0) or CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons

Figure 1-1. Linus Torvalds, principal author of the Linux kernel, on August 25, 1991, when he announced his new Linux kernel.

The History of Linux

Before we dive into Linux, let's first take a step back in history. The creation of Linux starts with another operating system known as UNIX, which was first released in 1971. In 1983, the GNU Project (which stood for "GNU's not Unix") was started to create a complete UNIX-compatible operating system. Efforts stalled, and the project was missing a *kernel*. Around 1987, a UNIX-like operating system for students was released called MINIX, but its licensing prevented it from being distributed freely. Linus Torvalds (Figure 1-1) at the University of Helsinki in Finland was frustrated by the licensing of MINIX and began working on his own operating system kernel. His kernel, released in 1991, when combined with the GNU applications and open-source licensing, became the Linux operating system we know today.

What Is a Kernel, and What Does It Do?

The *kernel* is the special piece of the operating system that controls the CPU hardware, allocates memory, accesses data, schedules processes, runs the applications, and protects them from each other. It is the first program loaded on the computer when the computer starts up. The most critical pieces of code in the kernel are loaded into protected areas of memory so that they can't be overwritten by other applications running in the operating system.

Since then, thousands of developers from around the world have contributed to enhancing the Linux kernel as well as the many pieces of software that make up the many different Linux distributions. Those developers include volunteers as well as developers from commercial companies. Today, the nonprofit Linux Foundation helps to create standards, awareness, and advancements across many different Linux projects.

What Is an Operating System?

The short answer is that an *operating system*, or OS, is software that you load on your hardware to make it "do things." Without an operating system, most hardware is useless. For example, you might have a Dell computer that runs the Windows 10 operating system from which you run your applications. You might have an iPhone that runs the iOS operating system. You may also have an Apple MacBook that runs the Apple macOS operating system. The operating systems on these hardware platforms are what enable them to run applications, as shown in Figure 1-2.

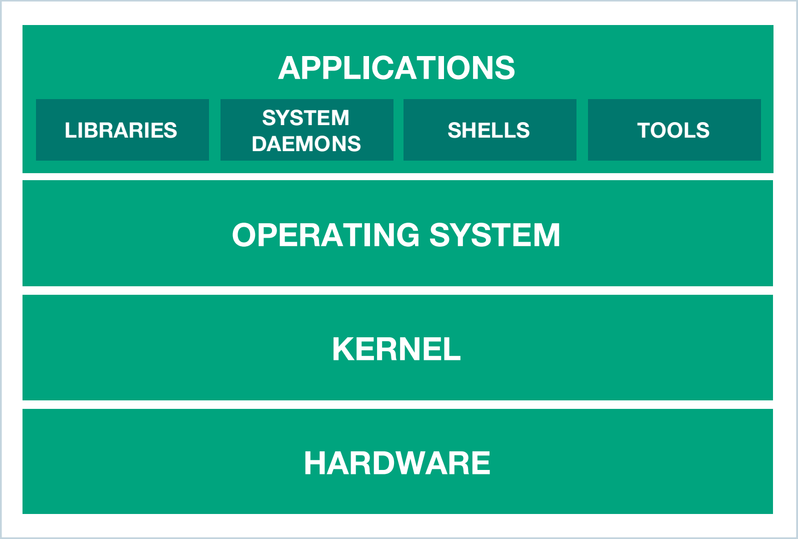


FIGURE 1-2. HOW AN OPERATING SYSTEM WORKS WITH HARDWARE AND APPLICATIONS

The Components that Comprise the Linux Operating System

Linux is an open-source OS that can be installed on a variety of different types of hardware to allow you to develop software, run applications, and more. At the heart of Linux is the *kernel*. Linux was developed in C and assembly language to run on i386 personal computers, but it has since been ported to more hardware than just about any other operating system in history. Today, Linux is the most installed operating system globally. In fact, the Space X Falcon 9 rocket and the International Space Station both use Linux!

Linux is typically administered from a command line interface (CLI), also known as a *shell*. Besides the kernel, which manages the hardware and software processes, Linux distributions include a collection of Linux software, such as device drivers for accessing and controlling hardware, shared libraries, applications, and system *daemons*, which run the in background and respond to network requests. Figure 1-3 shows an example of what a common Linux distribution might look like. Numerous programming languages are available for Linux, as well as more than 70,000 different applications. Applications are installed from *packages*, which contain the application itself and metadata about the application.

Definition: Metadata

Metadata is data about data. In essence, metadata describes the kind of information that an underlying data set will store. Take, for instance, a file system on a computer. When you view a directory listing, you see the file name, file size, create date, last modified date, and so forth. These are basic examples of metadata associated with each object in that directory.

What is a Linux Daemon?

A system daemon in Linux is typically a background system process that awaits a specific set of conditions before jumping into action. For example, your Linux system may have a daemon called *sshd*, which stands for Secure Shell daemon. This system daemon runs in the background and accepts authorized incoming requests to log into the Linux host. System daemons do not interact with users and are not typically under the direct control of users, but rather of the system itself.

What Is a Distribution?

Often called a "distro," a Linux *distribution* is the combination of specific versions of the Linux kernel with other libraries, system daemons, development tools, applications, packaging, and life-cycle management tools that are compatible with each other and tested for interoperability. The most common way that people acquire Linux today is by downloading one of the many different Linux distributions. Distributions are available not just for servers, desktop, and laptop computers, but also for a huge variety of more specialized devices that run Linux. Examples of Linux distributions are Ubuntu, Debian, Fedora, openSUSE, and Cumulus Linux.

Understanding User Space vs. Kernel Space

Operating systems all execute their kernel in protected and restricted memory that is called *kernel space* (see Figure 1-4) to prevent the kernel from terminating and crashing the system.

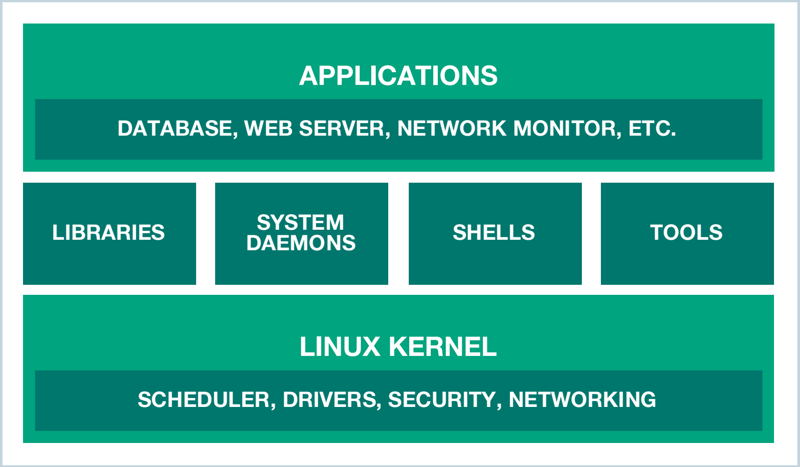

APPLICATIONS
DATABASE, WEB SERVER, NETWORK MONITOR, ETC.
LIBRARIES
SYSTEM DAEMONS
SHELLS
TOOLS
LINUX KERNEL
SCHEDULER, DRIVERS, SECURITY, NETWORKING

Figure 1-3. Example of a common Linux distribution

When a user runs an application or tool, that application or tool executes in what is called *user space*. This distinction is critical. Applications can come from a variety of sources, may be poorly developed, or originate unknown sources. By running these applications separate from kernel space, they can't tamper with the kernel resources and cause the system to *panic* (crash).

All applications, even system daemon processes that perform critical operating system functions, must make what is called a "system call" to the kernel in the kernel space in order to access system resources such as memory or network devices. Every modern multi-user operating system has some type of user space versus kernel space design, which is intended to keep it secure, high-performing, and reliable.

In short, the separation between user space and kernel space is made to ensure that Linux is as reliable and secure an operating system as possible.

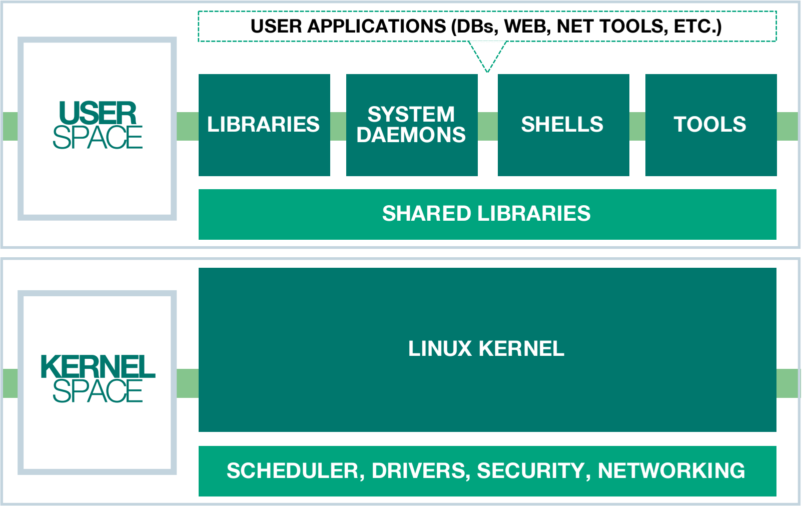

USER APPLICATIONS (DBs, WEB, NET TOOLS, ETC.)
USER SPACE
LIBRARIES
SYSTEM DAEMONS
SHELLS
TOOLS
SHARED LIBRARIES
KERNEL SPACE
LINUX KERNEL
SCHEDULER, DRIVERS, SECURITY, NETWORKING

Figure 1-4. User Space and Kernel Space in the Linux kernel

Benefits of Using Linux

Besides the fact that Linux is a great operating system, is continually being enhanced, and has a huge community following, Linux has gained such tremendous popularity because there are so many different benefits to using it. Some of these benefits include:

- **Consistent operating model**. No matter what version or distribution of Linux you use, whether you're on a supercomputer or a tiny embedded device, the general operation of Linux is the same no matter where you go. What this means is that, with some exceptions, the command line syntax is similar, process management is similar, basic network administration is similar, and applications can be (relatively) easily ported between distributions. The end result of this consistent operating model is a cost savings generated by greater staff efficiency and flexibility.

- **Scalability**. At this point, you already know that Linux is eminently scalable and is able to run on everything from wristwatches to supercomputers to globally distributed computing clusters. Of course, the benefit of this scalability isn't just the device mix, but also that its basic functionality — command line tools, configuration, automation, and code-compatibility — remains the same no matter where you're using it.

- **Open-source and community optimized**. With Linux's open-source, freely available nature, you might be concerned about future enhancements, bug fixes, and support. Fortunately, you can put those worries aside. If you look at the Linux kernel alone, with its 22 million lines of code, you'll find a strong community developing it behind the scenes. In 2016, one report said that over 5,000 individual developers representing 500 different corporations around the world contributed to enhancements in the Linux kernel, not to mention all the other surrounding applications and services. A staggering 13,500 developers from more than 1,300 companies have contributed to the Linux kernel since 2005. You might wonder why commercial entities contribute code to Linux. While many open-source advocates see the open-source nature of Linux as purely idealistic, commercial contribution of code is actually a strategic activity. In this sense, the for-profit companies who are dependent on Linux contribute their changes to the core to ensure that those changes carry forward into future distributions without having to maintain them indefinitely.

- **Full function networking**. Over the years, Linux has built up a strong set of networking capabilities, including networking tools for providing and managing routing, bridging, DNS, DHCP, network troubleshooting, virtual networking, and network monitoring.
- **Package management**. The Linux package management system allows you to easily install new services and applications with just a few simple commands.

Linux Package Management

A Linux package management system is a tool that helps Linux administrators install and manage applications and extensions for the Linux operating system. Each Linux distribution carries its own package management capabilities. A Linux package includes all the bits necessary for a new application or service to operate. The package management system can also help an administrator address any dependencies that a package may have. A dependency is a software package necessary for another package to operate. By layering these dependencies, newly developed packages can then leverage the work of others without having to constantly reinvent the wheel. However, maintaining dependencies can be difficult, particularly as you continue to add packages. A good package management system will ensure that all dependencies are handled at the same time that you install new packages.

You will learn more about package management later in this book.

How Is Linux Used in the Enterprise?

Many modern ideas in data center computing have Linux underpinnings. Here are just a few examples:

- **Automation and orchestration.** Automation is used to perform a common task in a programmatic/scripted way, whereas orchestration is used to automate tasks across multiple systems in a data center. Linux is being used to automate and orchestrate just about every process in the enterprise data center.

- **Server virtualization.** Server virtualization is the ability to run a full operating system on top of an existing bare metal server. These virtual machines (VMs) can be used to increase server utilization, simplify server testing, or lower the cost of server redundancy. The software that allows VMs to function is called a *hypervisor.* Linux includes an excellent hypervisor called KVM.

- **Private cloud.** Another open-source project called OpenStack, which also runs on Linux, has become a leading cloud management platform for creating a private cloud. With private cloud, companies can leverage many of the same advantages of public cloud (scalability, self-service, multi-tenancy, and more) while running their own IT infrastructure on-premises.

- **Big data.** More and more companies are having to deal with exponentially increasing amounts of data in their data center, and because Linux offers such scalability and performance, it has become the go-to operating system for crunching big data via applications like Hadoop. Even Microsoft recently announced a big data solution based on Linux.

- **Containers.** Linux can also be used to run containerized applications, such as Docker containers, which are being used more and more by many companies. In fact, Linux is the foundation of the modern container movement; all container packaging and orchestration relies on Linux namespace and isolation mechanisms in order to operate.

Knowledge Check

Answer the following questions to check your knowledge concerning the basics of Linux:

- What is the Linux kernel?
- What is an operating system?
- What's the difference between user space and kernel space?

Summary

In this chapter, you learned what Linux is, where it came from, how it's being used, and how powerful it is. With that knowledge, it's time to get started using Linux yourself! In the next chapter, you will learn where to download a Linux distribution, discover the basics of Linux administration, including how the Linux file system works, how to manage processes, how to log into Linux, and how to deploy new packages.

Read on to continue your quest to learn Linux!

Chapter 2

Basics of Linux Administration

Even though this chapter is titled "Basics of Linux Administration," you should know that this chapter is meant for anyone getting started with Linux, whether or not you plan to be a Linux administrator in the future. You can consider this chapter a "getting started with Linux" resource.

Here's what you'll learn:

- Where you can get Linux
- The basics of the Linux file system
- How Linux processes work

Let's start with the most basic of Linux questions.

Where Do I Get Linux?

To get started with Linux, you need to download a Linux distribution, such as RedHat Enterprise Linux, Ubuntu, Debian, Fedora, openSUSE, CentOS, or Cumulus Linux. You want to make sure that you obtain a Linux distribution that is compatible with your hardware. For example, you might select a 32-bit i386 image or a 64-bit amd64 image.

For example, if you want to start with the Debian distribution, you can download an ISO-formatted image that you would use to install Debian Linux from https://www.debian.org/distrib/

While some people will want to run Linux directly on a physical server, desktop, or laptop, many people start learning Linux for the first time by running it inside of a virtual machine. With a VM option, you can run Linux inside your existing Microsoft Windows or Apple macOS operating system using virtualization tools such as VMware Workstation or VMware Fusion, both of which both offer a free, limited-time evaluation license. You can also go with a free product from Oracle called VirtualBox. Another option is to run Linux as a VM in the public cloud via a provider such as Amazon Web Services or Microsoft Azure.

In this book, I'll skip the steps on how to install Linux because you won't have to perform the typical installation if you use a live image. Instead, we will focus on building a skill set on the administrative tasks needed to understand and navigate Linux.

CLI vs. GUI

Linux can be administered through a graphical user interface (GUI) or command line interface (CLI). Most direct uses of Linux by consumers/individuals are done with a GUI, as with Android phone users or Linux desktop users. Most Linux servers are administered through the CLI, as administrators typically find it to be more efficient. Throughout this book, I'll be using the CLI for examples.

How Do I Log In to Linux?

Because most Linux administration is done using a CLI, you log in to Linux at either the console of the Linux host machine or by remotely connecting to the Linux server over a network. For a new installation, you typically log in to the console to install system packages and then set up initial users with passwords and network access.

Most Linux servers are set up to allow users to connect via the network using the *Secure Shell* (SSH), an encrypted communications protocol.

SSH is a secure alternative to the insecure *telnet* that was used in the past. With SSH, your SSH client connects to the SSH server running on the Linux host where you log in with a username and password. We'll talk more about user administration later in this chapter.

Get an SSH client!

SSH clients are all over the place! If you're a Windows user, one of the most popular clients is PuTTY, which is available for download from www.putty.org. If you're a Mac user, you can just use the Terminal application that is already built into macOS. If you're on a mobile device, head to your device's application store and look around. You'll find a multitude of options.

Here's how establishing a connection might work from a system that already has an SSH client (such as macOS in this case) connecting to a Linux host (Cumulus Linux in this case) over the network:

```
macos1:~ david$ ssh cumulus@192.168.1.107
cumulus@192.168.1.107's password: ********
Welcome to Cumulus VX (TM)
Cumulus VX (TM) is a community supported virtual
appliance designed for experiencing, testing, and
prototyping Cumulus Networks' latest technology.
For any questions or technical support, visit our
community site at:  http://community.cumulusnetworks.com
The registered trademark Linux ® is used pursuant to a
sublicense from LMI, the exclusive licensee of
```

As you can see, with SSH, you connect using the command **ssh,** followed by the **Username,** an **@** symbol, and then the **Hostname** or **IP Address** of the Linux host to which you are trying to connect. You will be prompted for your password to log in. In the example above, the password is required, but is not echoed and therefore not shown.

Where do usernames and passwords come from?

You may be wondering where these usernames and passwords come from. The "superuser" username in Linux is called "root" because it is the only user that can modify the root directory. During installation, the root user is created, and you are able to select a password. Post-installation, administrators can use the root user account and account management commands to create new users (with associated passwords) for normal user activities.

How Do I Know What Type of Linux I Am Using?

Because there are so many different types of Linux, you want to be sure you know what distribution and version you are using (for the sake of searching the right documentation on the Internet, if nothing else). Keep in mind a couple different commands to identify your Linux version.

The **uname** command shows the basic type of operating system you are using, like this:

```
david@debian:~$ uname -a
Linux debian 3.16.0-4-686-pae #1 SMP Debian 3.16.43-2
(2017-04-30) i686 GNU/Linux
```

And the **hostnamectl** command shows you the hostname of the Linux server as well as other system information, like the machine ID, virtualization hypervisor (if used), operating system, and Linux kernel version. Here's an example:

```
david@debian:~$ hostnamectl
Static hostname: debian
Icon name: computer-vm
```

```
    Chassis: vm
    Machine ID: 0eb625ef6e084c9181b9db9c6381d8ff
    Boot ID: 8a3ef6ecdfcf4218a6102b613e41f9ee
    Virtualization: vmware
    Operating System: Debian GNU/Linux 8 (jessie)
    Kernel: Linux 3.16.0-4-686-pae
    Architecture: x86
```

As shown above, this host is running Linux. More specifically, the host is running Debian GNU Linux version 8 (codename jessie) with a Linux 3.16 version kernel on an x86 CPU architecture. Among other things, you can also see that this Linux installation is running on a virtual machine with VMware as the hypervisor. Cool, huh?

Where Do I Find Things?

An operating system has a file system that, similar to a filing cabinet, allows you to store and retrieve data. Most file systems use the concept of *directories*—also called *folders*—and files that are stored inside the directories. Everything in Linux—even hardware—is represented in this folder and file structure.

If you're new to Linux, you might be wondering how the Linux file system compares to something familiar like the Microsoft Windows file system. In Windows, you may be used to drive letters (like the C: drive) being used as the highest point of a storage volume. Linux represents the highest level of the volume differently. The Linux file system can span multiple physical drives, which are all a part of the same tree. The highest point of the Linux file system is the "/," or "root," with all other directories branching down the tree from there, as shown in Figure 2-1.

Interaction with and navigation of the Linux file system is done up and down the tree with commands such as:

- **pwd.** Display the directory you're currently in (short for print working directory)
- **ls**. List out files that are present in the folder
- **cd**. Change directory
- **rm**. Remove files
- **mkdir** and **rmdir.** Make and remove folders or directories, respectively

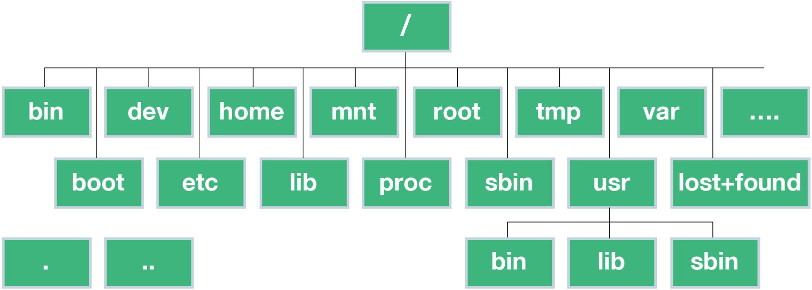


FIGURE 2-1. THE TYPICAL LINUX FILE SYSTEM

Let's do a quick exercise. First, by using the **pwd** command, you can see what directory I'm currently in.

```
david@debian:~$ pwd
/home/david
```

Next, to change to the root directory, you can use the **cd** command.

```
david@debian:~$ cd /
```

To get a simple list of files, you can use the **ls** command. This will display a very concise list of the files and folders that exist in the current directory.

```
david@debian:/$ ls
bin  boot  dev  etc  home  initrd.img  lib  lost+found
media  mnt  opt  proc  root  run  sbin  srv  sys  tmp
usr  var  vmlinuz
```

But, in most cases, you probably want more information than just a simple list of files. Linux uses command line *flags* or *switches* to extend what a command can do. For example, to list out all the files and folders in the current directory, along with full details about each one, you would type **ls -la**. This long listing format then shows you each file and directory, as well as the permissions and access rights for each object (we'll cover file permissions later in this chapter), the name of the user that owns the object (root), the name of the group that owns the object (again, root), the file size, and the data and time that the object was last modified. Here's what this output looks like for the root folder on my test system:

```
david@debian:/$ ls -la
total 88
drwxr-xr-x  21 root root  4096 May 15 11:50 .
drwxr-xr-x  21 root root  4096 May 15 11:50 ..
drwxr-xr-x   2 root root  4096 May 15 12:11 bin
drwxr-xr-x   3 root root  4096 May 15 15:53 boot
drwxr-xr-x  18 root root  3200 Jul 14 01:52 dev
drwxr-xr-x 134 root root 12288 Jul 14 01:55 etc
drwxr-xr-x   3 root root  4096 May 15 15:53 home
lrwxrwxrwx   1 root root    33 May 15 11:50 initrd.img -
> /boot/initrd.img-3.16.0-4-686-pae
drwxr-xr-x  19 root root  4096 May 17 00:41 lib
drwx------   2 root root 16384 May 15 11:49 lost+found
drwxr-xr-x   3 root root  4096 May 15 11:49 media
drwxr-xr-x   2 root root  4096 May 15 11:49 mnt
drwxr-xr-x   2 root root  4096 May 15 11:49 opt
dr-xr-xr-x 150 root root     0 Jul 14 01:52 proc
drwx------   2 root root  4096 May 16 14:29 root
drwxr-xr-x  23 root root   880 Jul 14 01:57 run
```

```
drwxr-xr-x   2 root root  4096 May 17 00:41 sbin
drwxr-xr-x   2 root root  4096 May 15 11:49 srv
dr-xr-xr-x  13 root root     0 Jul 14 01:52 sys
drwxrwxrwt  13 root root  4096 Jul 14 02:02 tmp
drwxr-xr-x  10 root root  4096 May 15 11:49 usr
drwxr-xr-x  12 root root  4096 May 15 12:12 var
lrwxrwxrwx   1 root root    29 May 15 11:50 vmlinuz ->
boot/vmlinuz-3.16.0-4-686-pae
```

Fun with the file system

The image shown in Figure 2-1 also shows you some of the most important directories in the Linux file system, including:

- **/bin, /sbin, /usr/bin,** and **/usr/sbin**. Where executable programs are stored.
- **/dev**. Where files representing hardware devices are stored. For example, if your Linux system had a floppy drive device, there would be a file named fd0 in the dev folder (/dev/fd0).
- **/etc**. Where configuration files are stored.
    - **/home**. Where user home directories are stored, one for each user.
    - **/var**. Where variable-length files, like log files, are stored.
- Of course, not all applications play nice, and not all Linux administrators are consistent. This is just where stuff is *supposed* to go, but things occasionally end up where they don't belong. While there may be some differences between Linux distributions when it comes to where things are located, in general, the baseline directory structure and usage of it should be the same because this is defined by the *file system Hierarchy Standard* (FHS). For more information on the FHS see: https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

Where Are the Applications, and How Do I Run Them?

One of the most common questions from new Linux users who are at a command line is, “What are the applications available to me, and how do I run them?” As mentioned previously, most user tools are found in the directories /bin, /usr/bin and system tools are typically located in /sbin and /usr/sbin. For example, tools like **cp** (to copy a file), **ps** (for process status), and **cat** (to display the contents of a file) are all found in /bin. The great thing is you don’t need to go into any of these directories to run these types of tools because these directories are included in your $PATH variable by default.

The $PATH variable includes all the locations that are searched when you run a command in the CLI. Because the /bin directories are in your path, when you execute the name of any of these sample tools, they will be found. Here’s what your $PATH variable might look like (shown by using the **echo** command to show the $PATH variable):

```
david@debian:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

You can execute applications or commands simply by typing the name of the command if application’s location is in your $PATH. If that application is not in one of the folders listed in your $PATH, you have to do one of the following:

- Navigate to the folder where the application is found and tell Linux that you want to execute the application in that folder, like this:

  ```
  david@debian:~$ cd /opt/app/bin
  david@debian:~$ ./myapp
  ```

  (the “dot slash” refers to the current folder, with the full command saying “in the current directory, execute ‘my app’”)

- Specify the full path of the application when you execute it, like this:

  ```
  david@debian:~$ /opt/app/bin/myapp
  ```

A useful command in determining which command will be run and from what directory it will be run is the **which** command. Use **which** with the executable of a command afterward to get a list of the location of the command that will be executed.

Besides the standard types of Linux tools, there are tens of thousands of applications you can install into Linux in just a few commands. Linux distributions offer *package managers* that help you search online package or application repositories and then download and install just about any application you might want. Package managers also make it easy to update your packages to get the latest version. Examples of package managers are **apt**, **dpkg**, **rpm**, and **yum**. The package manager that is available to you will be determined by the Linux distribution that you have installed. Linux running on Android mobile devices also has its own package manager (similar to the Apple "App Store").

On Debian and Ubuntu systems, you can run **apt list --installed** and get a list of the packages that are already installed, like this:

```
david@debian:~$ apt list --installed
accountsservice/stable,now 0.6.37-3+b1 i386
[installed,automatic]
acl/stable,now 2.2.52-2 i386 [installed]
acpi/stable,now 1.7-1 i386 [installed]
acpi-support-base/stable,now 0.142-6 all [installed]
```

(Output truncated)

Any **apt list** command will result in very long output, so you may consider *piping* it to the "less" pager tool, like this: **apt list | less**. This will show you the output page by page and allow you to press the space bar after each page to see the next page.

What is piping?

You can direct the output of a command to another command. Say you want to get a directory listing that doesn't scroll off the bottom of the screen. You can use the **`less`** paging tool by piping the output of **`ls -al`** to **`less`**. In this case, type **`ls -al | less`** at the command prompt and, when the screen fills up, you are prompted to hit a key to view the second page of the directory. Understanding the pipe character "|" and its usage is important as you begin your Linux journey. In fact, as you get deeper into Linux territory, you will find that the ability to pipe command output to other commands is invaluable when it comes to creating scripts to automate certain functionality.

How Do I Install Applications?

Before you start installing new services, you should typically ensure that you have the list of the most recent versions of available packages from the update repository. This command doesn't actually update any software, but it does make sure you're looking at a list of currently available package versions. You can update the list of packages that are available to you with **`apt update`**, like this:

```
david@debian:/opt$ sudo apt update
Ign http://ftp.us.debian.org jessie InRelease
Get:1 http://ftp.us.debian.org jessie-updates InRelease
[145 kB]
Get:2 http://security.debian.org jessie/updates
InRelease [63.1 kB]
Get:3 http://ftp.us.debian.org jessie Release.gpg [2,373
B]
```

(output truncated)

Then, install a package with **`apt install`**, like this:

```
david@debian:~$ sudo apt install apache2
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  apache2-doc apache2-suexec-pristine apache2-suexec-
custom
The following NEW packages will be installed:
  apache2
0 upgraded, 1 newly installed, 0 to remove and 0 not
upgraded.
Need to get 0 B/208 kB of archives.

After this operation, 361 kB of additional disk space
will be used.
Selecting previously unselected package apache2.
(Reading database ... 137657 files and directories
currently installed.)
Preparing to unpack .../apache2_2.4.10-
10+deb8u8_i386.deb ...
Unpacking apache2 (2.4.10-10+deb8u8) ...
Processing triggers for man-db (2.7.0.2-5) ...
Processing triggers for systemd (215-17+deb8u7) ...
Setting up apache2 (2.4.10-10+deb8u8) ...
```

Important!

For commands requiring elevated privileges, we'll be prepending those commands with the **`sudo`** command, which will be discussed later in this book. For now, you just need to understand that sudo allows you to run the command as an administrator.

In the above example, we used **`apt install`** to install the Apache web server. To verify that a package is installed correctly (and that you installed what you think you installed), you can use **`apt show`**.

```
david@debian:~$ apt show apache2
Package: apache2
Version: 2.4.10-10+deb8u8
Installed-Size: 361 kB
Maintainer: Debian Apache Maintainers <debian-
apache@lists.debian.org>
Replaces: apache2.2-common, libapache2-mod-macro (<<
1:2.4.6-1~)
Provides: httpd, httpd-cgi
Depends: lsb-base, procps, perl, mime-support, apache2-
bin (= 2.4.10-10+deb8u8), apache2-utils (>= 2.4),
apache2-data (= 2.4.10-10+deb8u8)
Pre-Depends: dpkg (>= 1.17.14)
Recommends: ssl-cert
Suggests: www-browser, apache2-doc, apache2-suexec-
pristine | apache2-suexec-custom
Conflicts: apache2.2-common (<< 2.3~)
Breaks: libapache2-mod-macro (<< 1:2.4.6-1~)
Homepage: http://httpd.apache.org/
Tag: role::metapackage, suite::apache
Section: httpd
Priority: optional
Download-Size: 208 kB
APT-Manual-Installed: yes
APT-Sources: http://ftp.us.debian.org/debian/
jessie/main i386 Packages
Description: Apache HTTP Server
 The Apache HTTP Server Project's goal is to build a
secure, efficient and
 extensible HTTP server as standards-compliant open-
source software. The
 result has long been the number one web server on the
Internet.
 Installing this package results in a full installation,
including the
 configuration files, init scripts and support scripts.
```

You can see that the Apache 2.4.10 web server was installed, and it says that this package results in a full installation; however, it also suggests that we install the apache-doc (for documentation) and www-browser (to act as our HTTP client/ web browser) packages.

Getting help

Linux commands can, at times, be confusing and can become complex. In Linux, help is always available!

Use the **man** command (shorthand for "manual") to provide detailed documentation for just about every Linux command. For example:

```
david@Debian$ man ls
NAME
       ls - list directory contents
SYNOPSIS
       ls [OPTION]... [FILE]...
DESCRIPTION
       List  information  about the FILEs (the current
directory by default).  Sort entries alphabetically if
none of
       -cftuvSUX nor --sort is specified.
       Mandatory arguments to long options are mandatory
for short options too.
       -a, --all
              do not ignore entries starting with .
       -A, --almost-all
              do not list implied . and ..
```

(output truncated)

Depending on the command, other options to get help are to append "-h" or just "help" after the command.

Linux Processes, Programs, and Services

When you start a program, in Linux, it will run interactively by default, which means you can interact with it via your terminal session with all input and output presented to you, the user. However, you can also run programs in the background (often called "services") so that you don't see their output and can still use your command prompt to continue your work (and continue running the service even when you are logged out). This can also be useful if you have a program that will take some time to process; you can just put it in the background and be alerted when it is completed.

But how do you know if it's still running, and how do you get a list of every process running on your system? The **ps** command displays a list of running processes in Linux. This command is often coupled with the **-ef** flag to show every process in the long list format shown below. You'll see right at the top that "/sbin/init" is PID (process identifier) #1, and it's owned by root (the superuser—more on the root user later in this chapter).

```
david@debian:~$ ps -ef
UID        PID   PPID  C STIME TTY          TIME CMD
root   1      0   0 01:52 ?     00:00:01 /sbin/init
root   2      0   0 01:52 ?     00:00:00 [kthreadd]
root   3      2   0 01:52 ?     00:00:00 [ksoftirqd/0]
root   5      2   0 01:52 ?     00:00:00 [kworker/0:0H]
```

(Output truncated)

You may want to pipe the **ps -ef** command to **less**, like **ps -ef | less**, to see the output page by page.

If you just enter **ps** by itself, you'll see only your running processes, like this:

```
david@debian:~$ ps
  PID TTY          TIME CMD
 1679 pts/1    00:00:00 bash
 1784 pts/1    00:00:00 ps
```

In this case, you can see that this user is running the **bash** shell, which is providing the command prompt and the **ps** command to show what processes are running (also the command that produced this output).

Linux uses the concept of *system services*, which are long-running programs that are run in the background and typically provide some service on behalf of system users. You can start, stop, and check the status of services with the command **systemctl**, like this:

```
david@debian:/opt$ systemctl status
● debian
    State: running
     Jobs: 0 queued
   Failed: 0 units
    Since: Tue 2017-08-08 20:49:02 EDT; 1h 37min ago
   CGroup: /
           ├─1 /sbin/init
           ├─system.slice
           │ ├─avahi-daemon.service
           │ │ ├─469 avahi-daemon: running
[debian.local
david@debian:/opt$
```

(Output truncated)

Service vs. Systemctl

There is currently a command transition happening, with the **service** command being phased out in favor of the new **systemctl** command. You may see references on websites to the older **service** command. Be aware that the new **systemctl** will soon replace **service** in all Linux distributions.

Importance of Linux Log Files

To be an effective administrator of any type of Linux system, you need to be able to find and search log files to determine the status of the system, including finding any system and application errors. Although applications can put their log file anywhere they like (such as placing them in the application's directory), most Linux system log files will be found in /var/log.

If you go over to /var/log (with **`cd /var/log`**) and do a **`ls -l`** (to list the files in long format), you'll find that there are quite a few Linux system log files.

```
david@debian:~$ cd /var/log
david@debian:/var/log$ ls -l
total 4924
-rw-r--r-- 1 root root 0 Jul 14 01:57 alternatives.log
-rw-r--r-- 1 root root 40586 May 15 12:12
alternatives.log.1
drwxr-xr-x 2 root root 4096  Jul 14 01:57 apt
-rw-r----- 1 root adm  1471  Jul 14 02:17 auth.log
-rw-r----- 1 root adm  24651 Jul 14 01:55 auth.log.1
-rw-rw---- 1 root utmp 0     Jul 14 01:57 btmp
-rw------- 1 root utmp 768   Jul 14 01:53 btmp.1
drwxr-xr-x 2 root root 4096  Jul 14 01:57 cups
```

(Output truncated)

The following are the most important system log files:

- **syslog.** Contains the centralized logging system, called syslog, in which you'll find messages related to the kernel, applications, and more. If configured, this could be the centralized log file for all Linux systems (or even all network devices) in your data center.
- **auth.log.** Contains authentication failures and successes
- **messages.** Contains general system messages of all types

A variety of different tools can be used to view and parse log files, such as:

- **`cat`**. Display the contents of a file
- **`less`**. View a file with pagination and scrolling
- **`grep`**. Search for a string in a file where the usage is **`grep PATTERN [FILE]`**
- **`head`**. See the first lines (head end) of a text file
- **`tail`**. View the last lines (tail end) of a text file. A common use case for tail is to watch the status of a log file in real time with the "-f" flag like **`tail -f /var/log/syslog`**

Even if you ignore the rest of the commands in the previous list, learn to use **`grep`**. I'll be using it later in this book.

Users and Superusers

Just as you might expect with any multi-user operating system, Linux supports the concept of users with differing levels of access. By default, you'll log in as a common user and be able to view most of what's happening on the system, although you're not allowed to view log files as a standard unprivileged user. To be able to reconfigure the system or view log files, you'll need administrative rights. In Linux, these administrative privileges are referred to as *superuser* privileges and are equivalent to the root user, who has a user ID of 0 (zero).

Adding, Modifying, and Deleting User Accounts

It's important to note that Linux user accounts can be added, modified, and deleted with the commands **`adduser`**, **`moduser`**, and **`deluser`**. To add, modify, or delete users, you must have the correct privileges (which are usually the privileges of the root user).

Take a look at this command sequence:

```
david@debian:/$ id
uid=1000(david) gid=1000(david)
groups=1000(david),24(cdrom),25(floppy),27(sudo),29(audi
o),30(dip),44(video),46(plugdev),108(netdev),110(lpadmin
),113(scanner),117(bluetooth)
david@debian:/$ whoami
david
david@debian:/$ sudo id
uid=0(root) gid=0(root) groups=0(root)
david@debian:/$
david@debian:/$ sudo whoami
root
```

Notice in the dialog above how the **id** command was used to see that we were "uid" (user ID) 1000, and how the **whoami** command was used to see that I am a user called "david." From there, I used the **sudo id** command to make sure I was the root user, and the **sudo whoami** command verified that I had become root. You'll note that the **id** command proves that I have the uid of 0 (zero).

Here's a real-world example. Suppose you'd like to view the latest system logs from the Linux syslog file. Doing so isn't possible with a regular user account. To view the syslog file (using the **tail** command, in this case), you must use the **sudo** command:

```
david@debian:~$ tail /var/log/syslog
tail: cannot open ‘/var/log/syslog’ for reading:
Permission denied
david@debian:~$ sudo tail /var/log/syslog
May 15 10:00:08 debian systemd[1]: Reached target
Network is Online.
May 15 10:00:08 debian systemd[1]: Started ACPI event
daemon.
May 15 10:00:08 debian systemd[1]: Listening on ACPID
Listen Socket.
```

```
May 15 10:00:08 debian systemd[1]: Started LSB: RPC
portmapper replacement.
May 15 10:00:08 debian systemd[1]: Reached target RPC
Port Mapper.
May 15 10:00:08 debian systemd[1]: Activated swap
/dev/disk/by-uuid/4a28e383-9cad-4d88-997e-62cfb508d606.
May 15 10:00:08 debian systemd[1]: Activated swap
/dev/sda5.
May 15 10:00:08 debian systemd[1]: Activated swap
/dev/disk/by-path/pci-0000:03:00.0-scsi-0:0:0:0-part5.
May 15 10:06:19 debian systemd[1]: Starting Session 106
of user david.
May 15 10:06:19 debian systemd[1]: Started Session 106
of user david.
```

In the above command sequence, you can see that first there was a permission denied error when trying to view the syslog file, but when the **sudo** command was used (which typically prompts you for the root password, since no other user was specified), the last 10 lines of the log file were shown. Many systems prevent you from becoming the root user with **su** and instead require you to use the **sudo** command.

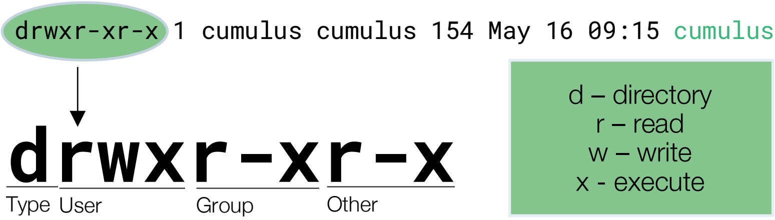
The privileges for who can run what are determined by the /etc/sudoers file, and that file should be edited using the **visudo** command to ensure safe access to a critically important configuration file. For more information on **sudo**, just use **man sudo** to view the manual page.

Files and Permissions

The reason that the user "david" was denied access to the file /var/log/syslog in the previous example is that the user "david" doesn't have permission to access to the file. You can see this if you execute **ls -l /var/log/syslog**:

```
david@debian:~$ ls -l /var/log/syslog
-rw-r----- 1 root adm 9074 May 15 10:17 /var/log/syslog
```

The file is owned by the user "root" and the group "adm." The file permissions are "rw" (shorthand for read/write) for the owner and "r" (shorthand for "read") for the group with no permissions for anyone else. Figure 2-2 shows how file permissions work in Linux.



```
drwxr-xr-x 1 cumulus cumulus 154 May 16 09:15 cumulus
```

FIGURE 2-2. LINUX FILE PERMISSIONS

In the file permissions graphic (Figure 2-2), a "d" on the left tells you whether you are looking at a directory (or folder). Then the three sets of permissions "rwx, r-x, r-x" say whether you can read, write, and execute (or start the application) at the user level, the group level, and the "everyone else" level (others). The type indicator shown in Figure 2-2 identifies the selected object as a directory, hence the "d" as the type. The two most important types of objects in the Linux file system are directories ("d") and files ("-"). There are other possible types as well, but for my purposes here, we'll stick with directories and files.

Knowledge Check

Answer the following questions to check your knowledge concerning the basics of Linux:

- How do you typically log in to Linux?
- What command allows you to become another user to execute a privileged command?
- File and directory permissions are assigned in three groups arranged as "UGO." What do the letters "UGO" represent?

Summary

In this chapter, you learned the basics of Linux administration: how to get Linux, how to log in to Linux, how the Linux file system works, and how to run and view applications. In the next chapter, you'll learn the basics of Linux network administration.

If you want to learn how to communicate with other hosts and devices on your local network and around the world, read on!

Chapter 3

Basics of Linux Network Administration

If you're ever going to do anything interesting with Linux, just like any other OS, you need to be connected to a network, whether it's your own local company network or the public Internet. In this chapter, you'll learn what you need to know to connect your Linux host to the network as well as some tools to help you troubleshoot if things don't go exactly as expected.

Here's what you'll learn:

- The different types of network interfaces in Linux
- How to configure IP addressing
- Networking troubleshooting tools available in Linux
- How to connect multiple network interfaces together to form a bond

We'll kick it off by talking about Linux network interfaces.

Understanding Linux Network Interfaces

Different versions of Linux may name network interfaces differently (see the callout about how Linux network device names are changing). In general, just about all Linux operating systems will have at least two network interfaces. They are:

- **Loopback.** The *loopback* (lo) interface will have an IP address of 127.0.0.1, which represents the host itself. Suppose you want to open a web page running on the same Linux server you are on. You could open http://127.0.0.1 in your web browser. That IP address won't be accessible over the network.
- **Ethernet.** The *ethernet 0* (eth0) interface is typically the connection to the local network. Even if you are running Linux in a virtual machine (VM), you'll still have an eth0 interface that connects to the physical network interface of the host. Most commonly, you should ensure that eth0 is in an UP state and has an IP address so that you can communicate with the local network and likely over the Internet.

Predictable network interface naming convention

If you've been in IT for any length of time, you know that the only constant is change, and that applies to network interface naming conventions as well. Up until very recently, the only naming convention you had to worry about was the one that was just presented. However, with systems featuring systemd v197 or later, common network names such as "eth0" will be assigned more predictable names based on what is known as the *predictable network interface* naming convention. So, rather than an interface named eth0, you may have one named ens3 or enp0s3. Ironically, under older versions of systemd, there was predictability about the existence of eth0. Under the Predictable Network Interface naming guidelines, however, there isn't a guaranteed standard network interface name across systems. The new scheme does have a number of benefits, the most important of which is that the physical interface to interface name association will survive reboots, even if you add hardware. Under the old system, if you added a network adapter, you ran the risk of automatically changing interface names, which had serious consequences for your configuration. For the purposes of this book, we're sticking with the legacy naming convention, but as you continue your Linux networking journey, the predictable network interface naming convention is the new paradigm you need to be aware of.

For more information this change, please visit https://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/

The Linux command to configure network interfaces/devices/links (whatever term you use) is **`ip link`**. In the following example, you can see how **`ip link`** (with no other options) shows two different interfaces, their status, and their MAC addresses associated with each one (we'll talk more about MAC addresses later):

```
david@debian:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
state UNKNOWN mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd
00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 00:50:56:a3:71:f5 brd ff:ff:ff:ff:ff:ff
```

The **`ip link`** command is also used to configure network interfaces. For example, you can change the status of interfaces with **`ip link set [dev] { up | down }`**. You can also reconfigure network interfaces with a command like **`ip link set lo mtu 1500`**.

For more information on the **`ip link`** command use **`man ip link`**.

Going back to the old

If you *really* want or need to use a deprecated command such as **`ifconfig`**, **`arp`**, or **`route`**, you still can. You just have to install the net-tools package on your system. On a Debian system like the one shown in the examples in this book, as a privileged (root) user, run **`apt-get install net-tools`**. If you're using a Linux distribution that uses yum as a package manager, such as CentOS, you can install net-tools using the **`yum -y install net-tools`** command.

If you're only doing this because you're comfortable with the old ways, however, we recommend that you start to phase out your use of these old commands because there's no guarantee that they'll be around forever, they aren't kept up to date, and they may not support all the features of the new commands.

MAC Addresses

A *media access control* (MAC) address is the unique identifier assigned to a network interface at layer 2—the Data Link layer—of the OSI Model. A network interface always has a MAC address—often referred to as the *hardware address*—even if it does not have an IP address. MAC addresses are assigned at the time that a network adapter is manufactured or, if it's a virtualized network adapter, the time that the adapter is created and appears as six groups of two hexadecimal digits each. On the Ethernet interface, eth0, shown above, the MAC address is also called the *link* or *ether address*. In the **`ip link`** output above, you can see that the MAC address in this case is *00:50:56:a3:71:f5*.

Elsewhere in the book

If you're not familiar with the ISO OSI model, that's ok. We'll be talking more about that once we get a bit deeper into networking later in this book (see the sidebar on page 65). Also, if you want to learn about how Ethernet addresses map to IP addresses, we'll be covering that too — stay tuned!

But what if you want to know the IP addresses of these network interfaces? That's next!

IP Addressing

They are unique on the same network, every device has at least one, and addresses typically fall somewhere between 1.1.1.1 and 255.255.255.255. What are they? IP addresses, of course! For this book, I'm going to assume that you already know the basics around TCP/IP, and we'll focus on how to work with them in Linux. Later in this chapter, we'll talk about how to configure IP addresses on your Linux machine.

To view IP addresses, use the **`ip address`** command, or just **`ip addr`**, like this:

```
david@debian:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
state UNKNOWN group default
    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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc mq state UP group default qlen 1000
    link/ether 00:50:56:a3:71:f5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.80/24 brd 192.168.1.255 scope global
dynamic eth0
```

As you can see, the IP address of the loopback interface is 127.0.0.1. What's the address of the eth0 interface in this case? The eth0 "inet" address (IPV4 address) is 192.168.1.80, and it's a dynamic address, received via DHCP, which we'll talk about below.

When it comes to Linux networking tools, there is one that just about everyone has heard of, and that is **ping.** Ping, which began life as an acronym but now enjoys its status as a full-fledged word, is the most basic network test tool around for testing network reachability. It sends out an *Internet Control Message Protocol* (ICMP) packet across the network and notifies you whether there is a response. If a host is up and able to communicate on the network, an ICMP response will be returned. If, however, a host is not reachable, you will get a notice that the host was unreachable or timed out (meaning that the ping test failed). Here's an example of a host that is unreachable:

```
david@debian:~$ ping -c5 192.168.192.196
PING 192.168.192.196 (192.168.192.196) 56(84) bytes of
data.
--- 192.168.192.196 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss,
time 4018ms
```

(The "-c5" was used to send just five ping packets; otherwise, ping will continue forever.)

In these results, five packets were transmitted, and all of them received no response, so there was 100% packet loss. What that means is that this host is unreachable, or down.

Another common Linux network troubleshooting tool is **traceroute**. Traceroute probes the network between the local system and a destination, gathering information about each IP router in the path.

The traceroute command is useful when you think there may be a network issue, such as a host down along a path or a slow response from one of the intermediary nodes, and you want to find out which node is creating the problem. Here's an example:

```
david@debian:~$ traceroute www.apple.com
traceroute to www.apple.com (23.46.180.139), 30 hops
max, 60 byte packets
 1  192.168.1.1 (192.168.1.1)  0.225 ms  0.273 ms  0.283
ms
 2  10.10.0.1 (10.10.0.1)  11.046 ms  11.938 ms  16.645
ms
 3  pool.hargray.net (64.202.123.123)  28.169 ms  22.060
ms  21.785 ms
 4  10ge14-8.core1.atl1.he.net (216.66.49.77)  22.552 ms
22.391 ms  19.566 ms
 5  atx-brdr-01.inet.qwest.net (63.146.26.69)  23.189 ms
21.705 ms  21.952 ms
 6  a23-46-180-139.deploy.static.akamaitechnologies.com
(23.46.180.139)  21.116 ms  21.365 ms  19.497 ms
```

But why are some of those addresses listed on the left actually names instead of IP addresses? That's because *domain name system* (DNS) is replacing the IP with a friendly name. You'll learn about DNS in just a couple of pages!

Remember, every Linux command shown here has verbose instructions on how to use it in the man page. Just type **man *commandname*** to learn more.

DHCP

What if you have dozens, hundreds, or thousands of computers on your network? It would be incredibly time-consuming to manually assign IP addresses and to actually track which machines have which IP address. That's where the *dynamic host configuration protocol* (DHCP) comes in.

DHCP is used to obtain an IP address when a host or device first comes on the network.

DHCP is commonly used for client systems or devices that don't experience any side effects from a periodically changing IP address. On server systems, administrators either manually configure static IP addresses, or they create what are known as static DHCP reservations that are tied to the MAC address of the network adapter. These static reservations ensure that the network adapter will get the same IP address every time it restarts.

Here's how the typical DHCP process works:

1. When a computer starts up, it sends a DHCP request out on the network.
2. Assuming a DHCP server is present, a DHCP server responds with the IP address configuration for that device.
3. That IP address is marked as reserved so that it's not accidentally assigned to some other device.

To learn more about the exact packets that make up the process of obtaining an IP address, see this diagram: http://www.smartpctricks.com/wp-content/uploads/2014/04/DHCP-Packets-Establishment.png.

Note that the prior text said "IP address configuration" and not just "IP address." The IP configuration that is returned to a requesting client contains, at a minimum, the IP address, the IP subnet mask, the IP default gateway, and DNS server details. Most end user devices are configured to use DHCP.

Most operating systems, including Linux, are configured to use a DHCP client to obtain their initial IP address. You can tell an interface is using DHCP if its IP address is set to **DYNAMIC,** as in the output below.

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc mq state UP group default qlen 1000
    link/ether 00:50:56:a3:71:f5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.80/24 brd 192.168.1.255 scope global
dynamic eth0
```

The local configuration file for the DHCP client (called **dhclient**) is at **/etc/dhcp/dhclient.conf**. This is a configuration file that dictates to Linux how it will receive IP configuration information from a DHCP server. To check the status on the DHCP client, you can **cat** the syslog (system log file) and **grep** for dhcp, like this:

```
david@debian:~$  sudo grep -Ei dhcp /var/log/syslog
May 15 08:37:44 debian dhclient: DHCPREQUEST on eth0 to
192.168.1.1 port 67
May 15 08:37:44 debian dhclient: DHCPACK from
192.168.1.1
May 15 08:37:44 debian NetworkManager[425]: <info>
(eth0): DHCPv4 state changed renew -> renew
May 15 08:37:44 debian nm-dispatcher: Dispatching action
'dhcp4-change' for eth0
```

You can find more details on the DHCP client leases in the files /var/lib/dhcp/*.leases

DNS

Computers that connect to each other using TCP/IP (the most prevalent form of connection protocol) talk with each other using IP addresses; however, it would be really painful to have to remember the IP address of everything you want to connect to. Imagine having to recall the IP address of Google each time you wanted to search the web. *Domain name system* (DNS) is used to map IP addresses to names. Everyone is familiar with using their web browser, entering a friendly name like google.com or apple.com, and being taken to the company's website without ever having to type an IP address. It's DNS behind the scenes that is mapping that friendly name to an IP address by doing a DNS lookup. To find out if your Linux host is using DNS, we will be running through some troubleshooting commands, such as **`dig`** and **`nslookup`**, later.

That being said, the basics of DNS in Linux are this:

- A local file called /etc/hosts is used for the first point of lookup for any host name prior to going out to a DNS server on the network. If the name is found there, no further searches are performed. As the superuser, you have the option to edit the hosts file and configure a static name to IP address mapping.
- The /etc/resolv.conf file shows the local domains to be searched and what server names to use for DNS resolution.

For example, here's what a sample **resolv.conf** file looks like:

```
david@debian:~$ sudo cat /etc/resolv.conf
search wiredbraincoffee.com wiredbraincoffee.com.
nameserver 192.168.1.1
```

In this case, the default domain is wiredbraincoffee.com, and the only name server is 192.168.1.1.

DNS Resolution

By default, DNS name resolution works as described here, but is very modular. The hosts portion of /etc/nsswitch.conf can include directory services like NIS+ or LDAP as well.

When it comes to troubleshooting DNS, you should be aware of the following important tools:

- **`dig`.** The *domain Internet roper*, or dig, performs verbose DNS lookups and is great for troubleshooting DNS issues.

- **`getent ahosts`**. The getent tool with the ahosts option enumerates name service switch files, specifically for host entries.
- **`nslookup`.** The name server lookup, or nslookup, performs a variety of different DNS server lookups: mail server lookups, reverse lookups, and more. It's commonly used to look up the IP address of a host.

Network Statistics and Counters

When performing network troubleshooting, it's always good to gather some statistics to answer questions like, "Is the network interface even transmitting any data? Is the interface taking errors? What process is sending all that traffic?"

Here's an example of the **`netstat`** command displaying active processes that have active network interface connections:

Here's an example of the **`ip -s link`** command showing us the statistics for our network links:

```
david@debian:~$ ip -s link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
state UNKNOWN mode DEFAULT group default
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  RX: bytes  packets  errors  dropped overrun mcast
  63339      505      0       0       0       0
  TX: bytes  packets  errors  dropped carrier collsns
  63339      505      0       0       0       0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc mq state UP mode DEFAULT group default qlen 1000
  link/ether 00:50:56:a3:71:f5 brd ff:ff:ff:ff:ff:ff
  RX: bytes  packets  errors  dropped overrun mcast
  410864536  1342597  0       0       0       925004
  TX: bytes  packets  errors  dropped carrier collsns
  20398071   163673   0       0       0       0
```

Here's an example of the **`netstat`** command showing us what our active processes are that have the network interface open:

```
david@debian:~$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address   Foreign Address   State
tcp   0      0      debian:ssh      iMac:52985        ESTABLISHED
tcp   0      0      debian:40980    192.168.1.128:37518 TIME_WAIT
tcp6  1      0      localhost:33904 localhost:ipp     CLOSE_WAIT
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags  Type   I-Node   Path
unix  20     [ ]    DGRAM  8963     /run/systemd/journal/dev-log
unix  6      [ ]    DGRAM  8972     /run/systemd/journal/socket
unix  2      [ ]    DGRAM  15451    /run/user/1000/systemd/notify
```

(output truncated)

And here's an example of the **`netstat -l`** command that shows us the active listening services on this host:

```
david@debian:~$ netstat -l
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address      Foreign Address    State
tcp        0      0 localhost:mysql    *:*                LISTEN
tcp        0      0 *:48875            *:*                LISTEN
tcp        0      0 *:sunrpc           *:*                LISTEN
tcp        0      0 *:ssh              *:*                LISTEN
tcp        0      0 localhost:ipp      *:*                LISTEN
Active UNIX domain sockets (only servers)
Proto RefCnt Flags   Type      State     I-Node Path
unix  2      [ ACC ] SEQPACKET LISTENING 8197   /run/udev/control
unix  2      [ ACC ] STREAM    LISTENING 8200  /run/systemd/journal/stdout
unix  2      [ ACC ] STREAM    LISTENING 15895 /run/user/1000/keyring/pkcs11
```

(output truncated)

How to Configure Network Interfaces

So far, we have covered a lot about how to view and show network information, but nothing about how to change network configurations. Let's cover some of the most common network configurations that someone new to Linux might want to perform.

We'll start off with changing an IP address. When it comes to making any changes in Linux, keep in mind that you can make two types of changes:

- Changes that are immediately effective but are *non-persistent* (meaning they won't survive a restart of the operating system)
- Changes that are effective after the next restart of the OS, known as *persistent changes*

To make an immediately effective change in your IP configuration, you use the **`ip`** command with its variety of command options such as **`link`**, **`route`**, and **`address`**. To use the **`ip`** command set, you'll have to have the iproute2 package installed. It usually is installed by default, but that depends on which version of Linux you are using.

Here we use the **`ip address`** command, like this:

```
root@debian:/home/david# ip address add 10.10.10.10/8
dev eth0
```

However, once the Linux machine is restarted, the default IP address will be back on interface eth0.

To make this IP address change persistent on a Debian or Ubuntu system, you need to edit the file /etc/network/interfaces and add the configuration for eth0. To edit this file, use the nano command like this: nano /etc/network/interfaces. If you are using CentOS or RHEL (Red Hat Enterprise Linux), the same configuration is found in the /etc/sysconfig/network-scripts directory.

Adventures in editing text files

Editing of text files in Linux is usually done with the **vi**, **emacs**, or **nano** commands. The "battle" between vi and emacs is long running, and their followers are fanatical. The vi text editor has been around since 1976 and is known for its lack of user-friendliness. With vi you must already know, or be willing to read the documentation on, how to use specific command options to even start editing text. Similar to vi, emacs has also been around since the 1970's and isn't known for its ease of use. Contrary to vi, emacs has so many plugins (including terminal, calculator, email, and web browser) that it feels like its own operating system. Nano, on the other hand, was created in 1999 and is very user-friendly because it works just like a traditional word processing application. For those who are new to editing text in Linux, my recommendation is to use nano.

To make the IP address change take effect, you can either reboot the host or use the **ifdown/ifup** commands. At that point, the **ip address** command output might look like this:

```
david@debian:~$ ip address show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UNKNOWN group default qlen 1000
    link/ether 00:0c:29:d0:e8:7e brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.72/24 brd 192.168.1.255 scope global
eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fed0:e87e/64 scope link
       valid_lft forever preferred_lft forever
```

You can see that the new IP address has been added, and making the change this way ensures that it is persistent after the Linux OS restart.

ifdown and ifup commands

The **ifdown** and **ifup** commands are used to restart an interface without having to restart a whole server. Once you make a change to a network interface configuration, you need to force that interface to reread its configuration file. You accomplish this by bringing the interface down with the ifdown command and then bringing it back up with the ifup command. When the ifup command is executed, the configuration file is reread and the interface is brought back into operation with its newly minted parameters.

By the way, the ifdown and ifup commands aren't included in the default path on all Linux distributions. As such, you may have to explicitly include the full path to the command. For example, to bring down the eth0 interface, you may need to type /sbin/ifdown eth0. Likewise, to bring the interface back up, you may need to type /sbin/ifup eth0.

The /etc/network/interfaces file is used to tell ifup and ifdown how to configure network interfaces as those interfaces are brought up and down. For example, you would configure an interface with a static IP address by entering the details in the interfaces file. More information on the interfaces file can be found by typing **man interfaces**.

A word of warning: If you're connected to your Linux server via SSH, be mindful of which interface you're working with. If you accidentally bring down the network interface that your SSH connection is using, you might lose remote access to the server.

To recap, here are some of the common network changes that you will probably want to make in order for your Linux networking configuration to persist across reboots:

- Change the state of network interfaces by using the ifdown / ifup command set.
- **Add DNS name servers to an interface** by editing the /etc/resolv.conf file or the /etc/network/interfaces file.
- **Change the hostname of the server** by editing the /etc/hostname and /etc/hosts file, and verifying it with **`hostname -f`**.

Network Interface Bonding

There are times when you need more bandwidth than a single interface can provide, or you want some form of link redundancy in case of a cabling or other network problem. This link redundancy function goes by many names, depending on the vendor: EtherChannel, VMware PortGroups, Bonds, and Link Aggregation Groups (LAG) are just a few. Linux also provides this capability and calls it *bonding*. It allows you to create a single logical network link that is comprised of multiple physical links and that scales up as you add more interfaces, can provide load balancing across the interfaces, and can provide failover protection.

To use network bonding, you need to install the bonding kernel module via the **`modprobe`** command. Modprobe allows you to add additional capability to the Linux kernel. It works like this:

```
david@debian:~$ sudo modprobe bonding
```

At this point, you can create a bond using the iproute2 tools, which allow you to establish the bond as well as set its mode (we'll cover modes more in the next chapter). You can get some hints with **`ip link help`** and **`ip link help bond`**.

Please note that you may read about **`ifenslave`** when searching the Internet; however, that tool has been deprecated with the iproute2 tools taking its place. You will find that many once-common Linux commands become obsolete over time, so make sure to stay current!

You can put interfaces eth0, eth1, and eth2 into a bond like this (Figure 3-1):

```
david@debian:~$ sudo ip link add bond0 type bond mode
802.3ad
david@debian:~$ sudo ip link set eth0 master bond0
david@debian:~$ sudo ip link set eth1 master bond0
david@debian:~$ sudo ip link set eth2 master bond0
```

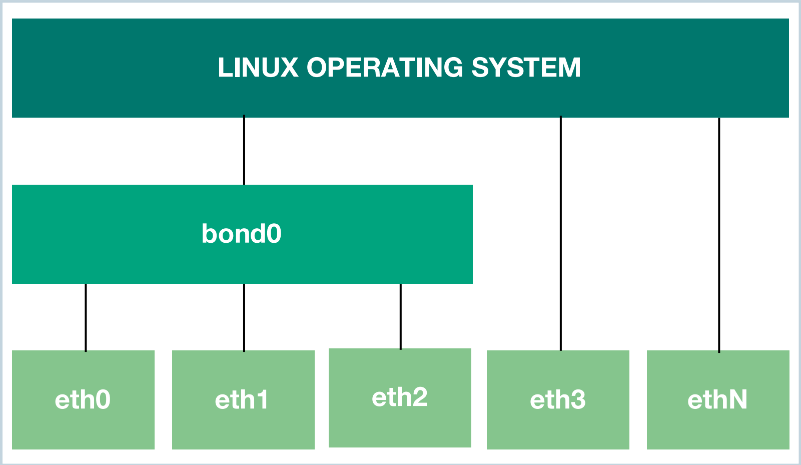


FIGURE 3-1. MULTIPLE ETHERNET INTERFACES BONDED INTO A SINGLE NETWORK INTERFACE

One of the most common parameters to set when creating a bond is the "mode," which is how the bond interacts with the connected network.

Linux network bonding modes

Linux currently supports the following bond modes:

- **balance-rr.** The default, round-robin bonding, which provides load balancing and fault tolerance
- **active-backup**. Provides fault tolerance whereby only one slave can be active at a time and, if it fails, the other slave takes over
- **balance-xor**. Provides fault tolerance and load balancing by transmitting based on hash
- **broadcast.** Provides fault tolerance by transmitting everything on all slave interfaces
- **802.3ad.** IEEE 802.3ad standard for dynamic link aggregation creates aggregation groups for links that share the same speed and duplex in order to provide for fault tolerance and load balancing. 802.3ad is one of the most common types of bonding. 802.3ad uses LACP to communicate with the other side of the bond.
- **balance-tlb**. Adaptive transmit load balancing that doesn't require any special switch support
- **balance-alb**. Adaptive load balancing that doesn't require any special switch support due to its use of ARP negotiation

The most common modes are active-backup and 802.3ad.

Bonding, when incorrectly configured or cabled, can be the source of some pretty messy network problems; 802.3ad mode runs the Link Aggregation Control Protocol (LACP) with the switch or server at the other end of the bond to make sure that everything is connected correctly.

Knowledge Check

Answer the following questions to check your knowledge concerning the basics of Linux networking administration:

- How do you show the IP addresses configured on a Linux host?
- How do you change the IP address configuration of a Linux host?
- How do you test basic network connectivity?

Summary

You should now know the basics of Linux network administration. You just learned about network interfaces, DHCP, DNS, IP address configuration, interface bonding, and more.

Coming up in the next chapter, you'll learn about L2 versus L3 networking, routing tables, bridges, ACLs, VXLANs, VLANs, and more!

Chapter 4

Understanding Linux Internetworking

You might have heard of something called “the Internet,” the largest internetwork ever created. In fact, the term *Internet* (with a capital *I*) is just a shortened version of the term *internetwork*. An internetwork is multiple networks connected together. For example, most companies create some form of internetwork when they connect their *local-area network* (LAN) to a *wide area network* (WAN) in order to connect to one or multiple other LANs. For IP packets to be delivered from one network to another network, IP routing is used — typically in conjunction with dynamic routing protocols such as OSPF or BGP. You can easily use Linux as an internetworking device and connect hosts together on local networks and connect local networks together and to the Internet.

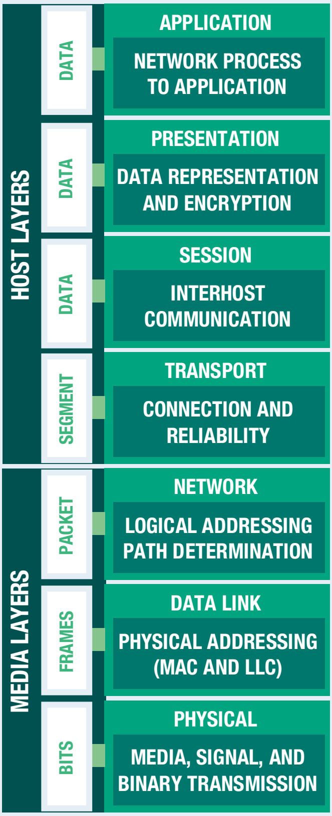
Here’s what you’ll learn in this chapter:

- The differences between layer 2 and layer 3 internetworking
- How to configure IP routing and bridging in Linux
- How to configure advanced Linux internetworking, such as VLANs, VXLAN, and network packet filtering

To create an internetwork, you need to understand layer 2 and layer 3 internetworking, MAC addresses, bridging, routing, ACLs, VLANs, and VXLAN. We’ve got a lot to cover, so let’s get started!

The OSI Model

Before we jump deeply into the networking pool, let's go over the model on which all networking standards are based: the International Organization of Standardization Open Systems Interconnection (ISO OSI) model. This model has been used for decades to describe the networking stack, and it describes the very wires (or lack thereof, in the case of wireless) that transfer data to the applications that operate on the network. This all-encompassing model has driven network development, and most products on the networking market are specifically designed to service one or more layers of the model, which are shown in this callout. In order for an application to "talk" to another application on another machine on the network, that application has to traverse down its own networking stack, ultimately placing its information onto the wireless or media that connects the two machines. The application, however, doesn't need to handle that task itself. It simply presents its data to the next lower layer—the presentation layer—which processes what it gets from the application layer and then sends it down the stack to the session layer and so forth. This is one of the reasons that applications don't need to develop their own communications stacks and can just rely on what is provided to them in the operating system.



HOST LAYERS	DATA	APPLICATION	NETWORK PROCESS TO APPLICATION
	DATA	PRESENTATION	DATA REPRESENTATION AND ENCRYPTION
	DATA	SESSION	INTERHOST COMMUNICATION
	SEGMENT	TRANSPORT	CONNECTION AND RELIABILITY
MEDIA LAYERS	PACKET	NETWORK	LOGICAL ADDRESSING PATH DETERMINATION
	FRAMES	DATA LINK	PHYSICAL ADDRESSING (MAC AND LLC)
	BITS	PHYSICAL	MEDIA, SIGNAL, AND BINARY TRANSMISSION

Layer 2 vs. Layer 3 Internetworking

In the OSI model, layer 1 is the physical layer that includes the physical media used to connect the network. Specifications in this area describe cable qualities and the properties of electrical and optical signals used to move bits around. Examples of layer 1 technologies include Gigabit Ethernet on category 5 cable, 100Gigabit Ethernet on parallel single mode fiber, and 802.11 wireless.

Above that is layer 2, or the data link layer; Ethernet is a broadly deployed layer 2 protocol. Ethernet networking works to encapsulate data and pass that data in the form of *frames*. Frames leverage the Media Access Control (MAC) addresses that we discussed in Chapter 3. An Ethernet frame includes the MAC address of the destination interface on the target system as well the MAC address of the source interface on the sending system so that the recipient device knows where the frame originated. Every Ethernet device, whether it's installed in a server, a switch, or a router, has a unique MAC address on their local network.

Transparent bridges are layer 2 devices that send all frames received on one port out the other bridge ports, based on knowledge of the frame's destination MAC address. Ethernet switches are multiport network bridges. Multiport network bridges learn of the MAC addresses in the network and intelligently forward frames based on the destination MAC address in the frame.

Layer 2 networking works in one of two ways:

- The device has explicit knowledge of a frame's destination address, and the device sends the frame out on the port where it knows the destination exists.
- In the event that the specific destination is unknown, the device falls back to sending the frame to every node in the layer 2 domain via what is known as a *broadcast*.

Definition: Broadcast Domain

In Ethernet networking, layer 2 broadcasts don't go past routers because that is the boundary of the layer 2 network. Thus, the entire Ethernet network is the broadcast domain because no broadcasts pass the Ethernet LAN.

The problem is that these approaches limit the ability for layer 2 networks alone to operate efficiently beyond relatively small-scale locations and very simple topologies. Layer 2 networks suffer from two major limitations. First, they allow for hosts to send traffic to unknown destinations. This causes broadcasts, which impact every node in the broadcast domain. Many networks have been taken offline due to "broadcast storms," or when many hosts are broadcasting at once. In contrast, layer 3 networks do not allow for unknown communication. If a layer 3 router does not have a route to the destination IP address, it will drop the packet instead of broadcasting like layer 2 does.

Second, layer 2 networks have globally unique MAC addresses that are assigned by the manufacturer. There is no organization to these addresses across manufacturers. If you have servers with Intel and Mellanox network cards, the layer 2 MAC addresses will not have any commonality. Again, when comparing layer 2 MAC addresses to layer 3 IP addresses, companies manually plan IP addressing schemes so that there is a hierarchy to these IP addresses. An office may have all IP addresses within it as part of a single IP subnet, like 10.0.0.0, allowing the company to use a single subnet to represent the entire office. With layer 2 addressing, there is no ability to summarize or aggregate MAC addresses; every unique MAC address must be shared with every host in the layer 2 domain.

When a node sends out an IP packet, it consults its routing and neighbor (ARP) tables and sends the packet to the device most likely to get that packet where it needs to go. If the destination is in the same layer 2 network, an entry in the neighbor (or ARP) table tells the sender how to use layer 2 internetworking. When IP devices need to communicate with other IP-based addresses that are outside of their local layer 2 network, the route table may point to a specific router that will get the packet closer to the destination or fall through to the default gateway, which is then responsible for getting the packet to the destination. If no default route exists and a matching route does not exist, the packet will be dropped.

Layer 2 Internetworking on Linux Systems

Initially, Linux networking was focused on end-node networking and layer 3 internetworking; however, the advent of virtualization and containerization changed that forever. Today's Linux networking stack has rich layer 2 internetworking functionality and continues to evolve at a rapid pace.

Bridging

What do you do when you have two different Ethernet networks that need connecting? Build a bridge! Bridges have traditionally been dedicated hardware devices, but you can easily create a bridge in Linux. For example, when you have a Linux host that has two or more network interfaces, you can create a bridge to pass traffic between these interfaces. You can add two interfaces to a Linux bridge with **`ip link set`** and **`ip link add`** using:

```
david@debian:~$  sudo ip link add br0 type bridge
david@debian:~$  sudo ip link set eth0 master br0
david@debian:~$  sudo ip link set eth1 master br0
```

Here's what is happening:

- The first command, **`ip link add`**, is creating a bridge named br0.
- The two **`ip link set`** commands add the two Ethernet interfaces, eth0 and eth1, to the new bridge resulting in a connection between these two interfaces.

Once a bridge is created, you can view the MAC address table, which shows which ports can reach a specific MAC address, with the **`bridge`** command. The command shown in the example below uses **`fdb show`** as its parameter. In this command, fdb stands for forwarding database management, and show is a way for you to see the current contents of this database:

```
david@debian:~$ sudo bridge fdb show
[sudo] password for david:
01:00:5e:00:00:01 dev eth0 self permanent
33:33:00:00:00:01 dev eth0 self permanent
33:33:ff:d0:e8:7e dev eth0 self permanent
01:00:5e:00:00:fb dev eth0 self permanent
33:33:00:00:00:fb dev eth0 self permanent
01:00:5e:7f:ff:fa dev eth0 self permanent
01:00:5e:00:00:01 dev eth1 self permanent
33:33:00:00:00:01 dev eth1 self permanent
01:00:5e:00:00:01 dev eth2 self permanent
33:33:00:00:00:01 dev eth2 self permanent
```

Once the bridge has "bridged," the different Ethernet networks, all the devices on these networks can communicate, at least at layer 2 (see Figure 4-2).

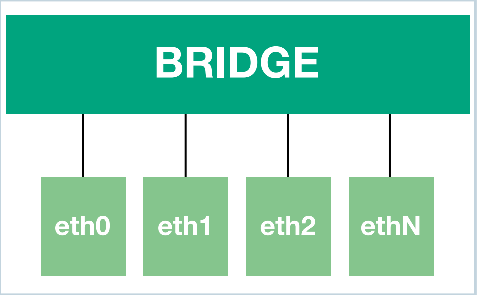


FIGURE 4-2. LINUX BRIDGE CONFIGURATION

Spanning Tree

The downside to big networks is that you can accidentally create loops that feed upon themselves and that can ultimately bring the network down. For example, if you accidentally plug one switch port directly into another port on the same switch, you may have created a loop. You can mitigate these loops through the use of *spanning trees*. It's also important to note that layer 3 has a TTL (time to live) field that reduces the impact of loops — packets eventually die and are dropped — while layer 2 does not have a TTL and will loop a frame forever.

A spanning tree is always recommended for any bridge or device configured with a bridge interface to prevent *bridging loops*, reduce broadcast traffic, and provide automatic failover if you have redundant links. You can perform most spanning tree configurations in Linux by using the **`mstpd-ctl`** command, which controls the *multiple spanning tree protocol daemon* (MSTPD).

Bridges bridge frames, and routers route packets. Modern network switches can do a little of both depending on the hardware and software on the device. The really cool thing about Linux is that it can be used to create both layer 2 and layer 3 switches and routers, allowing you to both bridge and route using Linux.

Bridging Loops

Because bridges forward broadcast packets out every port, the broadcast is amplified by both devices when there are multiple paths between two bridges. For example, if two bridges are connected with two links, the first bridge receives a broadcast frame from an attached host. The bridge will take this single frame and send one copy on each link to the other bridge. This second bridge will receive these two broadcast frames, one on each link, and will make new copies, sending them back on each link. This back and forth broadcast replication, known as a "broadcast storm," will continue forever.

Unlike layer 3 packets, layer 2 frames do not possess a TTL field. A packet contains a special field that is set by the host that first created the packet. Each router along the path will decrement this field by 1. If a misconfiguration in the network causes a similar loop, the TTL field will eventually be decremented to 0 and the packet will be dropped. Because a layer 2 frame does not have this field, there is no limit to how many bridges a frame can pass through. Also, because the packet is being bridged and not routed, the TTL field will never be examined by any of the devices and never decremented. The lack of TTL is one of the major problems with layer 2 networks.

The *Spanning Tree Protocol* (STP) does not add a TTL field to the frame, but it will prevent layer 2 loops from forming, preventing the broadcast storm described earlier. Bridges that speak STP will exchange information about the network using *Bridge Protocol Data Units* (BPDUs). Through this BPDU exchange, the bridges will build a loop-free "tree" of the network. In our two-switch example, STP would disable one of the two links and never send traffic over it, until the active link failed.

The basics of TCP/IP addresses

IP version 4 addressing, known as IPv4, uses a 32-bit number to identify every host/device. These addresses are usually written using *dotted decimal,* such as 192.168.192.168. Every device has a configured subnet mask, such as 255.255.255.0, that tells the device which part of the IP address is used to identify the network and which part is the device.

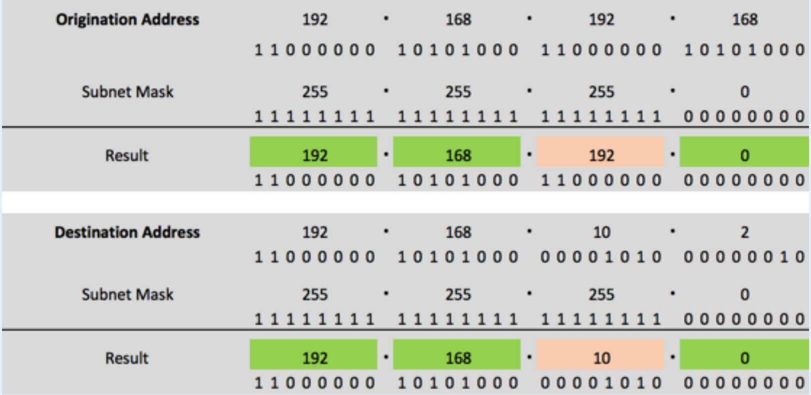
The 32-bit address is broken up into four 8-bit sections called *octets.* For example, the decimal to binary conversation for the above IP address (192.168.192.168) is

11000000 10101000 11000000 10101000.

The conversation of the subnet mask from 255.255.255.0 is

11111111 11111111 11111111 00000000

How, exactly, does your networking stack know that 192.168.10.2 is not in the same network as 192.168.192.168 when using a 255.255.255.0 subnet mask? If you've ever wondered how the math works, the magic lies in the use of the bitwise AND operator. In the figure below, you can see that performing a bitwise AND operation between the origination address and the local network's subnet mask results in a calculation that shows that the local network is 192.168.192.0. When a node in this network wants to communicate with the IP address 192.168.10.2, a similar operation is performed on this destination address with the result indicating that the destination address is 192.168.10.0. Because the destination address has been determined to be non-local, this traffic is sent to the local layer 3 device, typically a router, which then forwards the packet to the correct destination network.



Origination Address	192	168	192	168
	11000000	10101000	11000000	10101000
Subnet Mask	255	255	255	0
	11111111	11111111	11111111	00000000
Result	192	168	192	0
	11000000	10101000	11000000	00000000
Destination Address	192	168	10	2
	11000000	10101000	00001010	00000010
Subnet Mask	255	255	255	0
	11111111	11111111	11111111	00000000
Result	192	168	10	0
	11000000	10101000	00001010	00000000

Layer 3 Internetworking View on Linux Systems

The IP protocol is pretty heavily embedded in Linux systems, and it is the primary (and default) way for Linux systems to communicate with the rest of the world, so we'll start with layer 3 internetworking. One interesting thing to note is that the tables, tools, and processes used by end-nodes to reach other end-nodes are exactly the same as those used by routers (layer 3 internetworking devices) to forward packets to end-nodes.

Neighbor Table

When an IP node wants to communicate with a system in the same layer 2 domain, it looks in its neighbor table, or ARP table, to determine how to construct the Ethernet frame. If the desired destination IP address is not in the neighbor table, the node issues an ARP request, which is broadcast to everyone in the layer 2 domain, that asks, "Please tell me the MAC address for the node with IP address X.X.X.X." Assuming the target device is available, the node with that IP address will respond. In Linux, you view (and manipulate) the Neighbor table using the **`ip neighbor show`** command (also known as **`ip neighbor show`**, **`ip neigh show`**, or even just **`ip n s`**):

```
david@debian:~$ ip neigh show
172.20.10.2 dev eth0 lladdr ac:bc:32:9c:a6:3b REACHABLE
172.20.10.1 dev eth0 lladdr 72:70:0d:4c:6b:64 STALE
```

You'll receive a list of IP addresses that have been recently resolved to MAC addresses, their associated MAC addresses, which interface is used to reach the layer 2 network where they can be reached, and the confidence of knowing these IP/MAC address relationships. Typically, the neighbor table is maintained dynamically based on the ARP protocol; however, it can be manually controlled with the **ip neighbor** command.

IP Routing

The routing table has knowledge of specific networks, or summaries of networks, that a node can reach. Minimally, each routing table will have a "default route" where the node can send any IP packet that is not in an attached layer 2 network. You can view the routing table with the **`ip route show`** command, like this:

```
david@debian:~$  ip route show
default via 172.20.10.1 dev eth0  proto static  metric
1024
172.20.10.0/28 dev eth0  proto kernel  scope link  src
172.20.10.10
```

Here you can see that the routing table knows that the 172.20.10.0/28 network is a locally attached layer 2 network. The routing table also includes a route to the default gateway (172.20.10.1), which Linux calls "default," that will be used to reach any node that isn't on the local network. If you're used to networking on non-Linux systems, you may have seen a default route expressed as something like 0.0.0.0/0.

Routes can be added or deleted from the routing table in a few different ways:

- By assigning IP addresses to node interfaces
- By manually adding or removing them using the **`ip route`** command
- By dynamically inserting them using routing protocols

For example, to create a static route to router 192.168.1.1 through the eth1 interfaces, you would use the **`ip route`** command, like this:

```
ip route add default via 192.168.1.1 dev eth1
```

However, once the host is restarted, this route disappears because it's not persistent. To make this route persistent, you would edit the /etc/network/interfaces file and, after the network device configuration, add a **`post-up`** command with the same **`ip route`** command so that this static route is added every time the Linux host is restarted or the network interface is brought up. Here's an example of what it might look like in the /etc/network/interfaces file:

```
iface eth1 inet static
address 192.168.1.1
netmask 255.255.255.0
post-up ip route add default via 192.168.1.1 dev eth1
```

The purpose of the **`post-up`** command is to add the default route only after the network interface is brought up.

Definition: Free Range Routing

Free range routing (FRR) is an open-source Linux suite of IP routing protocols that includes BGP, IS-IS, LDP, OSPF, PIM, and RIP. Because it integrates with a wide variety of Linux stacks, FRR has a wide range of use cases including connecting hosts, VMs, and containers to the network, Internet access routers, and Internet peering. Based on the Quagga project, FRR is used by many companies for many use cases around the world.

Virtual LANs (VLANs)

You already know that a LAN is a local area network spanning a relatively small physical area. Building on that concept, a *virtual LAN* (or VLAN) allows LANs to span multiple switches across very large networks while still achieving traffic isolation from other networks. VLANs are used to isolate hosts or applications from each other for the purposes of security, data flow, and scale. Individual interfaces can be a part of one or more VLANs. When they are a part of more than one VLAN, in order to maintain some semblance of sanity, the frames traversing that link are tagged with an *IEEE 802.1Q* tag. These tags are an additional piece of information placed at the front of the frame to identify the VLAN. Interfaces carrying multiple VLANs are often called *trunks*. VLANs are configured using both the **`ip link`** and **`bridge`** Linux commands.

Suppose you want a Linux system to have eth1 in one bridge (VLAN11), eth3 in a second bridge (VLAN12), and eth2 in both (i.e. a tagged trunk). First, we make sure the 802.1Q trunking driver is installed. Then we create a bridge, add the ports to the bridge, and make sure the ports are part of the desired set of VLANs. Notice that both eth1 and eth3 used untagged VLANs. However, per the bridge's configuration, traffic from those ports will be placed onto their configured VLANs, which are VLAN 11 and VLAN 12 in this case. Untagged traffic from the trunk port will be placed into the native VLAN, which is VLAN 1 by default.

If you look at the Ethernet frames, you can't tell that the interfaces are part of a VLAN; however, eth2 is a member of both VLANs, and all frames carry the 802.1Q VLAN tag (shown in Figure 4-3).

In the following configuration, we ensure that the 802.1q module is loaded, add bridge0 (br0) as the native VLAN (VLAN 1), add the Ethernet interfaces to br0, assign eth1 and eth3 to their respective VLANs (11 and 12), and bring all interfaces up.

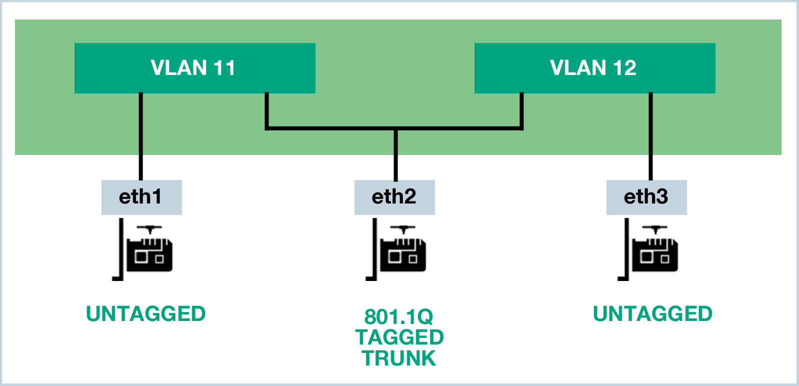


Figure 4-3. Tagged and untagged VLAN traffic

```
sudo modprobe 8021q
sudo ip link add br0 type bridge vlan_filtering 1
sudo ip link set eth1 master br0
sudo ip link set eth2 master br0
sudo ip link set eth3 master br0
sudo bridge vlan add dev eth1 vid 11 pvid untagged
sudo bridge vlan add dev eth3 vid 12 pvid untagged
sudo bridge vlan add dev eth2 vid 11
sudo bridge vlan add dev eth2 vid 12
sudo ip link set up dev br0
sudo ip link set up dev eth1
sudo ip link set up dev eth2
sudo ip link set up dev eth3
```

Much ado about 802.1q

The purpose of the VLAN is to have multiple devices on the same VLAN communicate as if they were the only devices on that network, giving administrators flexibility that they didn't have with physical networks alone. The IEEE networking standard that defines VLANs (or virtual local area networks) is 802.1Q. This standard details how Ethernet frames will have a VLAN tag, or identifier, inserted into them and how Ethernet bridges and switches will handle frames with the VLAN tags. Any Ethernet frame that doesn't have a tag stays on the *native VLAN,* whereas frames that do have tags are only seen by other network devices on that VLAN. Network links between switches that carry multiple VLANs are called *trunk links.*

To help you better understand the configuration above, there are a few things that you should know:

- In the command **`sudo bridge vlan add dev eth1 vid 11 pvid untagged`**, the *vid* parameter is the VLAN ID. VLAN IDs are used to specify which VLAN the interface is assigned to. The *pvid* parameter specifies the private VLAN ID. In this case, the private VLAN is left untagged.
- In the example above, MAC address tables are per-VLAN. If you were to look at the MAC address table in VLAN 11 and VLAN 12, you'd find that they would be very different. The trunk link would have a combination of MAC addresses from both VLAN 11 and 12.

Here are a few useful commands to see what's going on:

- **`bridge link show`.** Check the status of the bridge links
- **`bridge vlan show`.** Check the status of the VLANs traversing the bridge
- **`bridge fdb show`.** View the forwarding database

Overlay Networks with VXLAN

An *overlay network* is essentially a computer network that is built on top of another network. The overlay network is commonly called the "virtual network" that runs on top of an existing "physical network" (and thus, the "overlay" and "underlay" terminology). It's important to note that VLANs discussed in the last section are an example of a virtual network overlay on a L2 network.

What is encapsulation?

Encapsulation is when one piece of data or packet on a network is wrapped up in another type of data or network packet. For example, a text file could be encapsulated in an archive file. In networking, encapsulation is used as a means to move traffic that might otherwise not be able to traverse the communications mechanism. For example, you may encapsulate an IP packet encapsulated in an Ethernet frame to move traffic between local hosts, but encapsulation can even happen between the same two protocols. IP could be encapsulated with IP. A common modern-day example of encapsulation is the iSCSI storage protocol. In an iSCSI system, iSCSI commands and a storage payload are encapsulated inside a TCP packet, which is encapsulated inside an IP packet, which is, in turn, encapsulated inside an Ethernet packet. This multi-level encapsulation process enables what would have been local SCSI storage commands to transparently traverse an Ethernet-based TCP/IP network.

VXLAN, or *Virtual eXtensible LAN,* is an overlay network that runs on top of an existing IP network. VXLAN has a number of different use cases, including creating a massively scalable network (up to 16.7 million possible networks) and connecting data centers at layer 2 across a layer 3 network. VXLAN encapsulates frames with layer 3/4 (IP/UDP), sending them over both layer 2 and layer 3 networks. The benefits of VXLAN on layer 2 (IP) networks are global addressing, better scale, more resiliency, and better use of available bandwidth.

The connections between endpoints are called VXLAN tunnels. These VXLAN tunnels are encapsulating traffic as it flows across the network between the VXLAN tunnel endpoints (also called VTEPs, or VXLAN Tunnel EndPoints). The VXLAN encapsulation allows for the transport of traffic over networks that end hosts do not need knowledge of. This means a host could send an ARP request to another host, across the network, through a VxLAN tunnel, and never know about the VxLAN tunnel or the underlay network it travels through.

VTEPs can be implemented in hardware or software. The configuration of VTEPs and creation of the overlay networks is typically implemented using a commercial controller, such as VMware NSX or Midokura MidoNet, or using a protocol such as BGP EVPN over VXLAN (explained in the next chapter). However, some people build their own special purpose controllers. Regardless of which of these techniques you use, Linux provides the underlying VTEP building block.

If you have two Linux systems and you want to bridge them with VXLAN, you would install a bridge on both systems, add a local IP address to that bridge, and add a VTEP to that bridge pointing the VTEP to the other Linux host (shown in Figure 4-4).

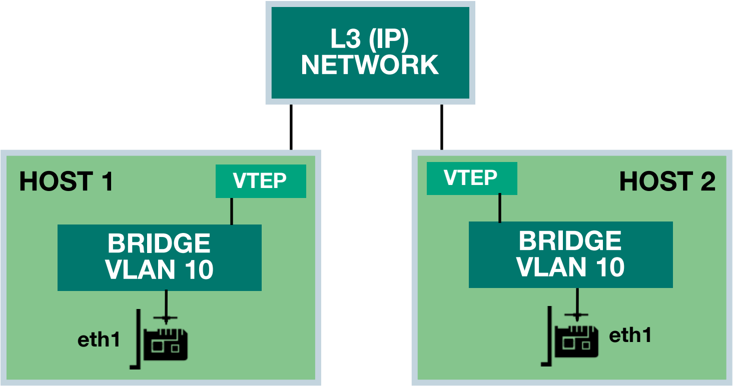


FIGURE 4-4. TWO LINUX HOSTS CONNECTED WITH VXLAN

Linux System 1

```
sudo ip link add br0 type bridge vlan_filtering 1
sudo ip link add vlan10 type vlan id 10 link bridge
protocol none
sudo ip addr add 10.0.0.1/24 dev vlan10
sudo ip link add vtep10 type vxlan id 1010 local
10.1.0.1 remote 10.3.0.1 learning
sudo ip link set eth1 master br0
sudo bridge vlan add dev eth1 vid 10 pvid untagged
```

Linux System 2

```
sudo ip link add br0 type bridge vlan_filtering 1
sudo ip link add vlan10 type vlan id 10 link bridge
protocol none
sudo ip addr add 10.0.0.2/24 dev vlan10
sudo ip link add vtep10 type vxlan id 1010 local
10.3.0.1 remote 10.1.0.1 learning
sudo ip link set eth1 master br0
sudo bridge vlan add dev eth1 vid 10 pvid untagged
```

Now these two systems both exist on the 10.0.0.x/24 layer 2 network (via the VXLAN overlay) even though they are connected by a layer 3 IP fabric. It's also worth noting that the hosts are completely isolated from the underlying layer 3 network.

Knowledge Check

Answer the following questions to check your knowledge concerning Linux internetworking:

- What's the difference between layer 2 and layer 3 and internetworking?
- What are some of the different types of VLANs?
- What is VXLAN and how does it help you?

Summary

You should now have a good understanding of the basics of Linux internetworking. In this chapter, you learned about layer 2 versus layer 3 networking, bridging, routing, traffic filtering, and VXLAN. I hope that you have enjoyed the chapter!

To learn about running a network operating system with Cumulus Linux, head to the next chapter.

Chapter 5

Cumulus Linux

Cumulus Linux is a Linux distribution that is focused on layer 2 and layer 3 internetworking — switching and routing, respectively — using off-the-shelf whitebox switching hardware to accelerate the packet processing that would be done by software on traditional servers (see Figure 5-1). This allows networking hardware controlled by Cumulus Linux to achieve packet processing rates and functionality on par with traditional switching and routing vendors such as Cisco Systems, Juniper Networks, and others.

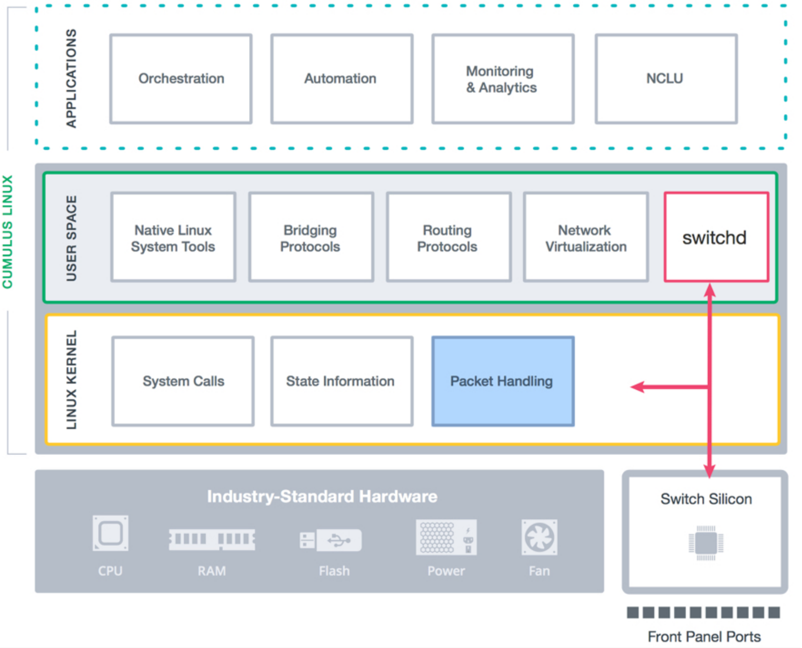


Figure 5-1. The Cumulus Linux architecture

It's important to know that the high-performance switched Ethernet interfaces in Cumulus Linux are accelerated by hardware, and Cumulus has made the decision to prefix these devices with "swp" (short for *switch port*). Even if you went to another Linux distribution and renamed the "eth" ports with "swp," they wouldn't be accelerated because they're missing Cumulus' secret sauce.

Hardware acceleration

While Linux networking can work on just about any hardware, Cumulus Linux is best run on commodity bare-metal switches that are hardware accelerated. The "hardware acceleration" portion of that means that the switches contain hardware called *ASICs*, specially designed to switch frames and route packets, similar to how a graphics card is specially designed for graphics. These ASICs are what make routers and switches different from regular servers and allow them to process hundreds of gigabits or even terabits of network traffic per second.

Linux at the Core

Cumulus Linux makes complete use of the Linux networking model that has been discussed in this book. All of the Linux networking tools you've read about in prior chapters work on Cumulus Linux systems. The Cumulus Linux distribution stays on the leading edge of Linux networking, contributing to the rapid advancement of Linux kernel networking functionality.

Latest and Greatest Networking Protocols

Internetworking requires protocols to interact with network peers and provide the services for the underlying network model. For example, if you want a Linux system to act as a router without having to painfully and manually manage what could be thousands of static routing rules, you need to run a routing protocol, such as BGP, on the Linux system so that you can automatically share IP routes with the rest of the network. Cumulus Linux uses the following:

- **FRRouting**. OSPF, BGP, PIM routing protocols for layer 3 internetworking
- **mstpd**. 802.1 spanning tree protocols for layer 2 internetworking
- **Linux kernel LACP**. For link aggregation
- **MLAG**. For multi-chassis link aggregation

Network Command Line Utility (NCLU)

Throughout this book, you have learned the power that Linux offers when used on the network. Linux networking offers numerous commands, many of which seemingly perform similar functions, and numerous text files that can be edited to manage network configurations.

This complexity occurs because baseline Linux networking relies on the wide variety of tools that we've spoken about in prior chapters. However, the differences between these Linux networking tools can be daunting, even for experienced Linux administrators. Imagine now just how overwhelming they may be for those who are early in their Linux networking journey.

Instead of trying to administer a Linux-powered network with hundreds of command and configuration files, Cumulus Linux includes a command line utility as part of the NCLU package that is invoked by the **`net`** command to provide a consistent and helpful user interface.

As we explore internetworking use cases throughout this chapter, I will lean on net and show you how to leverage it while using Cumulus Linux. One of the most useful facilities in NCLU is the built-in help and examples (as shown below); I'll point this out throughout the chapter.

```
$ net help
Usage:
    # net <COMMAND> [<ARGS>] [help]
    #
    # net is a command line utility for networking on
Cumulus Linux switches.
    #
    # COMMANDS are listed below and have context
specific arguments which can
    # be explored by typing "<TAB>" or "help" anytime
while using net.
    #
    # Use 'man net' for a more comprehensive overview.
    net abort
    net commit [verbose] [confirm] [description
<wildcard>]
    net commit delete (<number>|<number-range>)
    net commit permanent <wildcard>
    net del all
    net help [verbose]
    net pending [json]
    net rollback (<number>|last)
    net rollback description <wildcard-snapshot>
    net show commit (history|<number>|<number-
range>|last)
    net show rollback (<number>|last)
    net show rollback description <wildcard-snapshot>
    net show configuration
```

```
[commands|files|acl|bgp|multicast|ospf|ospf6|interface
<interface>]
Options:
    # Help commands
    help     : context sensitive information; see
section below
    example  : detailed examples of common workflows
    # Configuration commands
    add      : add/modify configuration
    del      : remove configuration
    # Commit buffer commands
    abort    : abandon changes in the commit buffer
    commit   : apply the commit buffer to the system
    pending  : show changes staged in the commit buffer
    rollback : revert to a previous configuration state
    # Status commands
    show     : show command output
    clear    : clear counters, BGP neighbors, etc
```

Building a Better Bridge

One of the most basic networking use cases is a single transparent bridge. In our example, we'll put the interfaces named swp1, swp2, and swp3 into a transparent bridge with swp3 connecting back into our layer 2 bridge infrastructure (Figure 5-2).

```
net add bridge bridge ports swp1,swp2,swp3
net commit
```

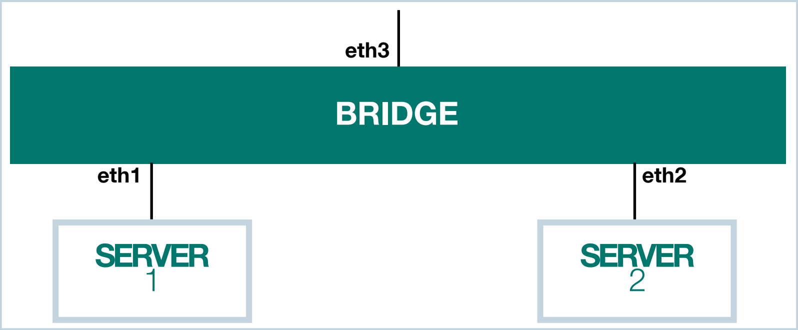


FIGURE 5-2. TRANSPARENT BRIDGE USING SPANNING TREE

This simple example has a few noteworthy things going on. The first is that we don't need to use the **`sudo`** command for privileged access. NCLU makes sure that the user has permission to invoke privileged commands (or belongs to a group that has permission). The second is that **`net`** puts commands into a "commit buffer" so that you can issue a bunch of commands, review them in a pending state (with **`net pending`**), correct them as needed, and then "commit" them to the system with **`net commit`**.

Two Links Are Better Than One

A very typical layer 2 edge use case is using two switches to act like one in a bond to a server. This provides for link- and switch-level redundancy. These types of connections are called *multi-chassis link aggregations* (MLAG), and they are typical of server connections in just about any server deployment.

Figure 5-3 shows such a deployment with swp1 and swp2 connected to servers (each is part of an 802.1ad bond on the server side), swp3 and swp4 connected back to the network core, and swp5 and swp6 acting as "peer links" between the two switches that form the redundant pair. In the example, 100 VLANs are trunked to each of the servers. Try **`net example clag`** for a few MLAG use cases and **`net example clag`**

l2-with-server-vlan-trunks for something close to what is described here:

```
$ net add clag peer sys-mac 44:38:39:FF:00:01 interface
swp5,swp6 primary
$ net add vlan 100-199
$ net add clag port bond bond-to-spines interface swp3-4
clag-id 500
$ net add clag port bond host-01 interface swp1 clag-id
1
$ net add clag port bond host-02 interface swp1 clag-id
2
$ net commit
```

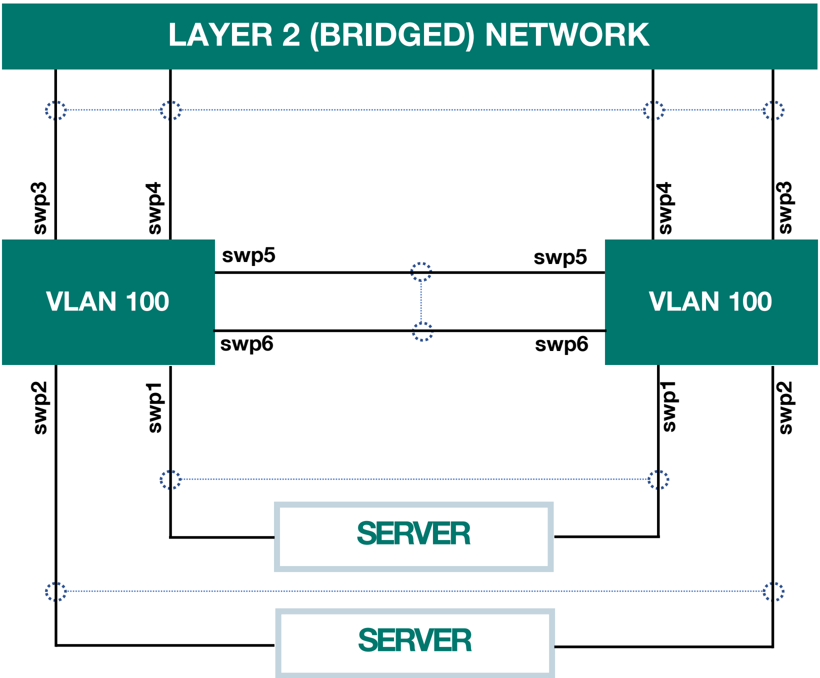


Figure 5-3. Two switches connected with MLAG

IP Fabrics Are Easy

A recent trend in modern network architecture (especially in data centers) is to reduce the size of the broadcast domains and use layer 3 (IP routed) internetworking to create *fabrics*. A fabric is a simple, high-speed, layer 3 network. The motivation behind this trend is that IP networks scale better than layer 2 networks and behave better in the face of unfortunate misconfigurations and failures.

Traditionally, layer 3 fabrics have been complex to configure because every interface on a switch/router needs to exist on an IP subnet with its link peer — a painstaking undertaking. Recent implementations of BGP and OSPF, such as FRRouting in Cumulus Linux, include the ability to connect routers via point-to-point links using “unnumbered” interfaces.

What is the leaf-spine network topology?

Local area networks were originally designed with a “three-tiered” network topology made up of the Core, Aggregation/Distribution, and Access layers. The spanning tree (STP) loop prevention protocol was commonly used to prevent loops. As modern data centers became much more dynamic, network architects realized the inefficiencies in the three-tiered architecture and came up with a better design.

The *leaf-spine network topology* was introduced to ensure that all devices have the exact same number of segments to the core. The end result is that the leaf-spine network has consistent network delay and low latency. This is possible because there are only two layers. The leaf-spine network topology is best used in network data centers to solve “east-west” data center traffic (traffic between hosts in the data center).

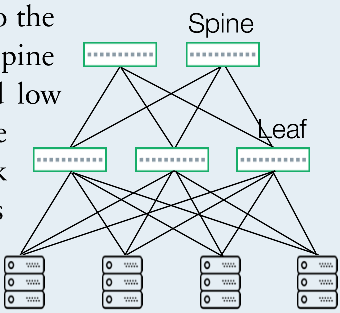


Figure 5-4 shows the configuration of a leaf switch in a layer 3 leaf-spine network built using BGP unnumbered. The leaf switch has a bridge with swp1-4 that has the 10.0.0.0/24 IPv4 subnets. Swp5 through swp8 are connected to spines using BGP unnumbered, advertising reachability of the bridge subnets to the rest of the network.

BGP unnumbered uses automatically assigned IPv6 addresses on the unnumbered interfaces. There is no requirement for the loopback interface (covered in Chapter 3), but it is recommended in order to uniquely identify the network device that is putting routes into BGP (the BGP Router-ID).

```
$ net add bgp autonomous-system 65001
$ net add loopback lo ip address 10.1.0.1/32
$ net add bgp ipv4 unicast network 10.1.0.1/32
$ net add vlan 1 ip address 10.0.0.1/24
$ net add bgp ipv4 unicast network 10.0.0.1/24
$ net add bgp neighbor swp5-8 interface remote-as
external
$ net add bgp ipv4 unicast neighbor swp5-8 activate
$ net add bridge bridge ports swp1-4
$ net commit
```

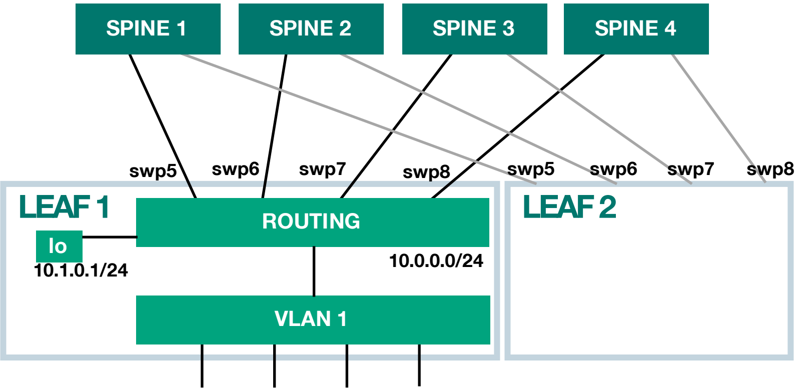


Figure 5-4. Creating a Layer 3 fabric with BGP

BGP EVPN—L3 Network Virtualization for Network Engineers

Many networks have the scale that requires layer 3 internetworking; however, some applications still require layer 2 peering over the layer 3 fabric. One example of where this can be extremely useful is VMware's vMotion. The Ethernet Virtual Private Networks facilities built into FRRouting's BGP daemon allows us to use BGP to build both the IP fabric as well as any distributed layer 2 overlays that are needed to support your applications. BGP EVPN will take any MAC address learned and advertise it to the remote EVPN peers. This allows each leaf in the network to know where to send the Layer 2 VxLAN traffic without flooding or the need for spanning tree.

This example builds on the network defined in the last section on IP fabrics. This time, VLAN 100 provides layer 2 connectivity via an overlay network. You'll add a Virtual Tunnel EndPoint (VTEP) to VLAN 100, send that VLAN tagged to all the hosts on the bridge, and advertise all the layer 2 hosts to the rest of the network with EVPN.

The details of the configuration show a leaf switch in a layer 3 leaf-spine network built using BGP unnumbered. The leaf switch has a default VLAN (1) with swp1-4 that has the 10.0.0.0/24 IPv4 subnet and a second VLAN (100) on swp1-4 that is tagged. Swp5 through swp8 are connected to spines using BGP unnumbered, advertising reachability of the bridge subnets to the rest of the network. VLAN 100 also has a VTEP and is advertised via BGP EVPN. (See Figure 5-5.)

```
$ net add bgp autonomous-system 65001
$ net add loopback lo ip address 10.1.0.1/32
$ net add bgp ipv4 unicast network 10.1.0.1/32
$ net add vlan 1 ip address 10.0.0.1/24
$ net add bgp ipv4 unicast network 10.0.0.1/24
$ net add interface swp1-4 bridge trunk vlans 100
$ net add vxlan vtep100 vxlan id 100
$ net add vxlan vtep100 vxlan local-tunnelip 10.1.0.1
$ net add vxlan vtep100 bridge access 100
$ net add vxlan vtep100 bridge learning off
$ net add vxlan vtep100 mtu 9216
$ net add bgp neighbor swp5-8 interface remote-as
external
$ net add interface swp5-8 mtu 9216
$ net add bgp neighbor swp5-8 interface remote-as
external
$ net add bgp ipv4 unicast neighbor swp5-8 activate
$ net add bgp evpn neighbor swp5-8 activate
$ net add bgp evpn advertise-all-vni
$ net commit
```

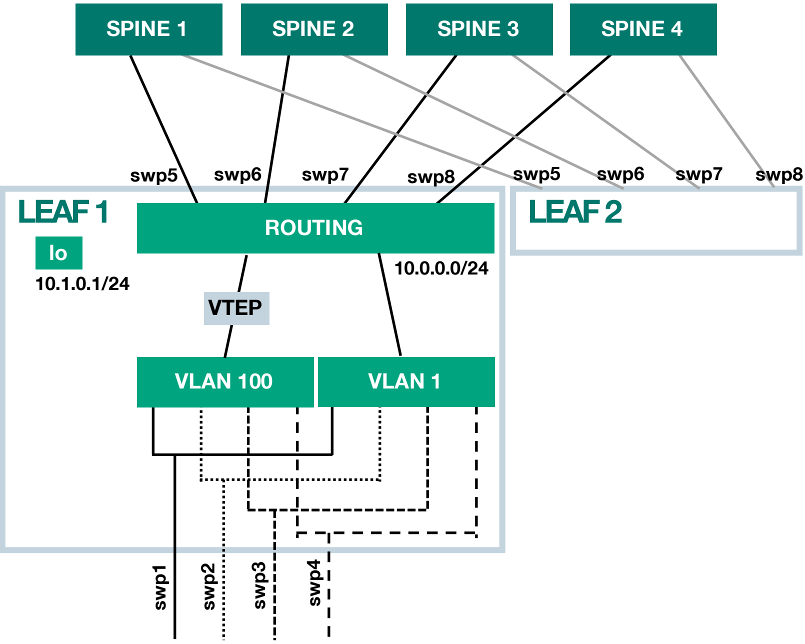


Figure 5-5. Creating a Layer 3 Fabric with EVPN

Note: The VXLAN header used to build the layer 2 network in EVPN makes Ethernet frames larger than the default of 1518, so you need to include the *maximum transmission unit* (MTU). In this case, set it to 9216 (large enough to support “jumbo” frames) so that you don’t have to worry about it ever again.

These use cases are just four examples of how Linux networking can be easy, efficient, and powerful. If you'd like to try out more use cases and commands, we recommend downloading Cumulus VX, a free prototyping environment where you can test out your new Linux networking skills.

Knowledge Check

Answer the following questions to check your knowledge concerning Linux internetworking:

- What makes Cumulus Linux unique?
- How does the NCLU help you?
- How does Cumulus Linux help you to bond links together?
- What is EVPN?

Your Cumulus Linux Action Plan

In this book, you've learned the basics of Linux — from how to log in to advanced Linux network configuration. With your newfound Linux awareness, what's your next step?

Tons of excellent resources are available in the Linux community, including blogs, documentation, and videos. As a part of that community, Cumulus Networks offers a plethora of learning resources as well. Here's your action plan for taking the next step with Cumulus Linux:

Step 1: Gain Access to Cumulus Linux. You can do so in two different ways:

- **Access Cumulus in the Cloud.** This is a pre-built virtual Cumulus lab environment available at no cost. You can sign up to access Cumulus in the Cloud at https://cumulusnetworks.com/products/cumulus-in-the-cloud/

OR

- **Download the Cumulus virtual appliance, Cumulus VX.** Cumulus VX is available for VMware, VirtualBox, KVM, and Vagrant. The Getting Started Guide will walk you through how to deploy it, power it on, log in, and start configuring all the Linux networking that was demonstrated in this book. Download Cumulus VX at https://cumulusnetworks.com/products/cumulus-vx/

Step 2: Check out the Cumulus Learning Resources at https://cumulusnetworks.com/learn/web-scale-networking-resources/

Here you'll find case studies, videos, validated designs, and white papers that will show how Cumulus Linux is being used in real data centers around the world.

Step 3: Join the conversations about Cumulus Linux in the Cumulus Slack Community at https://cumulusnetworks.slack.com/