Red Hat Enterprise Linux 8 Administration

Master Linux administration skills and prepare for the RHCSA certification exam

Miguel Pérez Colino | Pablo Iranzo Gómez | Scott McCarty

Red Hat Enterprise Linux 8 Administration

Master Linux administration skills and prepare for the RHCSA certification exam

Miguel Pérez Colino

Pablo Iranzo Gómez

Scott McCarty

Packt

BIRMINGHAM—MUMBAI

Red Hat Enterprise Linux 8 Administration

Group Product Manager: Vijin Boricha

Publishing Product Manager: Shrilekha Inani

Senior Editor: Shazeen Iqbal

Content Development Editor: Romy Dias

Technical Editor: Arjun Varma

Copy Editor: Safis Editing

Project Coordinator: Shagun Saini

Proofreader: Safis Editing

Indexer: Subalakshmi Govindhan

Production Designer: Shankar Kalbhor

First published: September 2021

Production reference: 1160921

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-80056-982-9

www.packt.com

To all the people contributing to and being part of the open source community, building it one piece at a time, pushing its boundaries to make huge things.

Contributors

About the authors

Miguel Pérez Colino is an experienced IT enthusiast with a clear orientation towards open source software and open standards. He has an extensive background in IT, from operations to the architecture of large deployments and from identifying and prototyping solutions to defining IT strategies. He has delivered large projects, including NATO interoperable command and control systems in defense, extensive large deployments in retail (>15,000 devices), and digital transformation in the finance sector.

Now, as part of the Cloud Platforms Business Unit of Red Hat, he works as a senior principal product manager building tools and defining methodologies to ease modernization and migration for customers, enabling the use of open source.

I want to thank my family, Sonia, Miguel, and Matilde, for the support and patience they have had with me during the months that I have been writing this book. Also, to Pablo Iranzo for being part of this adventure, and without whom we would not have been able to finish it. The whole Packt editing team has helped this first-time book authors immensely, as well as Scott and Gunnar, and I'd like to thank all of them for their contributions.

Pablo Iranzo Gómez is a software engineer whose Linux exposure began while studying physics, a period during which he was also involved in LUGs and some projects related to HPC clusters and system administration and consultancy.

Currently, he is a senior software engineer in Red Hat's SolEng department, focusing on OpenShift, with experience in consulting, cloud technical account management, OpenStack software maintenance in industries such as hospitality, retail, airlines, government, telco, 5G, partners, IT covering system administration and automation, virtualization, PaaS, support, the cloud, and so on, having a broad understanding of different views, needs, and risks across the industry.

Pablo was born in and lives in Valencia, Spain with his family.

I want to thank my wife, Eva, for all the support she offered me enrolling in this endeavor; to my kid, Pau, for all the happy moments that empowered me to drive forward; to Javier, for the countless hours of unconditional understanding; and I want also to say thanks to Miguel for offering me to help him with this; and, of course, the whole Packt team, who have helped, guided, and advised me during the whole process.

At Red Hat, **Scott McCarty** is a principal product manager for the container subsystem team, which enables key product capabilities in OpenShift Container Platform and Red Hat Enterprise Linux. His focus areas include container runtimes, tools, and images.

Scott is a social media start-up veteran, an e-commerce old-timer, and a weathered government research technologist, with experience across a variety of companies and organizations, from seven-person start-ups to 15,000-employee technology companies. This has culminated in a unique perspective on open source software development, delivery, and maintenance.

First, I would like to thank all of the people in the open source community that have contributed to creating code and documentation that helps us all achieve more. I would like to thank all of the people that build, distribute, document, and release Red Hat Enterprise Linux and bring it into environments that I never thought Linux could reach. Finally, I'd also like to thank all of the sysadmins out there that run all of this software to prevent downtime and move projects forward.

About the reviewer

Matthew Bach is a senior specialist solutions architect at Red Hat, focused on the emerging technologies portfolio, such as OpenShift, middleware, and Ansible for DoD customers. He has been in the IT industry for over 17 years and is a 13-year US Navy information systems technician veteran. Matthew currently holds several Red Hat and other security-related certifications. When Matthew is not working with his customers on exciting technologies, he likes to spend time with his family, or on a mountain bike exploring the many trails Virginia has to offer.

To my wife, Ashley, with love: Thank you for your love, support, and patience through all of my endeavors in the endless pursuit of this career. I can think of no one I'd rather be with for the rest of my days than you and our five awesome kids.

To Maurice Carroll, thank you for the countless hours you spent cultivating an interest in computers with a teenage grandson, and my very first box of Red Hat Linux. You lit a spark in me that remains today.

Table of Contents

3

Basic Commands and Simple Shell Scripts

4

Tools for Regular Operations

5

Securing Systems with Users, Groups, and Permissions

6

Enabling Network Connectivity

7

Adding, Patching, and Managing Software

Section 2: Security with SSH, SELinux, a Firewall, and System Permissions

8

Administering Systems Remotely

9

Securing Network Connectivity with firewalld

10

Keeping Your System Hardened with SELinux

11

System Security Profiles with OpenSCAP

Section 3: Resource Administration – Storage, Boot Process, Tuning, and Containers

12
Managing Local Storage and Filesystems

13
Flexible Storage Management with LVM

14
Advanced Storage Management with Stratis and VDO

15
Understanding the Boot Process

16
Kernel Tuning and Managing Performance Profiles with tuned

17
Managing Containers with Podman, Buildah, and Skopeo

Section 4: Practical Exercises

18
Practice Exercises – 1

19
Practice Exercise – 2

Other Books You May Enjoy

Index

Preface

Linux is everywhere, from personal devices to the largest supercomputers, from the computer labs at universities to Wall Street or the International Space Station, and even Mars! **Red Hat Enterprise Linux** (**RHEL** for short) is the Linux distribution most used in enterprise environments and knowing how to use it is a key skill for anyone in technology. No matter whether you are completely into managing infrastructure or you are a developer interested in knowing more about the platform you want to deploy on, learning about Linux – and, more precisely, about RHEL – will help you be more effective and could even boost your career.

In this book, we cover the basic RHEL administration skills from a very practical perspective, providing examples and tips that we have learned from our experience in "the trenches." You will be able to follow it from beginning to end, being able to practice with each step while learning about how things are built and why they behave as they do.

We hope you enjoy this book, that you make the most of it, and that you end up, after reading it, with a strong foundation of RHEL administration skills. That's what we wrote it for.

Enjoy reading ... and practicing!

Who this book is for

Anyone that aspires to build and work on IT infrastructures using Linux will benefit from this book as a reference for different useful tasks, tips, and best practices. It will help anyone seeking to pass the Red Hat Certified Systems Administrator (RHCSA) exam, although it will be no substitute for the official training, in which labs and specially crafted tests will be run during the whole process. The scope of the book is adjusted to the RHCSA, extending it with advice from real-world experience and many practical examples.

What this book covers

Chapter 1, *Installing RHEL8*, covers the installation of RHEL, from obtaining the software and the subscriptions to the installation of the system itself.

Chapter 2, *RHEL8 Advanced Installation Options*, introduces advanced use cases for the installer, including deploying instances in the cloud and automating the installation.

Chapter 3, *Basic Commands and Simple Shell Scripts*, explains the daily commands that will be used during system administration, and how they can be automated via shell scripting.

Chapter 4, *Tools for Regular Operations*, shows which simple tools are available in our system that can be used for regular daily operations such as starting or enabling a system service or reviewing what is going on in the system through logs.

Chapter 5, *Securing Systems with Users, Groups, and Permissions*, covers how to manage users, groups, and permissions in any Linux system, with some specifics on Red Hat Enterprise Linux.

Chapter 6, *Enabling Network Connectivity*, goes through the steps to connect a system to the network and the possible ways it can be configured.

Chapter 7, *Adding, Patching, and Managing Software*, reviews the steps to add, remove, and update can be managed in our system, including examples for upgrades and rollbacks.

Chapter 8, *Administering Systems Remotely*, covers how to remotely connect to your system in order to be more effective. It includes using `ssh` connections to create keys and using the terminal multiplexer (`tmux`).

Chapter 9, *Securing Network Connectivity with firewalld*, instructs you on how the network firewall configuration works in RHEL and how to properly manage it, including managing zones, services, and ports.

Chapter 10, *Keeping Your System Hardened with SELinux*, covers the usage and basic troubleshooting of SELinux.

Chapter 11, *System Security Profiles with OpenSCAP*, explains how to run security profiles with OpenSCAP and check compliance in RHEL with typical regulations.

Chapter 12, *Managing Local Storage and Filesystems*, covers filesystem creation, mount points, and general storage management.

Chapter 13, *Flexible Storage Management with LVM*, explains how LVM empowers more flexible storage management by being able to add disks and extend logical volumes.

Chapter 14, *Advanced Storage Management with Stratis and VDO*, introduces VDO and how it can be used in our system to deduplicate storage, as well as using Stratis to manage storage more easily.

Chapter 15, *Understanding the Boot Process*, explains how the system boots and the details that make it important.

Chapter 16, *Kernel Tuning and Managing Performance Profiles with tuned*, explains how kernel tunning works and how to use tuned for predefined profile usage.

Chapter 17, *Managing Containers with Podman, Buildah, and Skopeo*, covers containers and tools for managing and building them.

Chapter 18, *Practice Exercises – 1*, allows you to test your acquired knowledge.

Chapter 19, *Practice Exercises – 2*, provides more complex testing of your acquired knowledge.

To get the most out of this book

All software requirements will be indicated in the chapters. Note that this book assumes that you have access to a physical or virtual machine, or have access to the internet to create a cloud account, in order to perform the operations that the book will guide you through.

Software/hardware covered in the book	**Operating system requirements**
Red Hat Enterprise Linux 8	You will need to install Red Hat Enterprise Linux. It can be installed in a virtual machine on Linux, macOS, or Windows.

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book's GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration`. If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots and diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781800569829_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Mount the downloaded `RHEL8.iso` disk image file as another disk in your system."

A block of code is set as follows:

```
#!/bin/bash
echo "Hello world"
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
branch = main
repo = myrepo
username = bender
protocol = https
```

Any command-line input or output is written as follows:

```
$ mkdir scripts
$ cd scripts
```

Bold: Indicates a new term, an important word, or words that you see on screen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "Select **System info** from the **Administration** panel."

> **Tips or important notes**
> Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at `customercare@packtpub.com` and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read *Red Hat Enterprise Linux 8 Administration*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Section 1: Systems Administration – Software, User, Network, and Services Management

Deploying and configuring systems and keeping them up to date is the base task that every system administrator performs in their day-to-day work. In this section, the core parts of doing so are explored in a restructured way so that you can follow the tasks one by one and learn, practice, and understand them properly.

The following chapters are included in this section:

- *Chapter 1, Installing RHEL8*
- *Chapter 2, RHEL8 Advanced Installation Options*
- *Chapter 3, Basic Commands and Simple Shell Scripts*
- *Chapter 4, Tools for Regular Operations*
- *Chapter 5, Securing Systems with Users, Groups, and Permissions*
- *Chapter 6, Enabling Network Connectivity*
- *Chapter 7, Adding, Patching, and Managing Software*

1
Installing RHEL8

The first step to start working with **Red Hat Enterprise Linux**, or **RHEL**, is to have it running. Whether in your own laptop as the main system, in a virtual machine, or in a physical server, its installation is necessary in order to get your hands on the system you want to learn to use. It is highly encouraged that you get yourself a physical or virtual machine to use the system while reading this book.

In this chapter, you will deploy your own RHEL8 system, so as to be able to follow all the examples mentioned in this book as well as discover more about Linux.

The topics to be covered in this chapter are as follows:

- Obtaining RHEL software and a subscription
- Installing RHEL8

Technical requirements

The best way to get started is by having an **RHEL8** virtual machine to work with. You may do it in your main computer as a virtual machine or using a physical machine. In the following section of this chapter, we will review both options and you will be able to run your own RHEL8 system.

> **Tip**
> A virtual machine is a way to emulate a complete computer. To be able to create this emulated computer on your own laptop, in case you are using macOS or Windows, you will need to install virtualization software such as Virtual Box, for example. If you are already running Linux, it is already prepared for virtualization, and you will only need to add the `virt-manager` package.

Obtaining RHEL software and a subscription

To be able to deploy RHEL, you will need a **Red Hat Subscription** to obtain the images to be used, as well as access to repositories with software and updates. You can obtain, free of charge, a **Developer Subscription** from the developers' portal site of Red Hat using the following link: `developers.redhat.com`. You then need to follow these steps:

1. Log in or create an account at `developers.redhat.com`.
2. Go to the `developers.redhat.com` page and click on the **Log In** button:

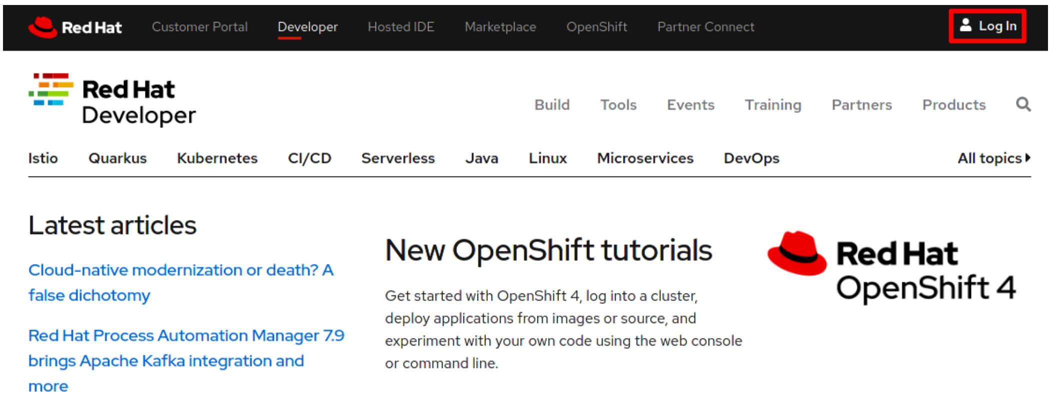


Figure 1.1 – The developers.redhat.com home page, indicating where to click to log in

3. Once in the login page, use your account or, if you do not have one, create it by clicking on **Register** in the top-right corner or on the **Create one now.** button directly in the registration box, as follows:

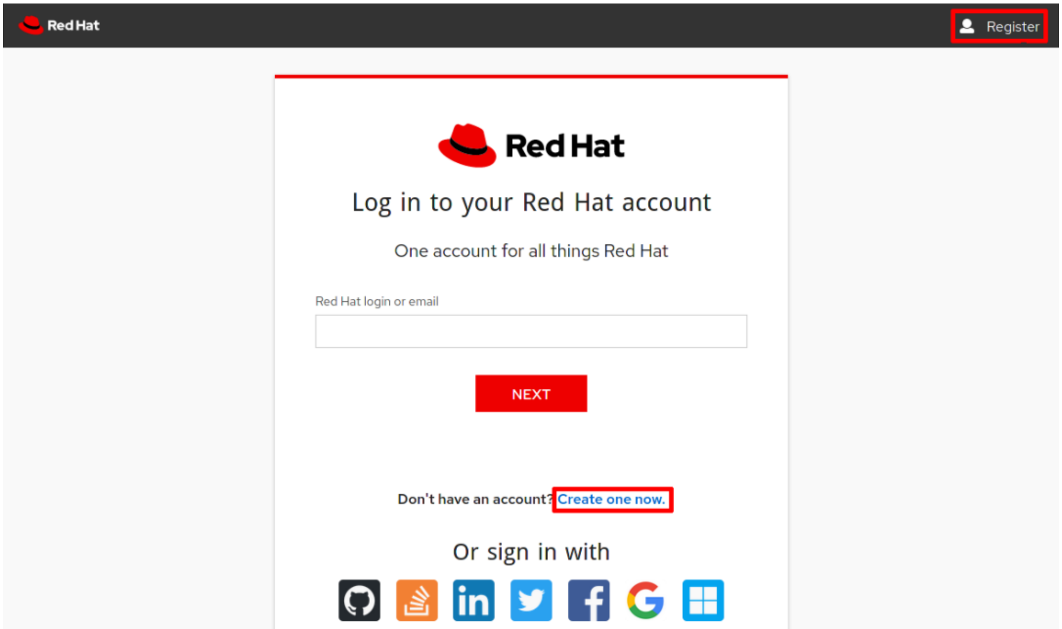


Figure 1.2 – Red Hat login page (common to all Red Hat resources)

You can choose to use your credentials in several services (in other words, *Google*, *GitHub*, or *Twitter*) if you prefer to do so.

4. Once you have logged in, go to the **Linux** section

   You can find the **Linux** section in the navigation bar before the content:

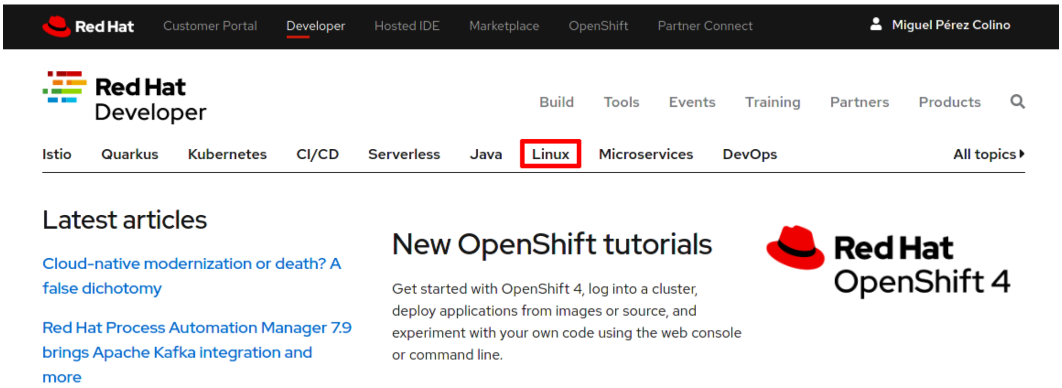


Figure 1.3 – Accessing the Linux page at developers.redhat.com

Click on **Download RHEL** which appears as a fancy button on the next page:

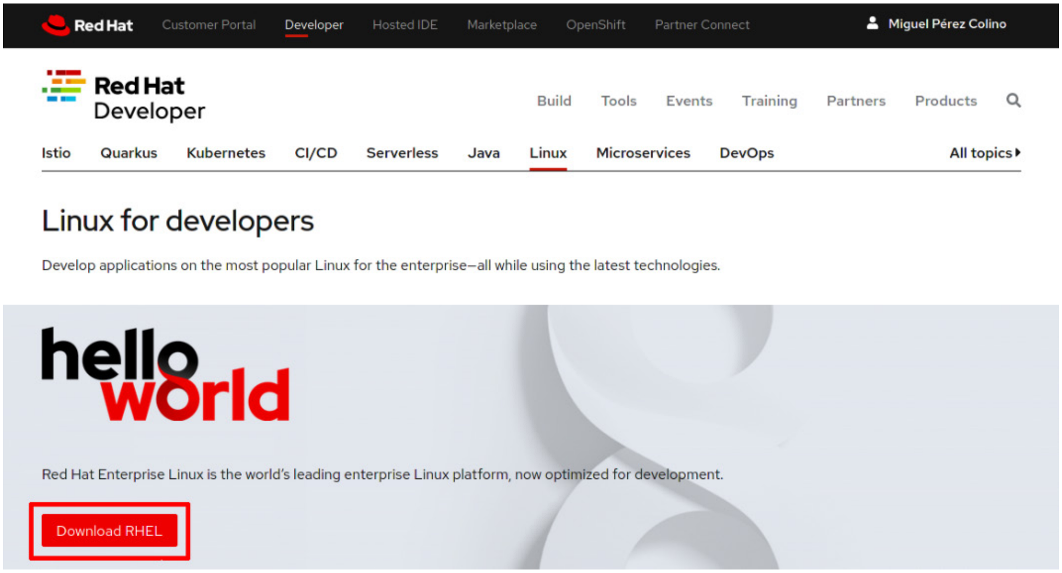


Figure 1.4 – Accessing the RHEL downloads page at developers.redhat.com

Then select the ISO image for the **x86_64 (9 GB)** architecture (which is the one used in Intel- and AMD-based computers):

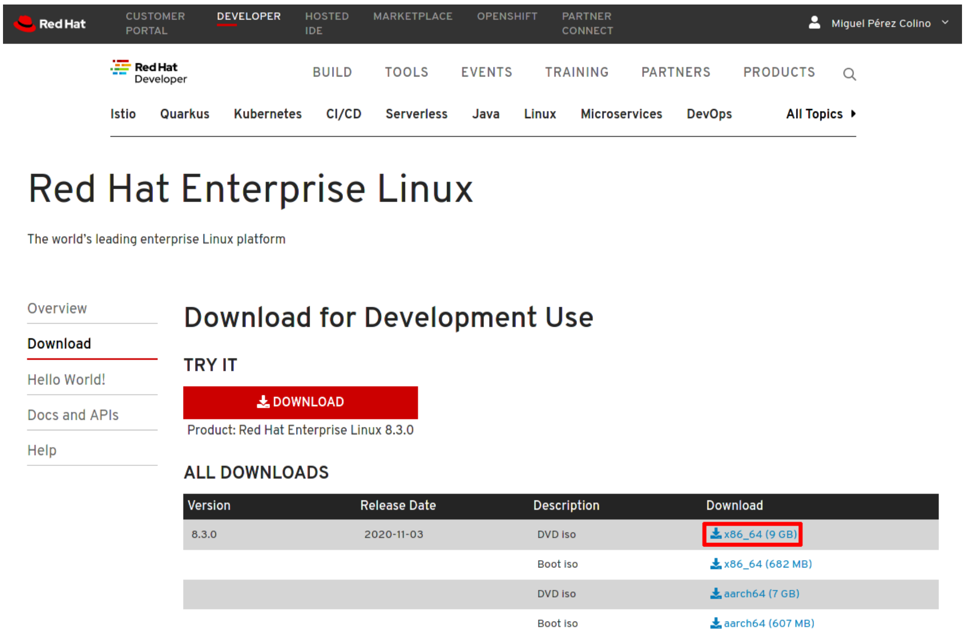


Figure 1.5 – Choosing the ISO download of RHEL8 for x86_64

5. Obtain the **RHEL8 ISO** image as follows:

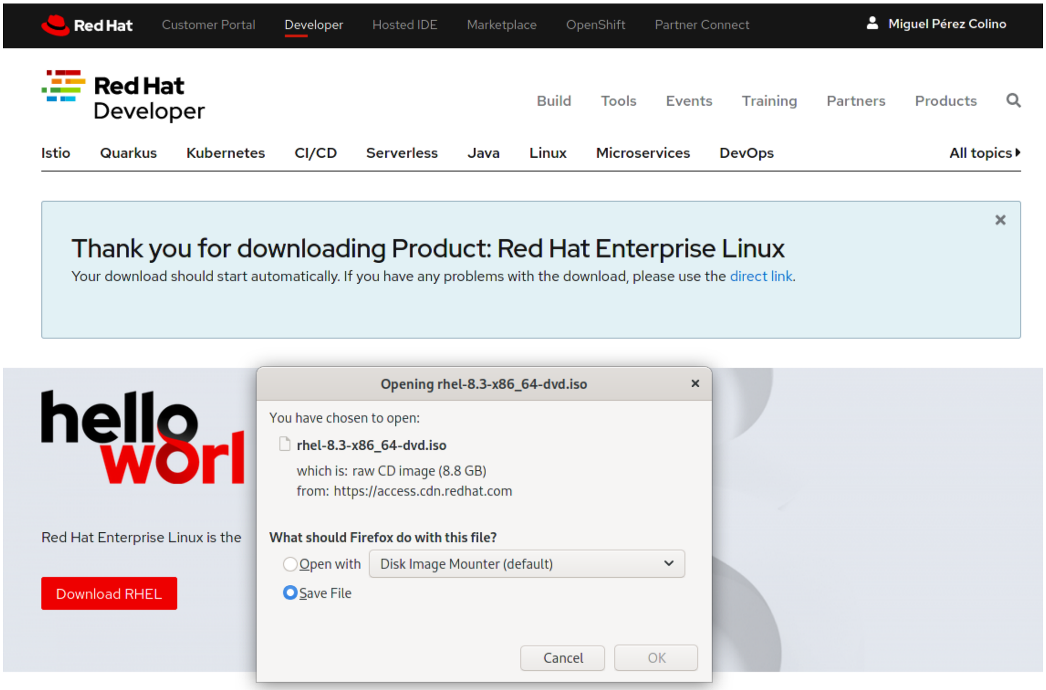


Figure 1.6 – Download dialog for RHEL8 for x86_64

The ISO image is a file that contains an exact copy of the contents of a full DVD (even when we are not using a DVD). This file will later be used to install our machines, whether dumping it to a USB drive for *Bare Metal* installations, unpacking it for network installations, or attaching it for virtual machine installations (or using out-of-band capabilities in servers such as IPMI, iLO, or iDRAC)

> **Tip**
>
> To verify the ISO image, and ensure that the one we have obtained is not corrupted, or altered, a mechanism called "checksum" can be used. Checksums are a way to review a file and provide a set of letters and numbers that can be used to verify that the file is precisely the same one as in the origin. Red Hat provides a list of `sha256` checksums for doing so in the downloads section of the Customer Portal (`https://access.redhat.com/`). An article describing the process is available here: `https://access.redhat.com/solutions/8367`.

We have the software, in this case the ISO image, to install RHEL8 in any computer. These are the same bits that are used in production machines worldwide and that you can use yourself for learning purposes with your developer subscription. Now it is time to give them a go in the next section.

Installing RHEL8

For this section of the chapter, we will follow the typical installation process to have RHEL installed on a machine. We will follow the default steps, reviewing the options available for each one.

Preparation for a physical server installation

A physical server requires some initial setup before beginning with installation. Common steps include configuring the disks in the *internal array*, connecting it to the networks, preparing the switches for any *interface aggregation* that is expected (teaming, bonding), preparing access to external *disk arrays* (in other words, *fiber channel arrays*), setting up out-of-band capabilities, and securing the **BIOS** configuration.

We will not get into the details of these preparations, except for the boot sequence. The server will require to boot (start loading the system) from an external device such as a *USB thumb drive* or *optical disk* (whether physical or emulated through the out-of-band capabilities).

To create a bootable USB thumb drive from a machine with Linux or macOS, this is as simple as doing a "disk dump" with the `dd` application. Perform the following steps:

1. Find your USB device in the system, usually `/dev/sdb` in Linux, or `/dev/disk2` in macOS (in macOS, this command requires special privileges; please run it as `sudo dmesg | grep removable`):

```
$ dmesg | grep removable
```

```
[66931.429805] sd 0:0:0:0: [sdb] Attached SCSI removable
disk
```

> **Important note**
>
> Please verify the disk name very carefully, as the procedure for using "disk dump" will completely overwrite the disk target.

Check whether the USB is mounted and, if so, dismount it (for macOS users, please use `diskutil list` to ascertain whether the device is mounted):

```
$ lsblk /dev/sdb
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb       8:0    1  3,8G  0 disk
├─sdb1    8:1    1  1,8G  0 part /run/media/miguel/USB
├─sdb2    8:2    1 10,9M  0 part
└─sdb3    8:3    1 22,9M  0 part
```

In this case, only partition 1 of the `sdb` disk, referred to as `sdb1`, is mounted. We will need to *unmount* all the partitions mounted. In this example, this is straightforward as there is only one. To do so, we can run the following command:

> **Important note**
>
> Using **super-user do** or **sudo**, for administrative tasks, such as unmounting devices, we could open an administrator shell (`root` in Linux and Unix-like systems) or run the command using `sudo`, which provides administrative privileges to the current user. When running commands with `sudo`, the user will be requested to enter their password (not the admin password, but the user's own password) to proceed with the execution (this default behavior may be overridden in the `sudoers` configuration file).

```
$ sudo umount /dev/sdb1
```

Dump the image! (Warning, this will erase the selected disk!):

```
$ sudo dd if=rhel-8.3-x86_64-dvd.iso of=/dev/sdb bs=512k
```

> **Tip**
>
> Alternative methods are available for creating a boot device. Alternative graphical tools are available for creating a boot device that can help select both the image and the target device. In Fedora Linux (the community distribution where RHEL was based on, and a workstation for many engineers and developers), the **Fedora Media Writer** tool can be used. For other environments, the **UNetbootin** tool could also serve to create your boot media.

Now, with the USB thumb drive, we can install any physical machine, from a tiny laptop to a huge server. The next part involves making the physical machine boot from the **USB thumb drive**. The mechanism for doing that will depend on the server being used. However, it is becoming common to offer an option to select a boot device during bootup. The following is an example of how to select a temporary boot device in a laptop:

1. Interrupt the normal startup. In this case, the boot process shows that I can do that by pressing *Enter*:

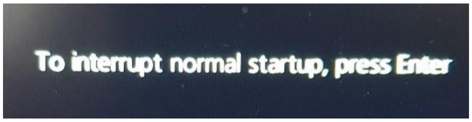


Figure 1.7 – Example of a BIOS message to interrupt normal startup

2. Choose a temporary start up device, in this case by pressing the *F12* key:

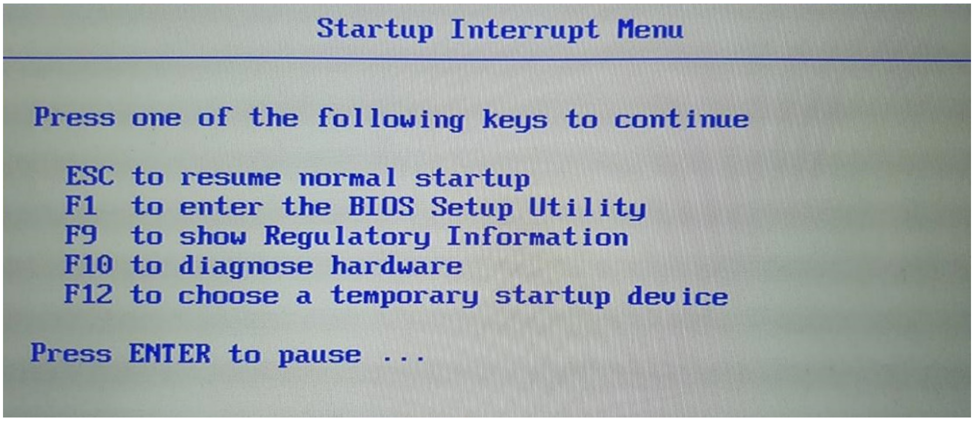


Figure 1.8 – Example of a BIOS menu for interrupted startup

3. Select the device to boot from. We want to boot from our USB thumb drive, which, in this case, is **USB HDD: ChipsBnk Flash Disk**:

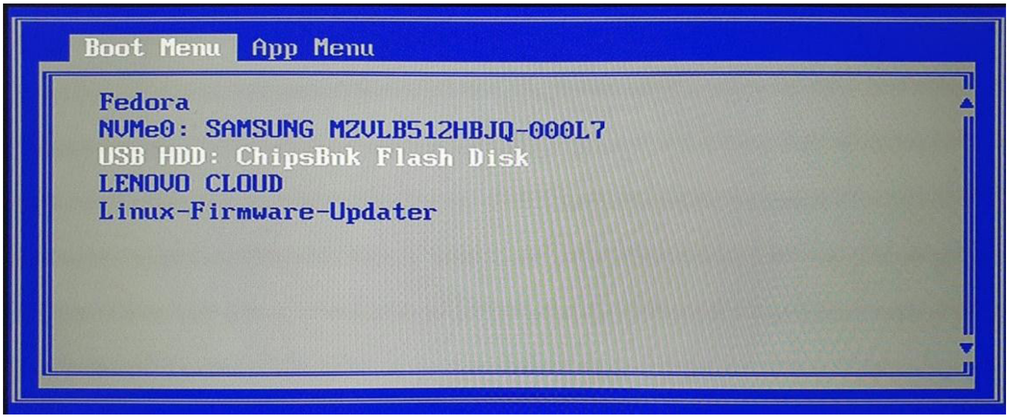


Figure 1.9 – Example of a BIOS menu to choose the USB HDD boot device

Let the system start the installer from the USB drive.

Once we know how to prepare a USB drive with an RHEL installer, and how to make a physical machine to boot from it, we can skip to the *Running an RHEL installation* section in this chapter and proceed to install it. This can be pretty useful if we have a mini server (in other words, an Intel NUC), an old computer, or a laptop to be used as the machine for following along with this book.

Next, we will look at how to prepare a virtual machine in your installation, in case you are considering following this book with your current main laptop (or workstation) but you still want to keep a separate machine to work with.

Preparation for a virtual server installation

A **virtual server** works like having some virtualization software that emulates a real machine in your current system. In a Linux workstation, installing `virt-manager` will add all the under-the-hood components required to run (for your information, these components are **KVM**, **Libvirt**, **Qemu**, and **virsh**, among others). Other no-cost virtualization software, recommended for Windows or macOS systems, includes **Oracle VirtualBox** and **VMware Workstation Player**.

The examples in this section will be executed using `virt-manager`, but are easily applicable to any other virtualization software, whether in a laptop or in the largest deployments.

The preliminary steps have been described above and require obtaining the **Red Hat Enterprise Linux ISO** image, which, in this case, will be `rhel-8.3-x86_64-dvd.iso`. Once downloaded and, if possible, having checked its integrity (as mentioned in the last tip of the *Obtaining RHEL software and a subscription* section), let's prepare to deploy a virtual machine:

1. Start your virtualization software, in this case, `virt-manager`:

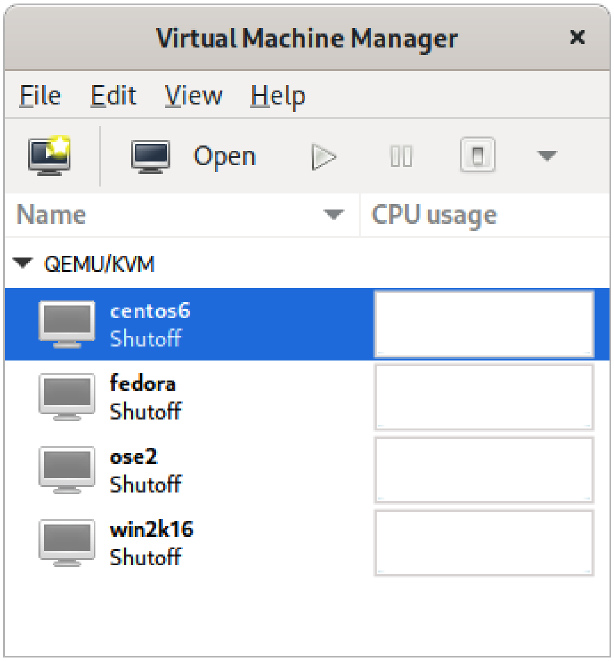


Figure 1.10 – The virtual manager main menu

2. Create a new virtual machine by going to **File** and then clicking on **New Virtual Machine**. Select **Local install media (ISO Image or CDROM)**:

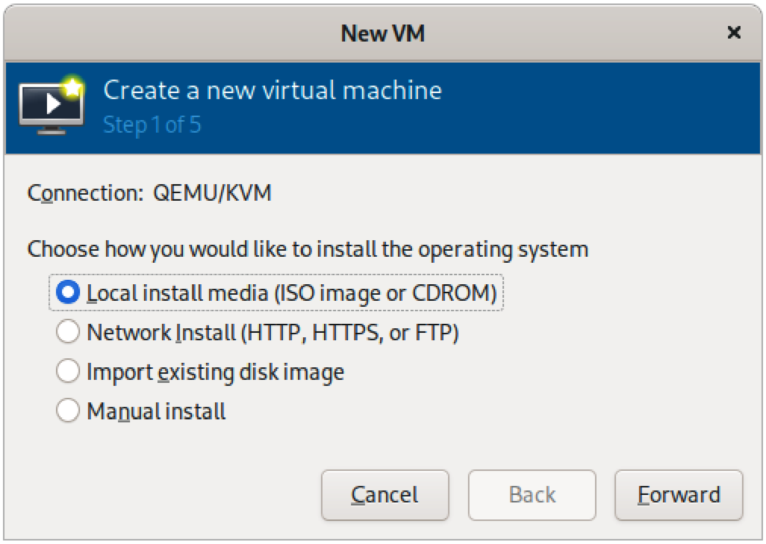


Figure 1.11 – Virtual manager – New VM menu

3. Select the *ISO image*. With this, the virtual machine will be configured with a **virtual DVD/CDROM drive** and already prepared to boot from it. This is customary behavior. However, when using a different virtualization software, you may want to perform a check:

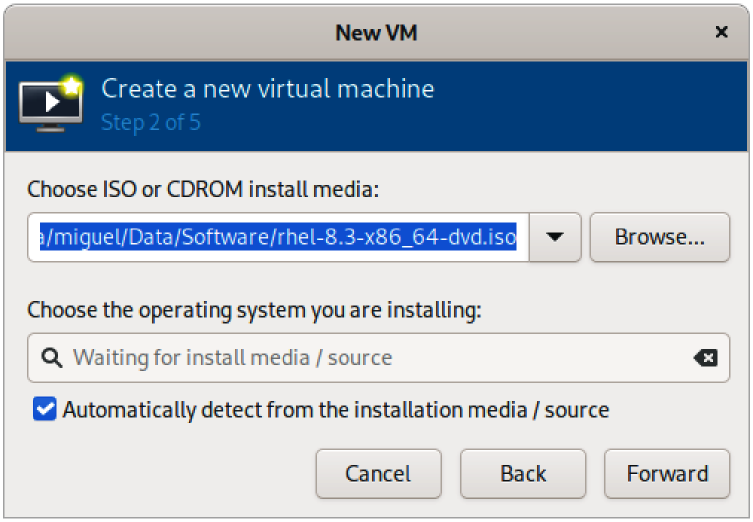


Figure 1.12 – The virtual manager menu to select an ISO image as an installation medium

4. Assign memory and CPU to the virtual machine we are creating (note: a virtual machine is usually referred to as a **VM**). For **Red Hat Enterprise Linux 8** (also referred to as **RHEL8**), 1.5 GB of memory is the minimum, while 1.5 GB per logical CPU is recommended. We will use the minimum settings (1.5 GB memory, 1 CPU core):

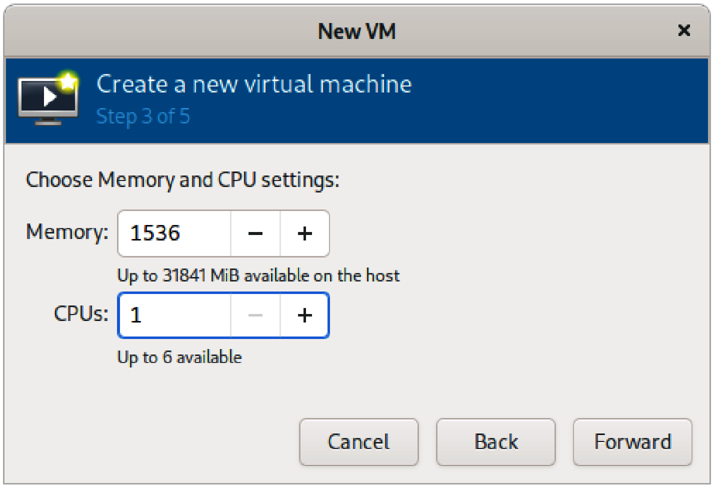


Figure 1.13 – The virtual manager menu for selecting memory and CPU

It is time to assign at least one disk to the virtual machine. In this case, we will assign a single disk with the minimum disk space, 10 GB, but in future chapters, we will be able to assign more disks to test other functionalities:

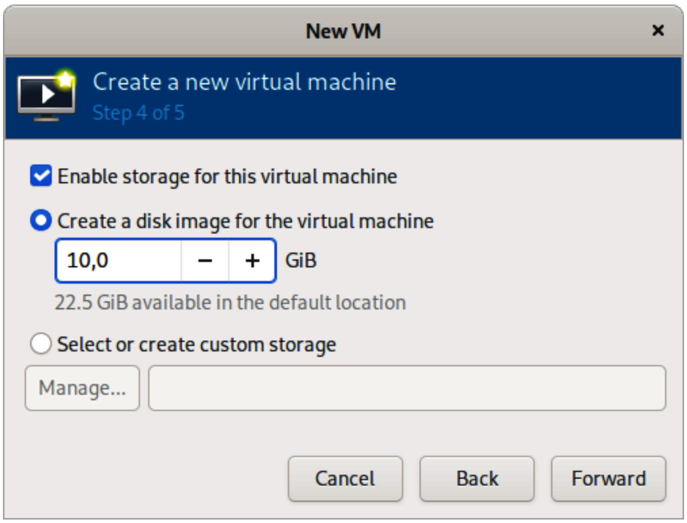


Figure 1.14 – The virtual manager menu to create a new disk and add it to the virtual machine

5. Our virtual machine has all that we need to get started: A boot device, memory, CPU, and disk space. In this last step, a network interface is added, so now we even have a network. Let's review the data and launch it:

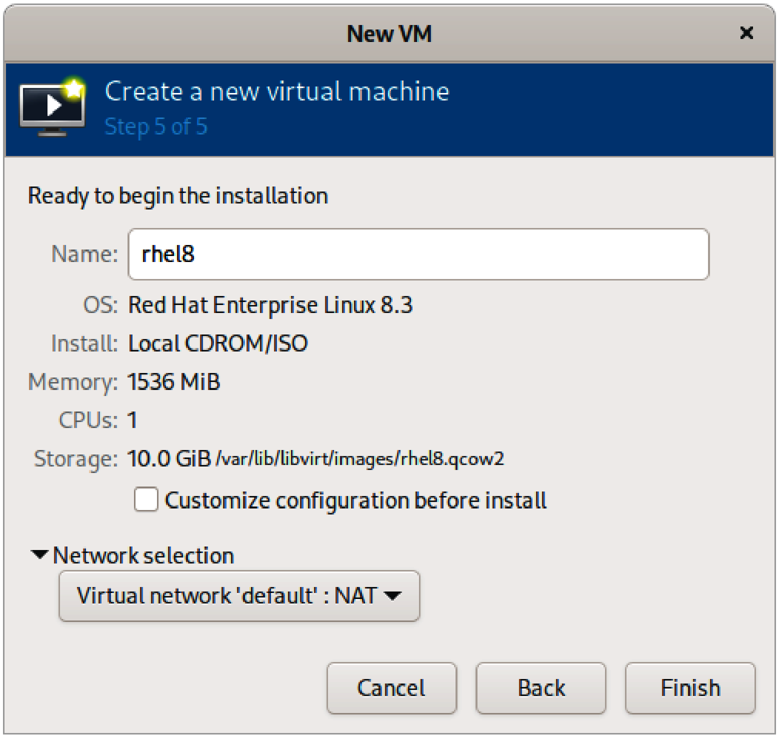


Figure 1.15 – The virtual manager menu for selecting the name of the virtual machine and the network

After taking these steps, we have a fully functional virtual machine available. Now it is time to complete the process by installing the RHEL operating system on it. Check how to do this in the next section.

Running an RHEL installation

Once we have prepared our virtual or physical server for installation, it's time to proceed with it. We will know whether all the previous steps were performed correctly if we arrive at the following screen:

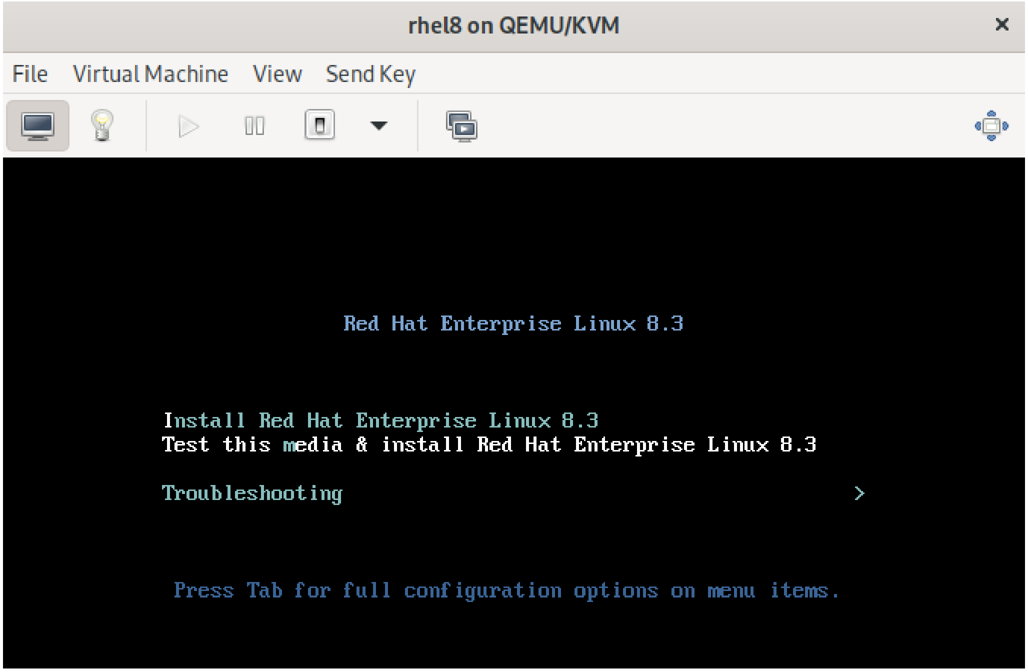


Figure 1.16 – Initial boot screen for RHEL8 installation with Install selected

We are offered three options (*selected in white*):

- **Install Red Hat Enterprise Linux 8.3**: This option will boot and run the installer.
- **Test this media & install Red Hat Enterprise Linux 8.3**: This option will check the image being used to ensure that it is not corrupt, and that the installation can proceed with certainty. It is recommended to use this one for the first time using a just downloaded ISO image or just created media, such as a USB thumb drive or DVD (in a virtual machine, it takes approximately 1 minute to run the check).

- **Troubleshooting**: This option will help you review other options in case there are problems with installation, with a running system, or with hardware. Let's take a quick look at the available options on this menu:
  - **Install Red Hat Enterprise Linux 8.3 in basic graphics mode**: This option is useful for systems with an old graphics card and/or an unsupported one. It can help to get the system installed in case an issue with visualization is identified.
  - **Rescue a Red Hat Enterprise Linux system**: This option can be used when we have a system with issues booting or when we want to access it to introspect it (in other words, review a possible compromised system). It will initiate a basic in-memory system to perform these tasks.
  - **Run a memory test**: The system memory can be checked to prevent issues, as in the case of a brand-new server, for instance, where we want to ensure that its memory is running correctly, or a system suffering issues and panics that may indicate a memory-related issue.
  - **Boot from local drive**: In case you booted from the install media, but you already have a system installed.
  - **Return to main menu**: To go back to the previous menu.

> **Important note**
>
> The RHEL boot menu will show several options. The one selected will show in white, with one single letter in a different color, in this case, "i" for install and "m" for test media. These are shortcuts. Pressing the key with that letter will take us directly to this menu item.

Let's proceed with **Test this media & install Red Hat Enterprise Linux 8.3** to let the installer review the ISO image we are using:

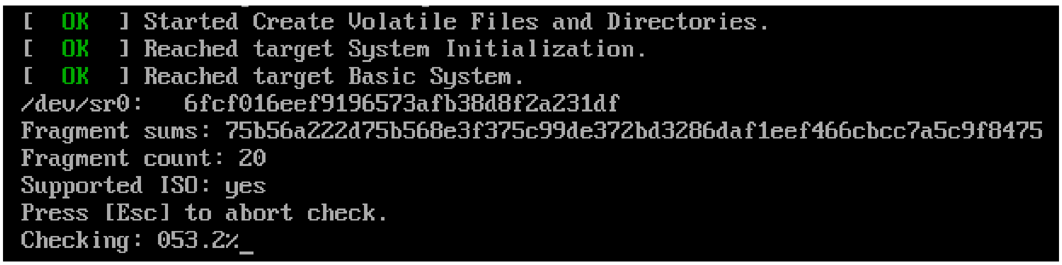


Figure 1.17 – RHEL8 ISO image self-check

Once completed, it will reach the first installation screen. The installer is called **Anaconda** (a joke, as it is written in a language called **Python**, and it follows a step-by-step approach). It is important to pay attention to the options we will select during installation, as we will review them later in the *Automating deployments with Anaconda* section of the book.

Localization

The first step to installation is selecting the installation language. For this installation, we will select **English**, followed by **English (United States)**:

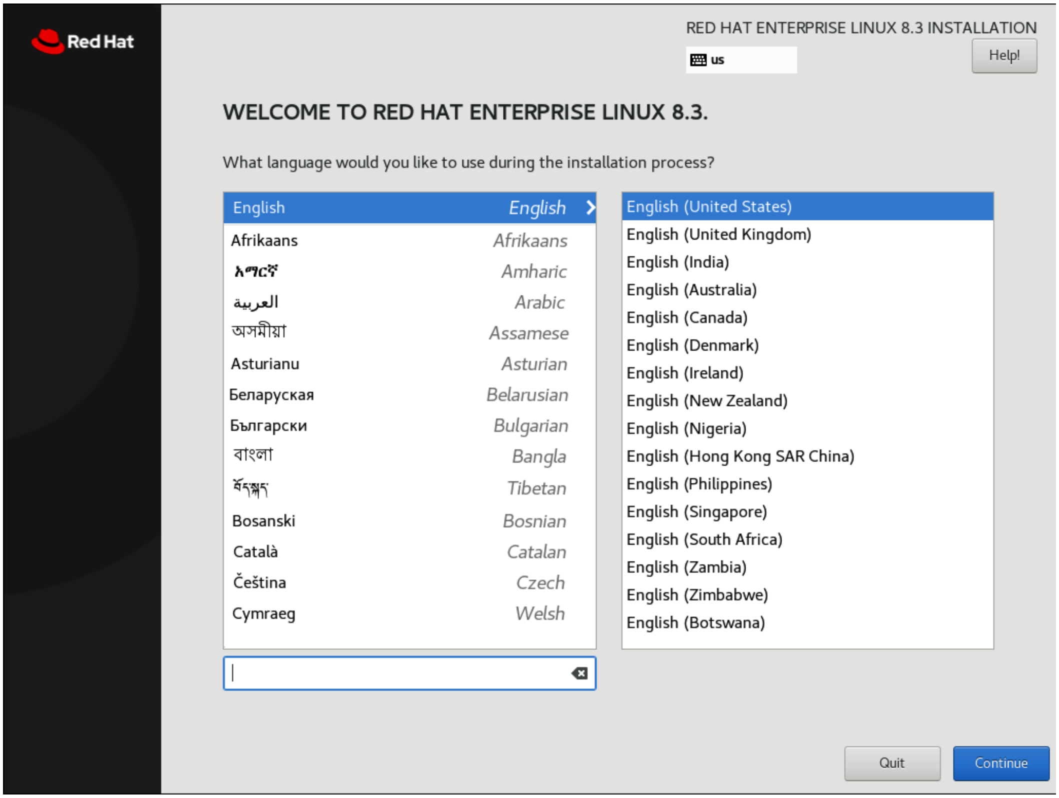


Figure 1.18 – RHEL8 install menu – Language

In case you cannot easily find your language, you may type it in the box under the list to search for it. Once a language is selected, we can click the **Continue** button to proceed. This will take us to the **INSTALLATION SUMMARY** screen:

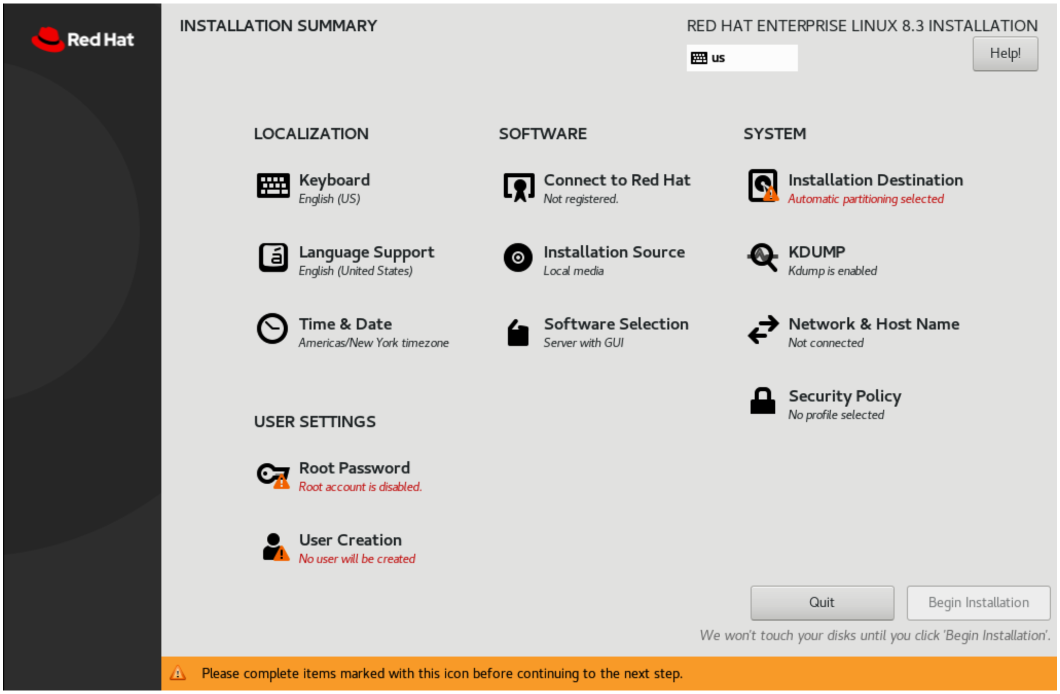


Figure 1.19 – RHEL8 install menu – Main page

On the **INSTALLATION SUMMARY** screen, all the configuration parts required are shown, with many of them (the ones without a warning sign and red text underneath) already preconfigured with defaults.

Let's review the **LOCALIZATION** settings. First, **Keyboard**:

Figure 1.20 – RHEL8 install – The Keyboard selection icon

We can review the keyboard settings, which can help, not just changing the keyboard, but adding extra layouts in case we want to switch between them:

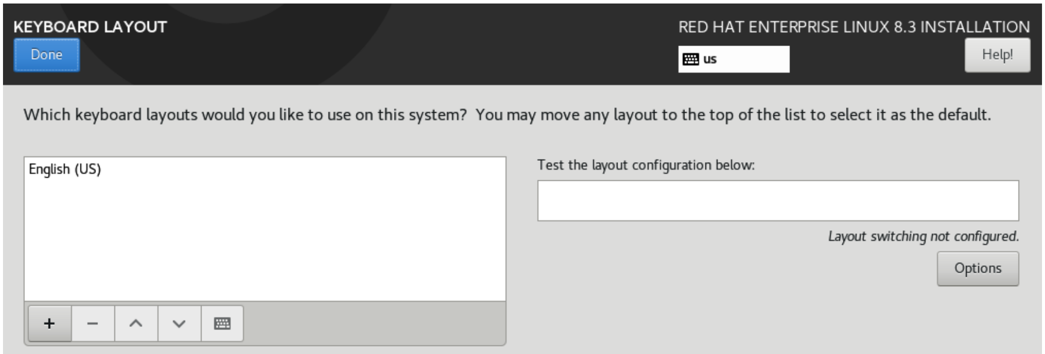


Figure 1.21 – RHEL8 install – Keyboard selection dialog

This can be done by clicking on the + button. Here is an example of adding the **Spanish; Castilian (Spanish)** layout. We search for `spa` until it appears, and then we select it and then click **Add**, as follows:

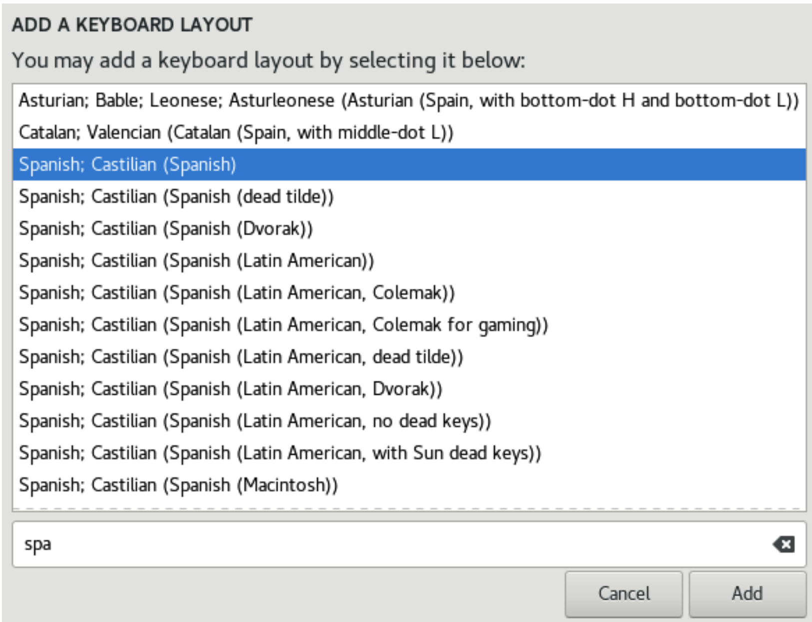


Figure 1.22 – RHEL8 install – Keyboard selection list

To make it the default option will require clicking on the ^ button underneath. In this case, we will keep it as a secondary option so that the supporting software gets installed. Once completed, click **Done**:

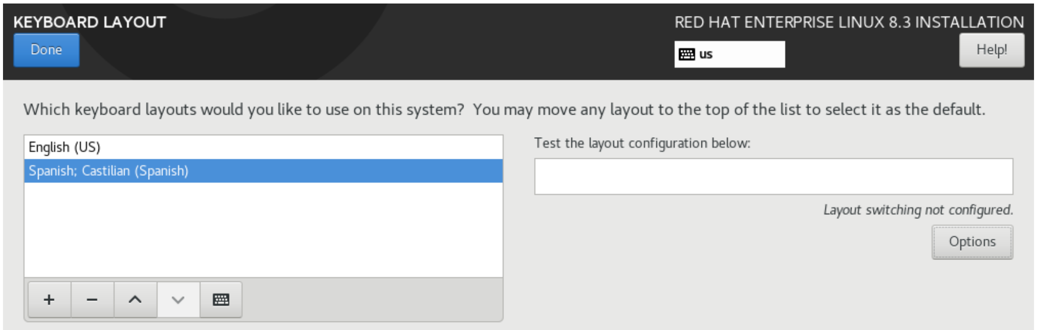


Figure 1.23 – RHEL8 install – Keyboard selection dialog with different keyboards

Now, we will move on to **Language Support**:

Figure 1.24 – RHEL8 install – Language selection icon

Here, we can also add our local language. In this example, I'll use **Español**, and then **Español (España)**. This will again include the software required to support the language that has been added:

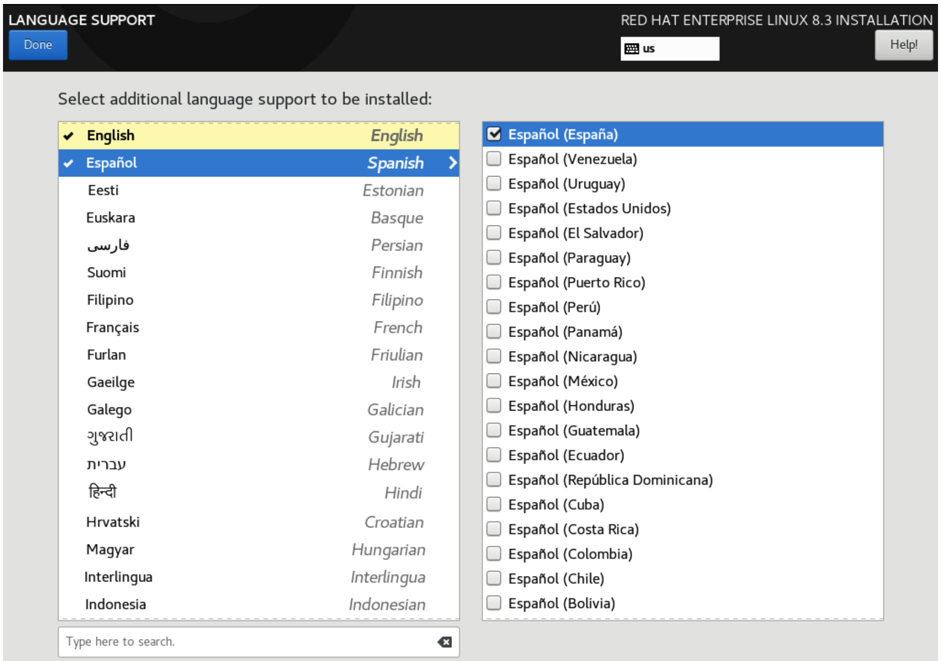


Figure 1.25 – RHEL8 install – Language selection dialog with different languages

We will proceed with both languages configured, although you may want to choose your own localized language.

Now, we will move on to **Time & Date**, which can be seen as follows:

Figure 1.26 – RHEL8 install – Time and Date selection icon

The default configuration is set to the city of New York in the United States of America. You have two possibilities here:

- Use your local time zone. This is recommended when you want to have all the logs registered in that time zone (in other words, because you are only working in one time zone or because there are local teams for each time zone). In this example, we are selecting the **Spain, Madrid, Europe** time zone:

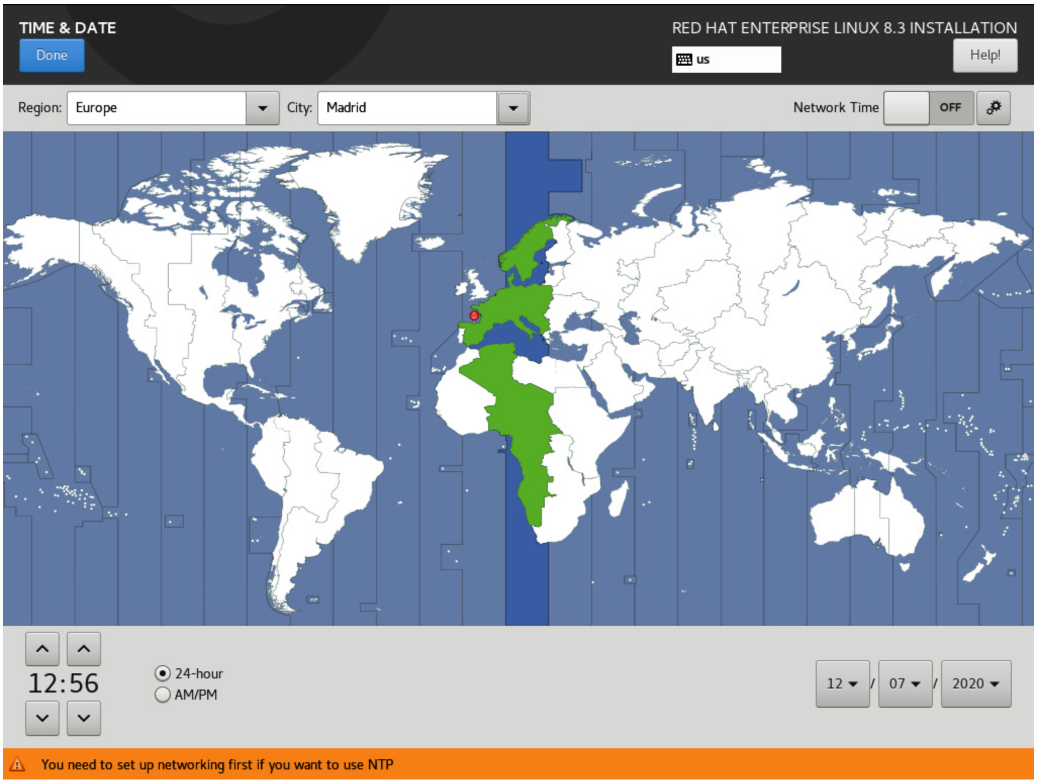


Figure 1.27 – RHEL8 install – Time and Date selection dialog – Madrid selected

- Use **Coordinated Universal Time** (also known as **UTC**) to have the same time zone for all the servers around the globe. This can be selected under **Region:** | **Etc**, and then **City:** | **Coordinated Universal Time**:

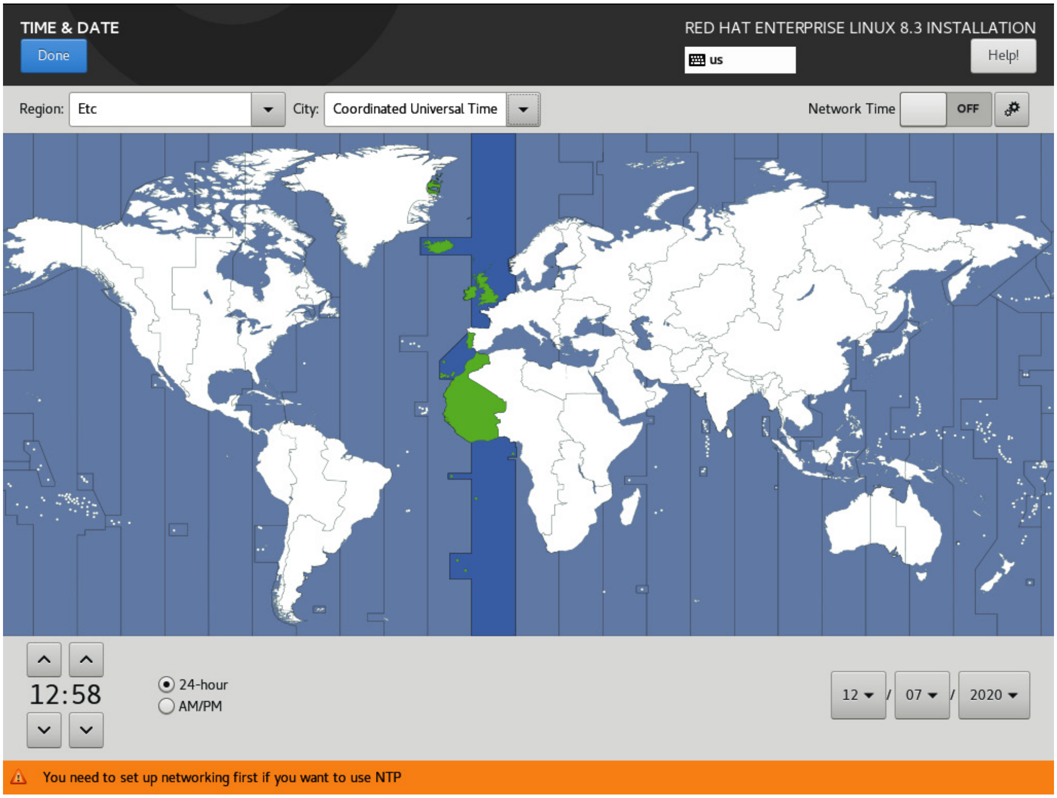


Figure 1.28 – RHEL8 install – Time and Date selection dialog – UTC selected

We will proceed with the localized time for Spain, Madrid, Europe, although you may want to select your localized time zone.

> **Tip**
>
> As you can see in the screen, there is an option to select **Network Time** to have the machine's clock synchronized with other machines. This option can only be selected once the network is configured.

Software

With the **Localization** configuration completed (or almost completed; we may come back for the network time later), we move on to the **Software** section, or, more precisely, to **Connect to Red Hat** under it:

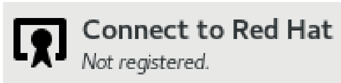


Figure 1.29 – RHEL8 install – Connect to Red Hat selection icon

In this section, we can use our own Red Hat account, like the one we created previously under `developers.redhat.com`, to access the latest updates for the system. To configure it, we will need to configure the network first.

For the purposes of this deployment, we will not configure this section now. We will review how to manage subscriptions and get updates in *Chapter 7*, *Adding, Patching, and Managing Software*, of this book.

> **Important note**
>
> Systems management with Red Hat Satellite: For large deployments with more than 100 servers, Red Hat offers "Red Hat Satellite," with advanced software management capabilities (such as versioned content views, centralized security scans with OpenSCAP, and simplified patching and updating for RHEL).
> To connect to a Red Hat Satellite, the activation key can be used, thereby simplifying the management of systems.

Let's now move on to **Installation Source**, as follows:

Figure 1.30 – RHEL8 install – Installation Source icon

This can be used for installation using remote sources. It is very useful when using the boot ISO image that only contains the installer. In this case, as we are using the full ISO image, it already contains all the software (also referred to as *packages*) needed to complete the installation.

The next step is **Software Selection**, as shown in the following screenshot:

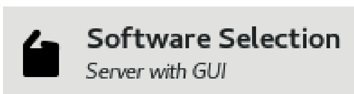


Figure 1.31 – RHEL8 install – Software Selection icon

In this step, we can select a predefined set of packages to be installed on the system so that it can perform different tasks. While it can be very convenient to do so in this stage, we are going to adopt a more manual approach and select the **Minimal Install** profile to add software to the system later.

This approach also has the advantage of reducing the **attack surface** by installing just the minimum required packages in the system:

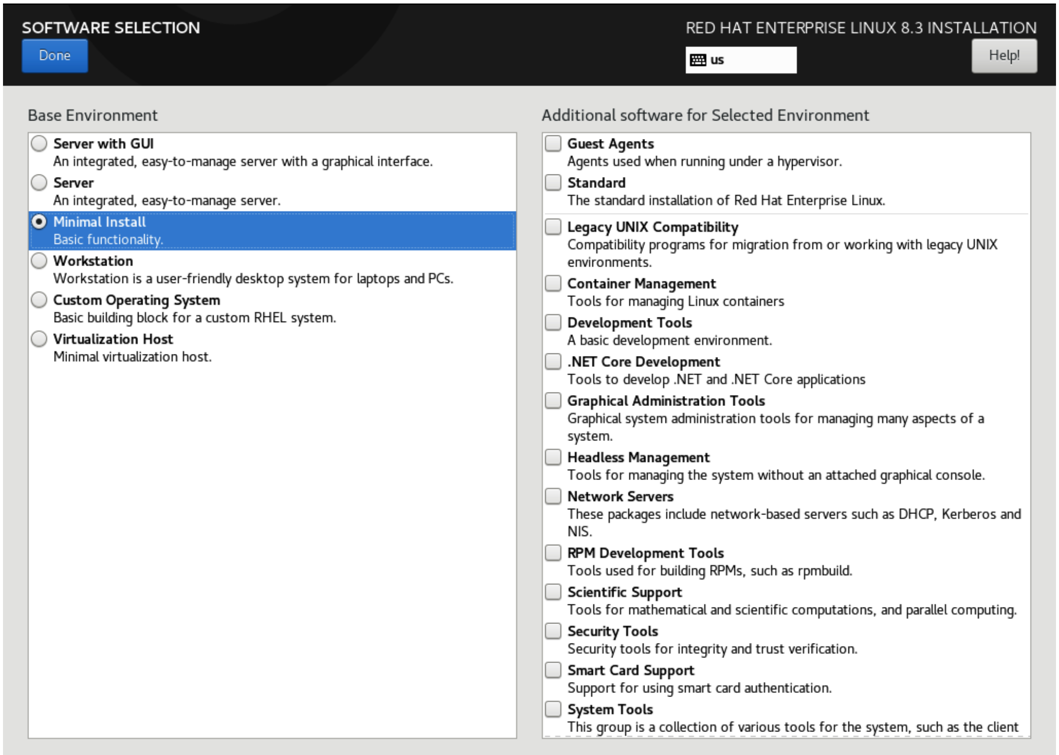


Figure 1.32 – RHEL8 install – Software Selection menu; Minimal Install selected

System

Once the set of packages has been selected, let's move on to the **System** configuration section. We will start with the destination of the installation, where we can choose the disk or disks to be used to install and configure them:

Figure 1.33 – RHEL8 install – Installation Destination icon with a warning sign as this step is not complete

This task is very important as it will define not just the way the system is deployed on the disk, but also how the disk is distributed and with which tools. Even when in this section, we won't use the advanced options. We will take some time to review the main options.

This is the default **Device Selection** screen, with only one local standard disk discovered, no **Specialized & Network Disks** options, and ready to run the **Automatic** partitioning. This can be seen in the following screenshot:

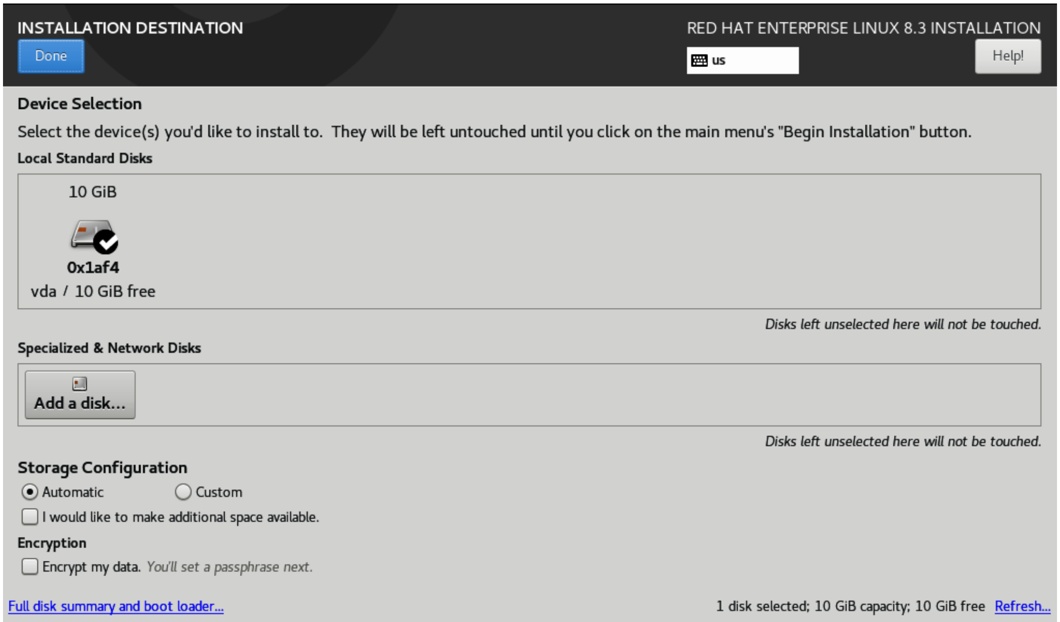


Figure 1.34 – RHEL8 install – INSTALLATION DESTINATION menu, with automatic partitioning selected

Clicking **Done** in this section will complete the minimal set of data required to continue with the installation.

Let's review the sections.

Local Standard Disks are a set of disks to be used by the installer. It may be the case that we have several disks, and we only want to use a specific disk:

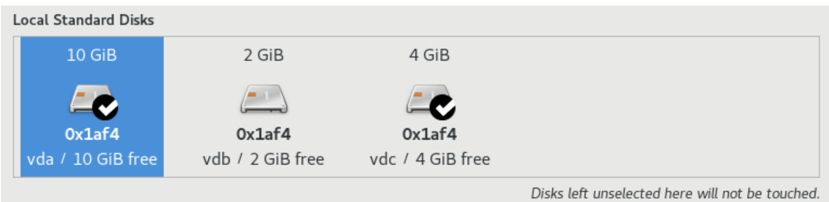


Figure 1.35 – RHEL8 install – INSTALLATION DESTINATION menu, with several local disks selected

This is an example of having three available disks and using only the first and third ones.

In our case, we only have one disk, and it is already selected:

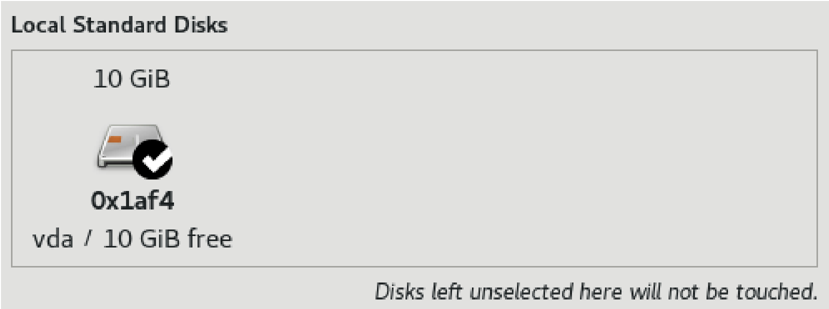


Figure 1.36 – RHEL8 install – INSTALLATION DESTINATION menu, with a single local disk selected

It would be easy to use full disk encryption by selecting **Encrypt my data**, which is highly recommended for laptop installations or for installing in environments with low levels of trust:

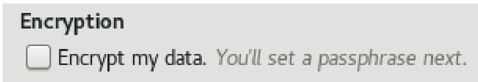


Figure 1.37 – RHEL8 install – INSTALLATION DESTINATION menu, with the data encryption option (not selected)

For this example, we will not encrypt our drive.

The **Automatic** install option will distribute the disk space automatically:

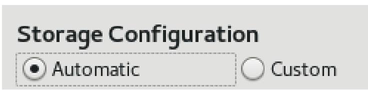


Figure 1.38 – RHEL8 install – INSTALLATION DESTINATION menu; Storage Configuration (Automatic)

It will do so by creating the following resources:

- `/boot`: Space to allocate the system core (`kernel`) and files to help during the boot process (such as the initial boot image, `initrd`).
- `/boot/efi`: Space to support the EFI boot process.
- `/"`: The root filesystem. This is the main storage space where the system lives. Other disks/partitions will be assigned to folders (when doing so, they will be called `mountpoints`).
- `/home`: Space where the user will store personal files.

Let's select this option and then click **Done**.

> **Tip**
> System partitions and the boot process: Do not worry if you still do not fully understand some extended concepts regarding system partitions and boot processes. To cover the filesystems, partitions, and how to manage disk space, there is a chapter entitled *Managing Local Storage and Filesystems* dedicated to it. To review the boot process, there is a chapter entitled *Understanding the Boot Process*, which reviews step by step the full system start up sequence.

The next step involves reviewing **Kdump**, or **Kernel Dump**. This is a mechanism that allows the system to save the status in case a critical event happens and it crashes (it dumps the memory, hence its name):

Figure 1.39 – RHEL8 install – Kdump configuration icon

In order to work, it will reserve some memory for itself where it will stay, waiting to act if the system crashes. The default configuration does a good calculation of the requirements:

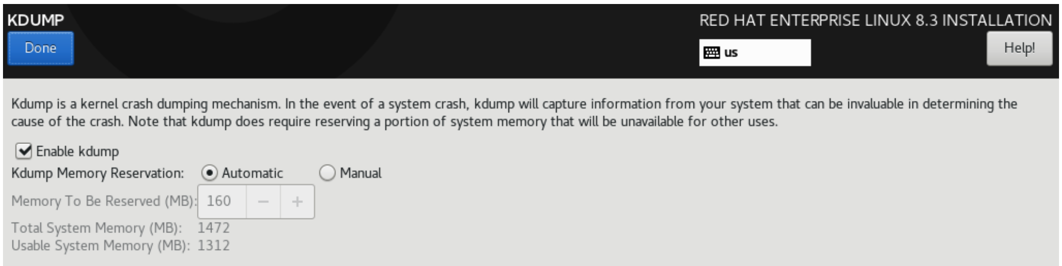


Figure 1.40 – RHEL8 install – Kdump configuration menu

Clicking **Done** will take us to the next step, **Network & Host Name**, which appears as follows:

Figure 1.41 – RHEL8 install – Network & Host Name configuration icon

This section will help to have the system connected to a network. In the case of a virtual machine, access to external networks will be handled by the **Virtualization Software**. It is very common that the default configuration uses **Network Address Translation** (**NAT**) and **Dynamic Host Configuration Protocol** (**DHCP**), which will provide a network configuration to the virtual machine and access to external networks.

Once on the configuration page, we can see how many network interfaces are assigned to our machine. In this case, there is only one, as follows:

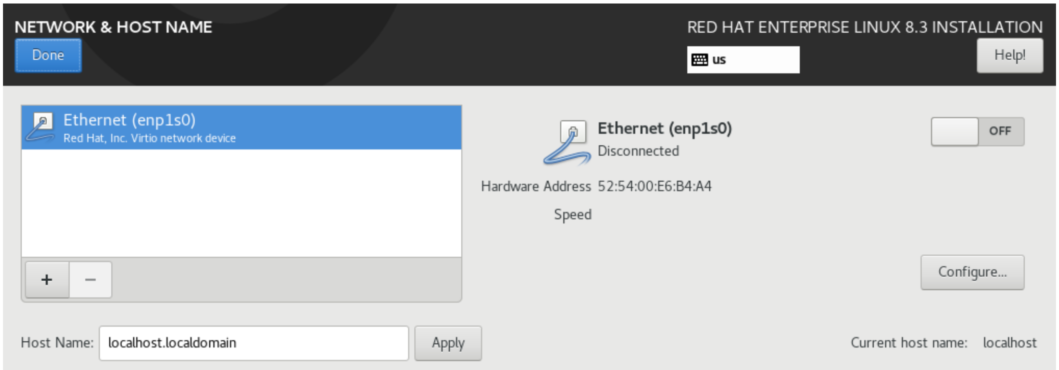


Figure 1.42 – RHEL8 install – NETWORK & HOST NAME configuration menu

First, we can enable the interface by clicking on the **ON/OFF** toggle on the right. To turn it off, it looks like this:

Figure 1.43 – RHEL8 install – NETWORK & HOST NAME configuration toggle (OFF)

And to turn it on, it should look like this:

Figure 1.44 – RHEL8 install – NETWORK & HOST NAME configuration toggle (ON)

We will see that the interface now has a configuration (**IP Address**, **Default Route**, and **DNS**):

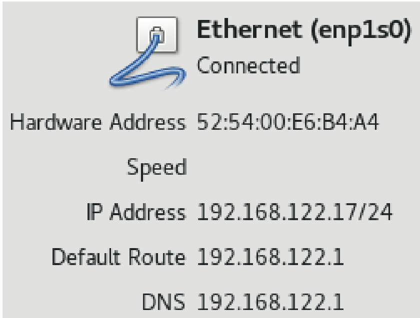


Figure 1.45 – RHEL8 install – NETWORK & HOST NAME configuration information details

To make this change permanent, we will click the **Configure** button at the bottom-right corner of the screen to edit the interface configuration:

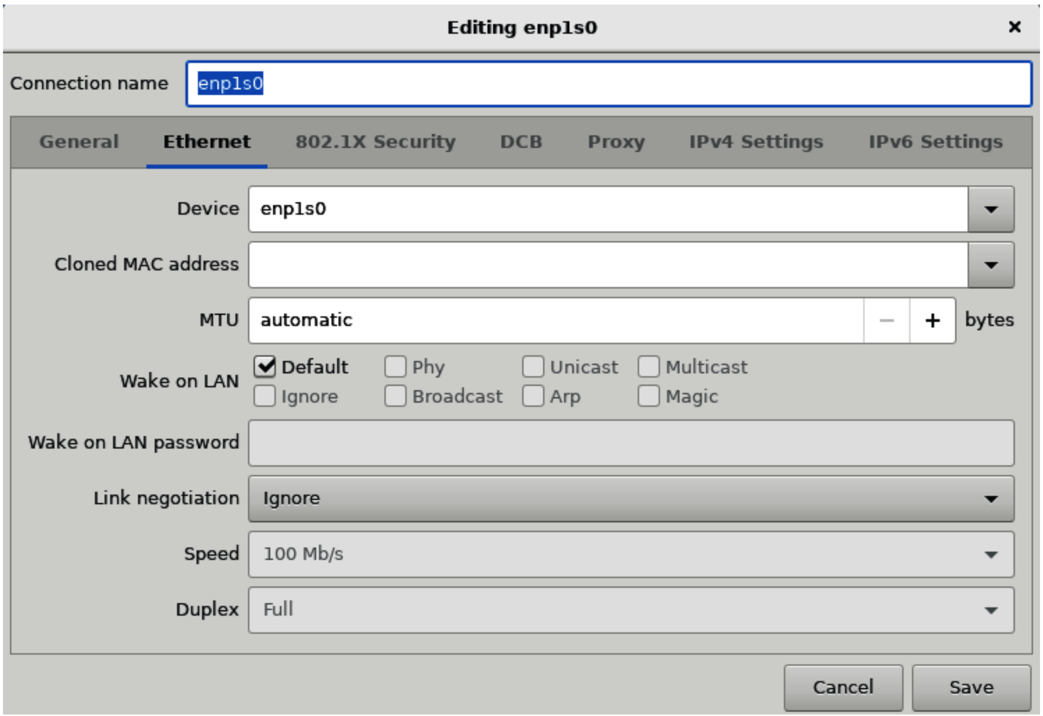


Figure 1.46 – RHEL8 install – NETWORK & HOST NAME configuration; interface configuration; Ethernet tab

Clicking on the **General** tab will present the main options. We will select **Connect automatically with priority** and leave the value as **0**, just like this:

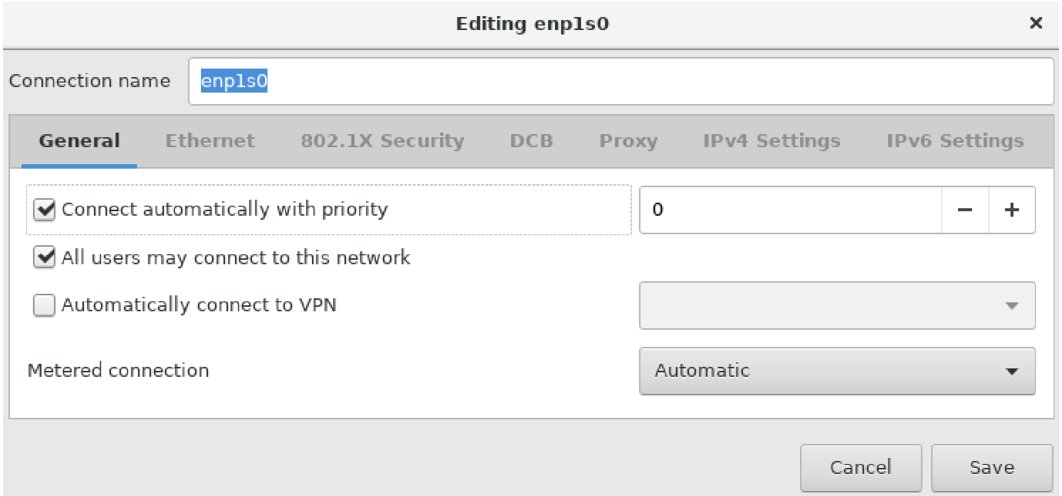


Figure 1.47 – RHEL8 install – NETWORK & HOST NAME configuration; interface configuration; General tab

Clicking **Save** will make the changes permanent and have this network interface enabled by default.

Now it's time to give a name to our virtual server. We will go to the **Host Name** section in the main page and type the name we want for it. We can use `rhel8.example.com`, and then click **Apply**:

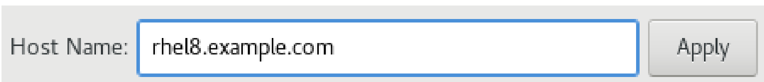


Figure 1.48 – RHEL8 install – NETWORK & HOST NAME configuration; Host Name detail

> **Tip**
>
> The domain `example.com` is used for demonstration purposes, and it is safe to be used on any occasion, knowing that it will not collide or cause any trouble to other systems or domains.

The networking page will look like this:

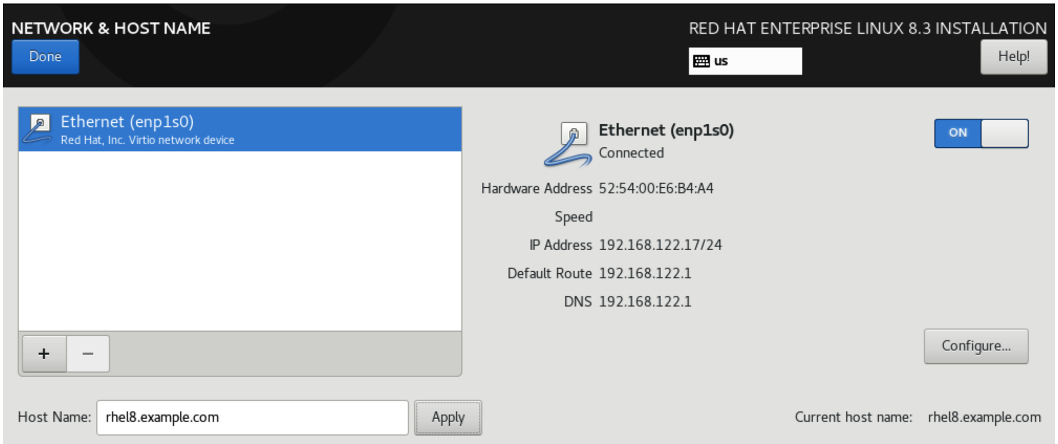


Figure 1.49 – RHEL8 install – NETWORK & HOST NAME configuration menu; configuration complete

Clicking **Done** will take us back to the main installer page, with a system connected to a network and prepared to connect once the installation is complete.

The chapter entitled *Enabling Network Connectivity* will describe in more detail the options available to configure the network in an RHEL system.

> **Important note**
> Now that the system is connected to the network, we can go back to **Time & Date** and enable network time (which is done automatically by the installer), as well as go to **Connect to Red Hat** to subscribe the system to Red Hat's **Content Distribution Network** (or **CDN**). The subscription of the system to the CDN will be explained in detail in *Chapter 7, Adding, Patching, and Managing Software*.

It is now time to review the final system option, security profiles, by going to **Security Policy** as follows:

Figure 1.50 – RHEL8 install – Security Policy configuration icon

In it, we will see a list of security profiles that can be enabled by default in our system:

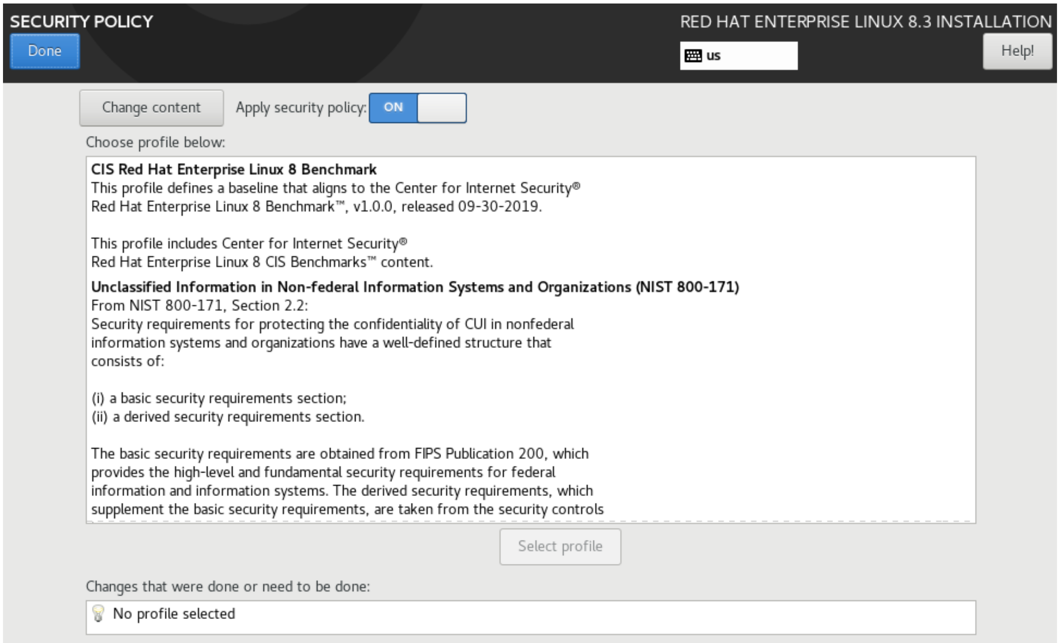


Figure 1.51 – RHEL8 install – SECURITY POLICY configuration menu

The security profiles have requirements that we are not covering in this installation (such as having separate `/var` or `/tmp` partitions). We can click on **Apply security policy** to turn it off, and then on **Done**:

Figure 1.52 – RHEL8 install – Security policy configuration toggle (off)

More on this topic will be covered in *Chapter 11, System Security Profiles with OpenSCAP*.

User settings

The main administrator user in a Unix or Linux system is called `root`.

We can enable a root user by clicking in the **Root Password** section, although this is not necessary and, in security restricted environments, you are advised not to do so. We will do so in this chapter in order to learn how to do it and explain the cases covered:

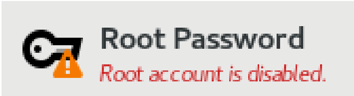


Figure 1.53 – RHEL8 install – Root Password configuration icon (warning as it is not set)

After clicking on **Root Password**, we are presented with a dialog to type it:

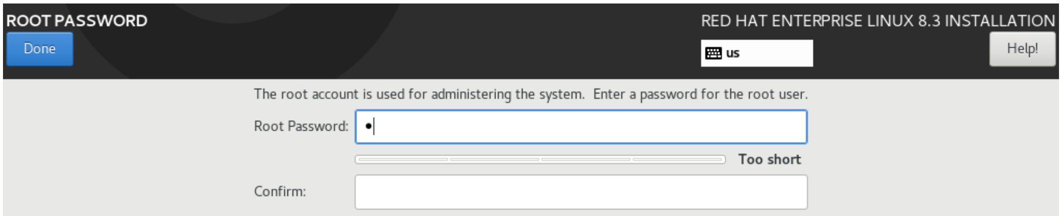


Figure 1.54 – RHEL8 install – Root Password configuration menu

It is recommended that the password has the following:

- More than 10 characters (and a minimum of 6)
- Lowercase and uppercase
- Numbers
- Special characters (such as $, @, %, and &)

If the password does not meet those requirements, it will warn us and it will force us to click **Done** twice to use a weak password.

It is now time to create a user for the system by clicking on **User Creation**:

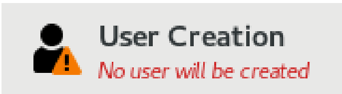


Figure 1.55 – RHEL8 install – User Creation configuration icon (warning as it is not complete)

This will take us to a section to input user data:

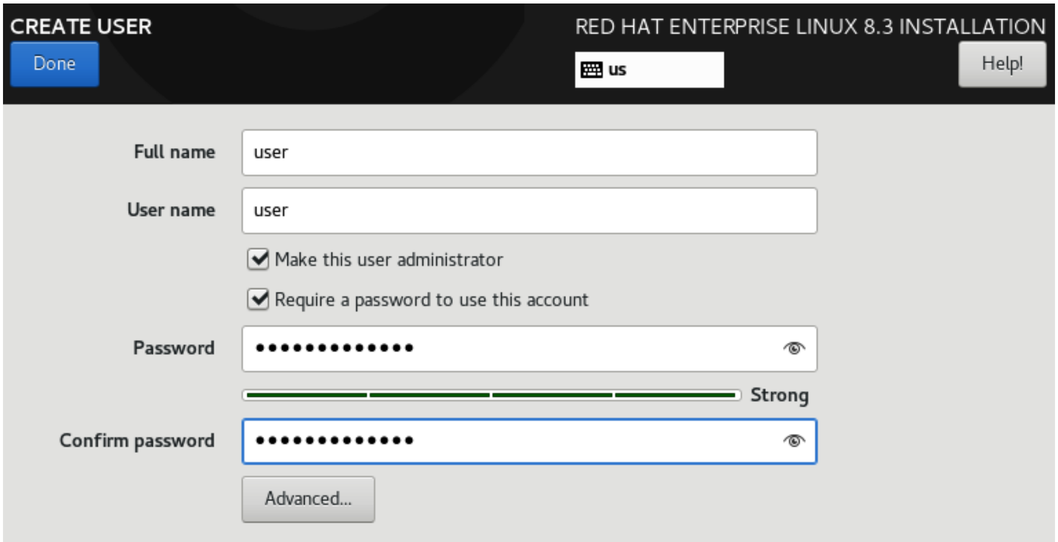


Figure 1.56 – RHEL8 install – User Creation configuration menu

The same password rules will apply here as in the previous section.

Clicking on **Make this user administrator** will enable the performance of administrative tasks (and also no need to configure the `root` password).

> **Tip**
> As a good practice, do not use the same password for the root account and for the user account.

The *Chapter 5, Securing Systems with Users, Groups, and Permissions* includes a section on how to use and manage administrative privileges for users with the `sudo` tool.

Click on **Done** to return to the main installer screen. The installer is ready to proceed with the installation. The main page will look like this:

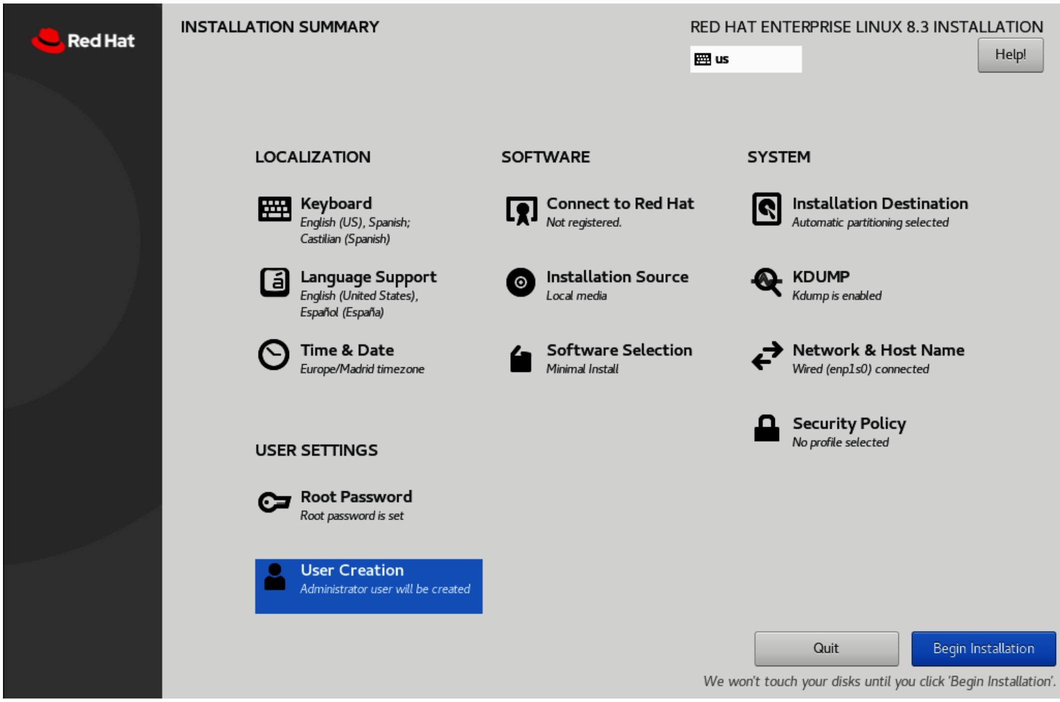


Figure 1.57 – RHEL8 install – Main menu once completed

Clicking on **Begin Installation** will launch the installation process:

> **Important note**
>
> If any of the steps required to start the installation are omitted, the **Begin Installation** button will be grayed out, and therefore not available to be clicked.

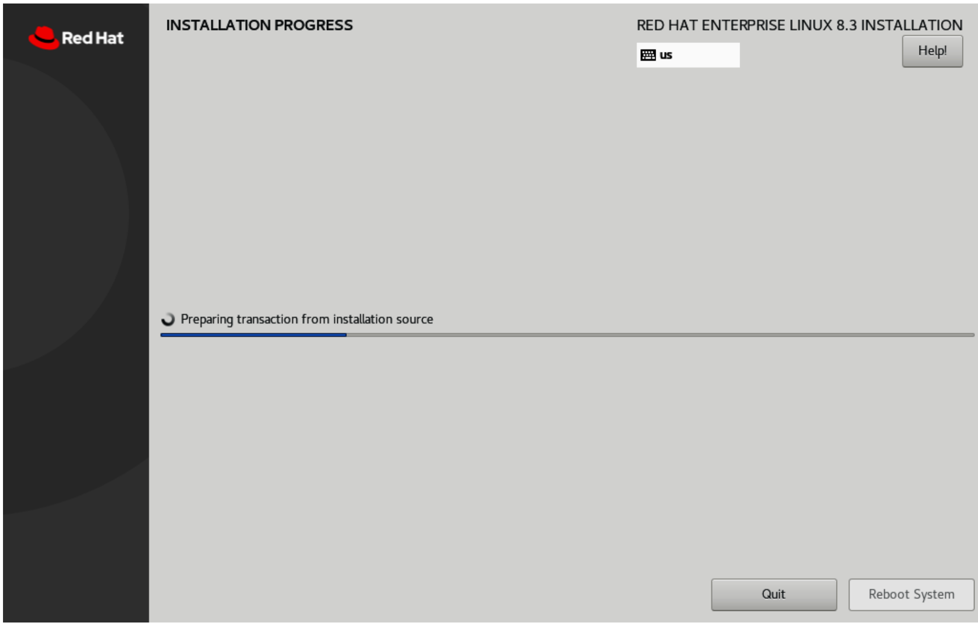


Figure 1.58 – RHEL8 install – Installation in progress

Once the installation is complete, we can click on **Reboot System** and it will be ready to use:

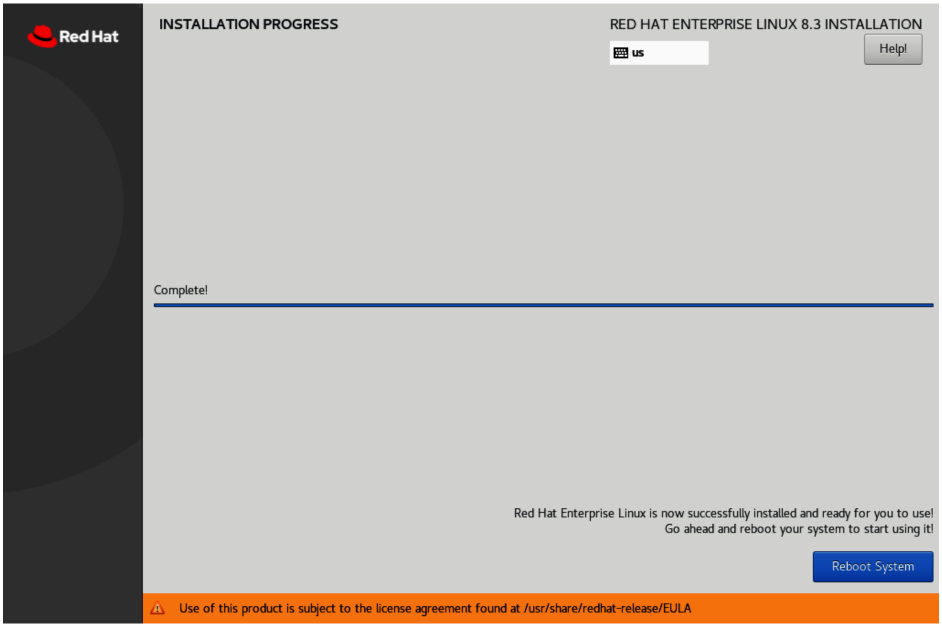


Figure 1.59 – RHEL8 install – Installation complete

It is important to remember to detach the ISO image from the virtual machine (or remove the USB thumb drive from the server) and check that the boot order is properly configured in the system.

Your first Red Hat Enterprise Linux 8 system is now ready! Congratulations.

As you can see, it is easy to install RHEL in a virtual or physical machine and have it ready to be used for any service we want to run in it. In the cloud, the process is very different as machines are instantiated from images to run. In the next chapter, we will review how to run RHEL in a virtual machine instance in the cloud.

Summary

The *Red Hat Certified System Administrator* exam is entirely practical, based on real-world experience. The best way to prepare for it is by practicing as much as possible, which is why this book begins by providing access to *Red Hat Enterprise Linux 8* (RHEL8) and offering alternatives on how to deploy your own virtual machine.

Different scenarios are covered regarding installation. These are the most common ones, and include using a physical machine, a virtual machine, or a cloud instance. In this chapter, we focused on using a virtual machine or a physical one.

When using physical hardware, we will be focusing on the fact that many people like to reuse old hardware, buy second-hand or cheap mini servers, or even use their laptop as the primary installation for their Linux experience.

In the case of virtual machines, we are thinking about the people that want to keep all their work on the same laptop, but without messing with their current operating system (which may not even be Linux). This could also work well with the previous option by having virtual machines on your own mini server.

After this chapter, you are ready to proceed with the rest of the book, having at least an instance or Red Hat Enterprise Linux 8 available to work with and practice on.

In the next chapter, we will review a number of advanced options, such as using the cloud for RHEL instances, automating the installation, and best practices.

Let's get started!

2
RHEL8 Advanced Installation Options

In the previous chapter, we learned how to install **Red Hat Enterprise Linux**, or **RHEL**, on a physical or virtual machine so that we use it while we're reading this book. In this chapter, we will review how to use RHEL *instances in the cloud* and the main differences that appear when doing so.

You will also learn not just how to deploy a system, but the best choices to do so, and be able to perform the deployment in an *automated fashion*.

To complete the installation, a section on *best practices* has been included so that you can start avoiding long-term issues from day one.

These are the topics that will be covered in this chapter:

- Automating RHEL deployments with Anaconda
- Deploying RHEL on the cloud
- Installation best practices

Technical requirements

In this chapter, we will review the automated installation process using **Anaconda**. For that, you will need to use the *RHEL8 deployment* we created in the previous chapter.

We will also create cloud instances, for which you will need to create an account in the cloud environment of your choice. We will be using **Google Cloud Platform**.

Automating RHEL deployments with Anaconda

Once you have finished your first deployment of RHEL locally, you can log in as root on the machine and list the files that the `root` user has in their folder:

```
[root@rhel8 ~]# ls /root/
anaconda-ks.cfg
```

You will find the `anaconda-ks.cfg` file. This is an important file, called a `kickstart`, and it contains the responses given to the installer, **Anaconda**, during the installation process. Let's review the content of this file.

> **Important note**
>
> In cloud images, there is no `anaconda-ks.cfg` file.

This file can be reused to install other systems with the same options as the ones we used for this installation. Let's review the options that we added during our previous installation.

Lines starting with # are comments and have no effect on the installation process.

The comment specifying the version that is being used is as follows:

```
#version=RHEL8
```

Then, a type of installation was performed. It can be `graphical` or `text` (for headless systems, it is common to use the second one):

```
# Use graphical install
graphical
```

The software source for installing application packages, or any other package, is specified with the `repo` entry. As we were using the ISO image, it was accessed (mounted, in Linux parlance) as if it were a *CDROM*:

```
repo --name="AppStream" --baseurl=file:///run/install/sources/mount-0000-cdrom/AppStream
```

Sections are specified with the `%` symbol. In this case, we will enter the `packages` section with the list of packages to be installed and use the `%end` special tag to close them. There are two selections: a group of packages that is defined by it starting with the `@^` symbol (in this case, `minimal-environment`) and the name of a package that doesn't require any prefix (in this case the package is `kexec-tools`, which are responsible for installing the `kdump` capability we explained previously):

```
%packages
@^minimal-environment
kexec-tools
%end
```

We continue to click options without a section. In this case, we have the keyboard layouts and system language support. As you can see, we added the *English US American keyboard* (marked as `us`) and the *Spanish*, *Spain* one (marked as `es`):

```
# Keyboard layouts
keyboard --xlayouts='us','es'
```

For the system language, we also added English US American (`en_US`) and Spanish, Spain (`es_ES`). There are several ways to manage, store, and represent text in operating systems. The most common one nowadays is `UTF-8`, which enables us to have many character sets under one single standard. That's why the system language has `.UTF-8` appended to it:

```
# System language
lang en_US.UTF-8 --addsupport=es_ES.UTF-8
```

> **Tip**
>
> **Unicode (or Universal Coded Character Set) Transformation Format – 8-bit**, or UTF-8 for short is a character encoding that extends the capabilities of previous ones in order to support Chinese, Cyrillic, or Arabic (among many others) in the same text (like the one representing a web page or a console). UTF-8 was presented in 1993 and is used by 95.9% of the world-wide web's pages. Previous character sets supported US English or Latin characters only, such as the **American Standard Code for Information Interchange**, or **ASCII**, published in 1963. To learn more about character encodings and their evolution, check out the Wikipedia pages for both UTF-8 and ASCII.

Now, it's time to configure the network interface. In this case, we only have one, named `enp1s0`. The configuration uses IPv4 with the **Dynamic Host Configuration Protocol** (**DHCP**) and IPv6, both of which are activated at boot. The hostname is configured as `rhel8.example.com`:

```
# Network information
network  --bootproto=dhcp --device=enp1s0 --ipv6=auto
--activate
network  --hostname=rhel8.example.com
```

Now, we need to define the installation media. In this case, we used an emulated CDROM/DVD using the ISO image file we downloaded:

```
# Use CDROM installation media
cdrom
```

The option for `firstboot` is enabled by default. In this case, as the installation does not include a *graphical interface*, it won't be run, but will be added to the `kickstart` file. We can safely remove it, like so:

```
# Run the Setup Agent on first boot
firstboot --enable
```

Now, let's configure the disks. First, to be safe, we will instruct the installer to ignore all the disks except for the target one; in this case, `vda`:

```
ignoredisk --only-use=vda
```

Important note

The disk's name will vary, depending on the platform you are running on. Typically, it will be `vda`, `xda`, or `sda`. In this example, we show the `vda` disk that was defined by the installer, Anaconda, as we used in the previous chapter.

Now, we must install the bootloader to enable the system to boot. We will do so in the **Master Boot Record** or **MBR** of the main disk, `vda`, and we will instruct it to use the `crashkernel` option, which enables the `kdump` mechanism (this dumps memory in case of a system crash):

```
# System bootloader configuration
bootloader --append="crashkernel=auto" --location=mbr --boot-
drive=vda
```

Now, we must partition the disk. In this case, this will be fully automated:

```
autopart
```

Space to be used by the system must be declared. We will clear the whole disk for this example:

```
# Partition clearing information
clearpart --none --initlabel
```

Let's set the time zone to Madrid, Europe:

```
# System timezone
timezone Europe/Madrid --isUtc
```

Now, we will set the root password and create a user (note that the encrypted password was redacted for security purposes):

```
# Root password
rootpw --iscrypted $xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
user --groups=wheel --name=user
--password=$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxx --iscrypted --gecos="user"
```

> **Tip**
> The generated Anaconda file from the previous chapter contains an example of the encrypted password hash. If we want to change it, a new encrypted password hash, to be included here, can be generated if we run the `python -c 'import crypt,getpass;pw=getpass.getpass();print(crypt.crypt(pw) if (pw==getpass.getpass("Confirm: ")) else exit())'` command.

Now, we need a special section where we can configure `kdump` so that we can reserve memory automatically:

```
%addon com_redhat_kdump --enable --reserve-mb='auto'
%end
```

We also need a special section specifying the password policy that will be used for installation:

```
%anaconda
pwpolicy root --minlen=6 --minquality=1 --notstrict --nochanges
--notempty
pwpolicy user --minlen=6 --minquality=1 --notstrict --nochanges
--emptyok
pwpolicy luks --minlen=6 --minquality=1 --notstrict --nochanges
--notempty
%end
```

And with this, our `kickstart` file to reinstall our system is complete.

To use it, we will need to pass the kickstart option to the installer. To do so we edit the kernel parameters. Let's see how is it done.

We start by pressing *Tab*, during boot, while the line **Install Red Hat Enterprise Linux 8.3** is selected. The boot line, starting with **vmlinuz**, will appear at the bottom of the screen:

Figure 2.1 – RHEL8 Installer – Editing the boot line

Let's remove the `quiet` option and add the one that lets the installer know where the kickstart is:

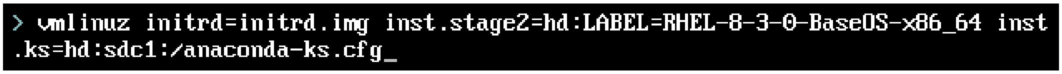


Figure 2.2 – RHEL8 Installer – Adding the kickstart option to the boot line

The option we've added is as follows:

```
inst.ks=hd:sdc1:/anaconda-ks.cfg
```

There are three parts to it that we can take a look at:

- `hd`: The kickstart will be in a disk, such as a second USB drive.
- `sdc1`: The device that hosts the file.
- `/anaconda-ks.cfg`: The path to the kickstart file in the device.

With this, we can reproduce the full installation we have done.

> **Tip**
> The *Red Hat Enterprise Linux 8 Customizing Anaconda* guide provides detailed options you can follow if you wish to create your own *Anaconda Kickstart* file or further customize this one. It can be accessed here: `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/customizing_anaconda/index`.

As you have seen, it is very easy to create a kickstart file and automate the deployment of Red Hat Enterprise Linux.

Now, let's move and look at a different way to make a RHEL 8 instance available: in the cloud.

Deploying RHEL on the cloud

Deploying Red Hat Enterprise Linux on the cloud has some differences from the previous deployments we've done. Let's look at what these differences are:

- We won't use an ISO image or Anaconda to perform a deployment, but a preconfigured image, usually prepared and made available by the cloud provider:

  - The image can be later customized and adapted to our needs.
- We will not be able to choose the configuration details of our system (such as selecting a time zone, for example) during installation time, but will be able to after.
- An automated mechanism will be in place to change settings, such as adding a user and their credentials to access the system or configure network:

  - The most extended and well-known mechanism used by cloud providers to do so is `cloud-init`.

  - Some of the images that are delivered by the cloud provider include the `cloud-init` software.

  - Systems are usually accessed remotely using the `ssh` protocol and the SSH keys that are generated by the user in the cloud provider (please check out *Chapter 8*, *Administering Systems Remotely*, for more details on how to access a system).

> **Important note**
> When it comes to creating RHEL images, it's possible to create our own for the cloud or virtualization. To do so, we can use the Red Hat Enterprise Linux image builder (`https://developers.redhat.com/blog/2019/05/08/red-hat-enterprise-linux-8-image-builder-building-custom-system-images/`). However, it is not part of RHCSA, so it will not be covered in this book. Instead, we will follow the approach of taking the default image and customizing it.

Cloud providers propose an initial getting started offer where you try their services at no cost. It's a good way to get started with RHEL and cloud services.

In this book, we'll be using Google Cloud as an example, so other clouds will not be covered. We will provide a brief example of how a Red Hat Enterprise Linux 8 instance can be created and modified in this cloud environment. To do so, we will use **Google Cloud** (it provides, as of December 2020, an initial credit that could last the whole duration required to complete this book).

To follow this chapter, you will need to complete the following steps:

1. If you do not have a Google account, you will need to create one (if you use Gmail and/or an Android phone, you will have one already).
2. Log into your Google account at `https://accounts.google.com` (or check you have already logged in). You will be required to sign in for a free trial, at which point you will have to provide a credit card number.
3. Go to `https://cloud.google.com/free` and claim your free credits.
4. Go to the cloud console at `https://console.cloud.google.com`.
5. Go to the **Projects** menu, which is shown here as **No organization** at the top bar, to show the projects for the new account:

Figure 2.3 – RHEL8 in Google Cloud – Organization menu access

6. Click on **NEW PROJECT**:

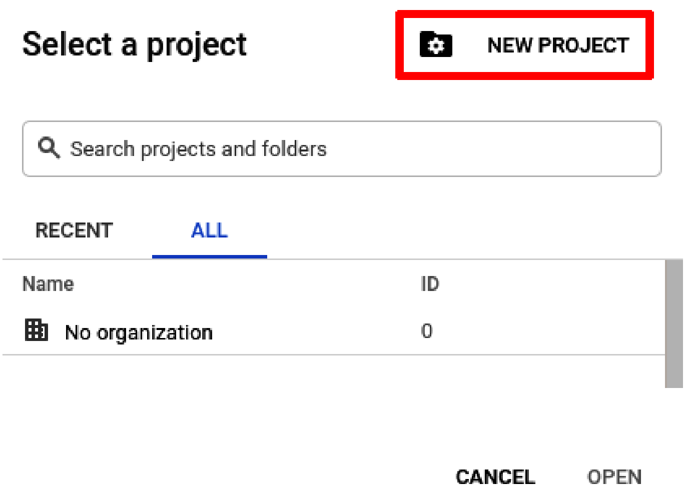


Figure 2.4 – RHEL8 in Google Cloud – Organization menu

7. Name it `RHEL8` and click **CREATE**:

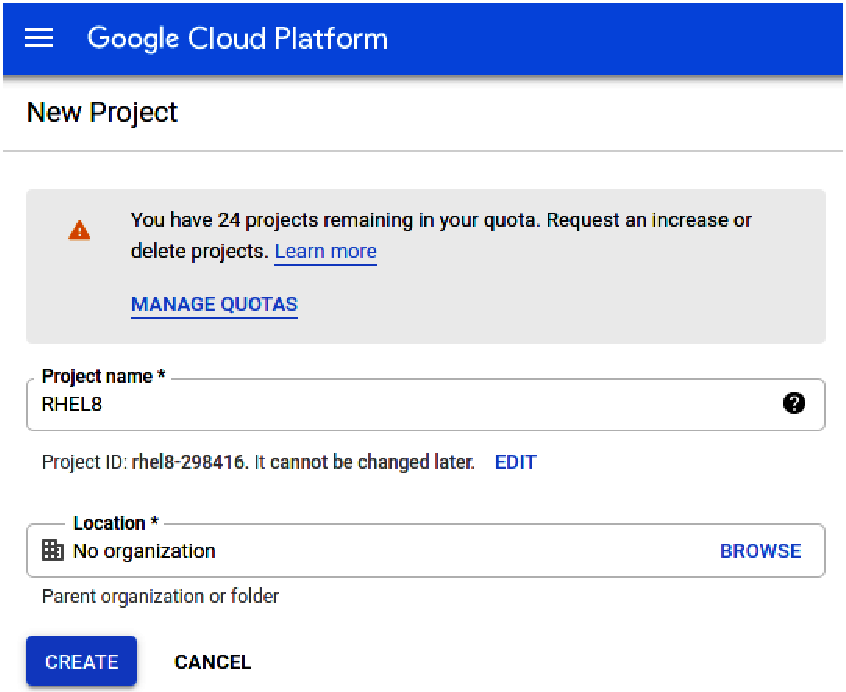


Figure 2.5 – RHEL8 in Google Cloud – Organization menu; create new project

> **Important note**
> Depending on how your Google account is configured, you may need to enable billing after this step.

8. Go to the top-left menu (also called **Hamburger Menu**, with three horizontal lines next to it), click on **Compute Engine**, and then click on **VM Instances**:

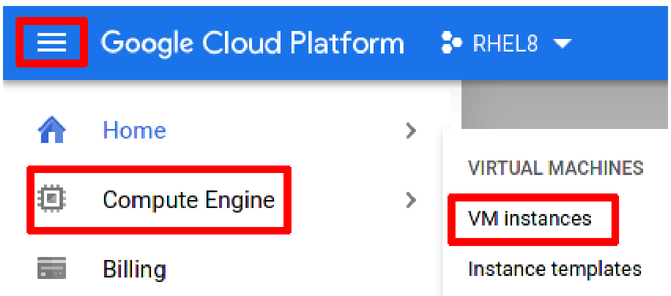


Figure 2.6 – RHEL8 in Google Cloud – Access the VM Instances menu

9. Once **Compute Engine** is ready (this may take a few minutes), click on **Create**:

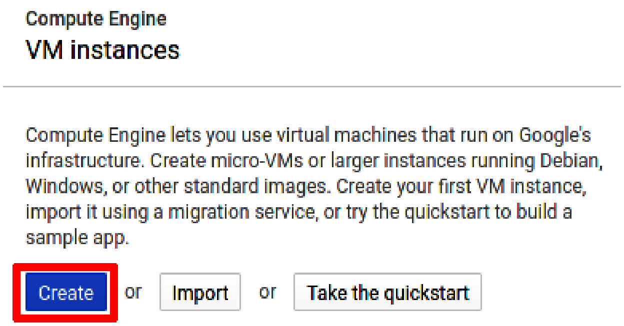


Figure 2.7 – RHEL8 in Google Cloud – create new VM instance

10. We will name the instance `rhel8-instance`:

Figure 2.8 – RHEL8 in Google Cloud – Create new VM instance; name

11. Select the most convenient region (or leave the one already provided):

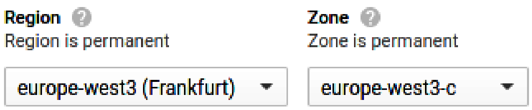


Figure 2.9 – RHEL8 in Google Cloud – Create new VM instance, region, and zone

12. Set the machine family and type to **General purpose | e2-medium**:

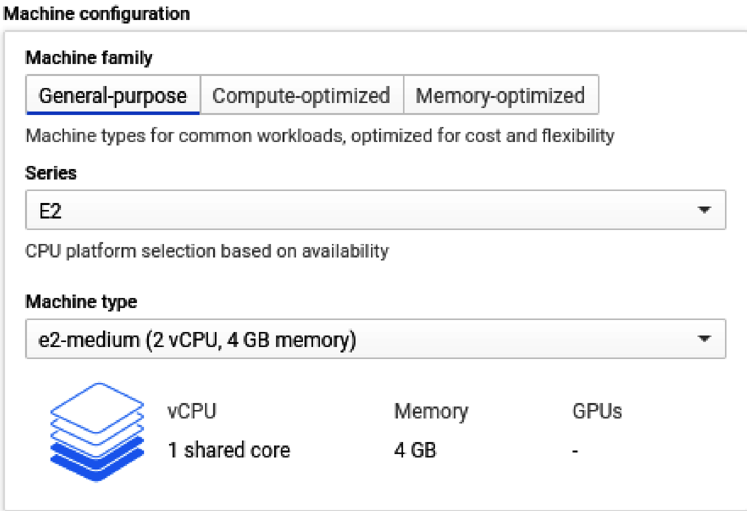


Figure 2.10 – RHEL8 in Google Cloud – Create new VM instance, type, and size

13. Click **Change** next to boot disk:

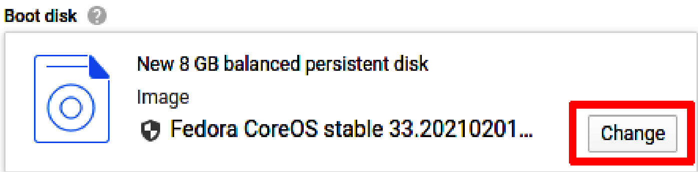


Figure 2.11 – RHEL8 in Google Cloud – Changing the boot disk

14. Change **Operating system** to **Red Hat Enterprise Linux** and **Version** to **Red Hat Enterprise Linux 8**. Then, click **Select**:

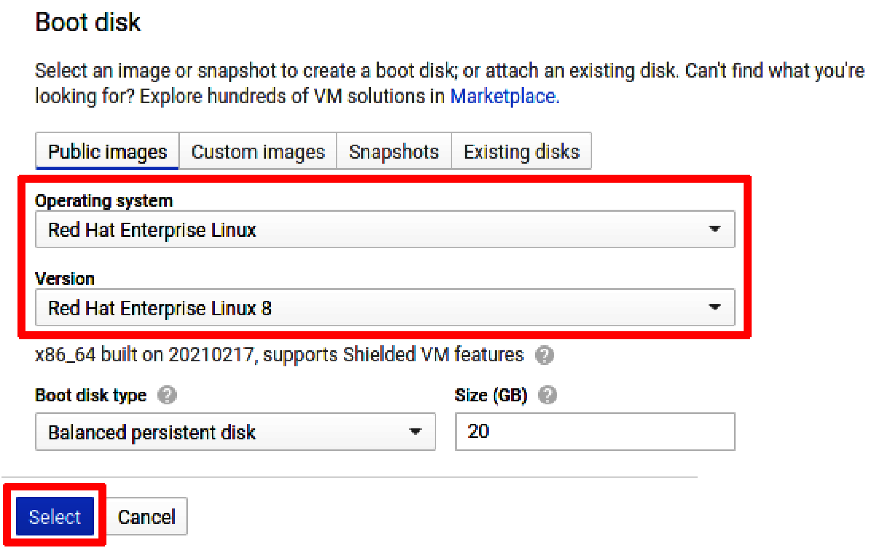


Figure 2.12 – RHEL8 in Google Cloud – Create new VM instance, image selection, and disk size

15. Click **Create** and wait for the instance to be created:

Figure 2.13 – RHEL8 in Google Cloud – VM instance list

16. Later, we will learn how to connect via `SSH`. Now, click on the triangle next to `SSH`, under **Connect**, and select **Open in browser window**, as follows:

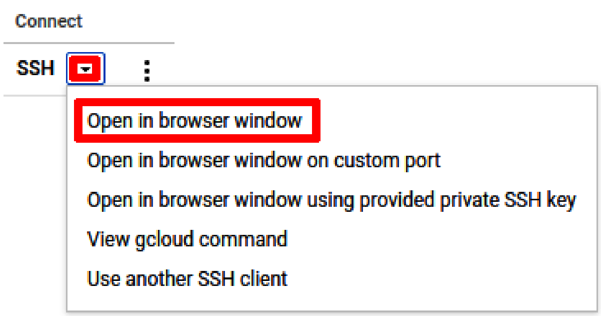


Figure 2.14 – RHEL8 in Google Cloud – VM instance, access console

17. With that, your fresh RHEL8 instance will be deployed, as shown in the following screenshot:

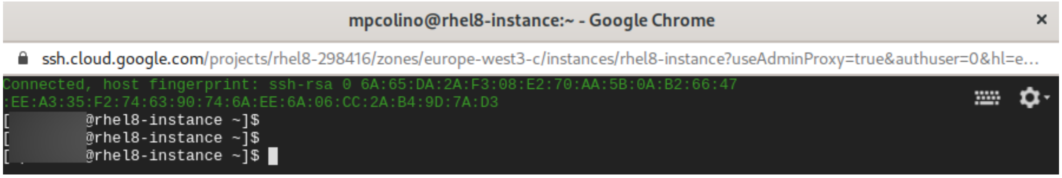


Figure 2.15 – RHEL8 in Google Cloud – VM instance, console

It takes some time to get set up in the cloud, configure your account, and find the `SSH` key (which will be shown in *Chapter 8*, *Administering Systems Remotely*) but once it's all set up, it's easy to get a new instance up and running.

To become an administrator, you only need to run the following command:

```
[miguel@rhel8-instance ~]$ sudo -i
[root@rhel8-instance ~]#
```

Now, you can check the time configuration with `timedatectl` and change it:

```
[root@rhel8-instance ~]# timedatectl
               Local time: Sat 2020-12-12 17:13:29 UTC
           Universal time: Sat 2020-12-12 17:13:29 UTC
                 RTC time: Sat 2020-12-12 17:13:29
                Time zone: UTC (UTC, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
[root@rhel8-instance ~]# timedatectl set-timezone Europe/Madrid
[root@rhel8-instance ~]# timedatectl
               Local time: Sat 2020-12-12 18:20:32 CET
           Universal time: Sat 2020-12-12 17:20:32 UTC
                 RTC time: Sat 2020-12-12 17:20:32
                Time zone: Europe/Madrid (CET, +0100)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

You can also change the language configuration with `localectl`:

```
[root@rhel8-instance ~]# localectl
   System Locale: LANG=en_US.UTF-8
       VC Keymap: us
      X11 Layout: n/a
```

To change `locale` or language support, you will need to install its *language package* first, as follows:

```
[root@rhel8-instance ~]# yum install glibc-langpack-es -y
... [output omitted] ...
[root@rhel8-instance ~]# localectl set-locale es_ES.utf8
[root@rhel8-instance ~]# localectl
   System Locale: LANG=es_ES.utf8
       VC Keymap: us
      X11 Layout: n/a
```

Now, you have a machine configured that you can use throughout this book. These locale changes are not needed to proceed, just to create a machine with the same configuration as in the previous chapter.

Now that we know how to automatically redeploy VMs using Anaconda and how to get instances in the cloud, let's move on and look at some of the best practices to be taken into account when performing installations.

Installation best practices

Red Hat Enterprise Linux installations have many options you can choose from, and what you choose should be tailored for your specific use case. However, some common recommendations apply. Let's look at the most common types.

The first type is **blueprints**:

- Standardize the core installation and create a blueprint for it:
  - This blueprint shall be minimal enough to serve as the base for all other blueprints and deployments.
- Build a set of blueprints for common cases when needed:
  - Try to use an automation platform to build extended cases (that is, Ansible).
  - Try to make the cases modular (that is, App Server; database blueprints can be combined into one single machine).
  - Be aware of the requirements you must apply to your templated blueprints and adapt to the environments you will use.

The second type is **software**:

- The less software that's installed, the smaller the attack surface. Try to keep servers with the minimal set of packages required on it for it to run and operate (that is, try not to add a graphical user interface to your servers).
- Standardize the installed tools where possible to be able to react quickly in case of emergency.
- Package your third-party applications so that you have healthy life cycle management (whether with RPM or in containers).
- Establish a patching schedule.

The third type is **networking**:

- In virtual machines, try not to overuse the number of network interfaces.
- In physical machines, use interface teaming/bonding whenever possible. Segment networks using VLANs.

The fourth type is **storage**:

- For servers, use **Logical Volume Management** (**LVM**) where possible (usually everything but `/boot` or `/boot/efi`).
- If you think you will need to reduce your filesystems, use *ext4*; otherwise, go for the default of *xfs*.

- Partition the disk carefully:
  - Keep the default boot partition with its default size. If you change it, enlarge it (you may need space there during upgrades).
  - The default swap partition is the safest bet, unless the third-party software has specific requirements.
  - For long-lived systems, have at least separate partitions for `/` (root) `/var`, `/usr`, `/tmp`, and `/home`, and consider even a separate one for `/var/log` and `/opt` (for ephemeral cloud instances or short-lived systems, this does not apply).

The fifth type is **security**:

- Do not disable *SELinux*. It has been improved a lot in the latest versions and it's very likely that it won't interfere with your system (if required, set it in permissive mode instead of fully disabling it).
- Do not disable the firewall. Automate port opening with the service deployment.
- Redirect logs to a central location whenever possible.
- Standardize the security tools and configuration that you want to install to check system integrity and audit (that is, *AIDE*, *logwatch*, and *auditd*).
- Review software install (*RPM*) *GPG* keys, as well as ISO images, to ensure integrity.
- Try to avoid using passwords (especially for your root account) and use strong ones where needed.
- Review your systems with *OpenSCAP* to check on security (if needed, create your own hardware SCAP profile with help from your security team).

Finally, we will look at the **miscellanea** type:

- Keep system time synchronized.
- Review *logrotate* policies to avoid "disk full" errors due to logs.

Following these best practices will help you avoid issues and make the installed base more manageable. With that, you know how to deploy Red Hat Enterprise Linux on a system in a structured, repeatable manner while providing services to other teams in a fast and resilient fashion.

Summary

In the previous chapter, we mentioned how to prepare a machine that we can work with throughout this book. An alternative to that is using cloud instances, with which we could be consuming virtual machine instances from the public cloud, which may simplify our consumption and provide us with enough free credit to prepare for *RHCSA*. Also, once the self-training process is complete, the machines can be still used to provide your own public services (such as deploying a blog).

Understanding the need to standardize your environments and the impact of doing so is also important when you're working with Linux as a professional. It is key to start with a good set of practices (automating installations, keeping track of installed software, reducing the attack surface, and so on) from the beginning.

Now that you've completed this chapter, you are ready to continue with the rest of this book, since you now have an instance of Red Hat Enterprise Linux 8 available to work and practice with. In the next chapter, we will review the basics of the system to make ourselves comfortable and gain confidence in using the system.

3
Basic Commands and Simple Shell Scripts

Once you have your first **Red Hat Enterprise Linux (RHEL)** system running, you want to start using it, practicing, and getting comfortable with it. In this chapter, we will review the basics of logging into the system, navigating through it, and getting to know the basics in terms of its administration.

The set of commands and practices described in this chapter will be used on many occasions when managing systems, so it is important to study them with care.

The following topics will be covered in this chapter:

- Logging in as a user and managing multi-user environments
- Changing users with the su command
- Using the command line, environment variables, and navigating through the filesystem
- Understanding I/O redirection in the command line
- Filtering output with grep and sed

- Listing, creating, copying, and moving files and directories, links, and hard links
- Using tar and gzip
- Creating basic shell scripts
- Using system documentation resources

Logging in as a user and managing multi-user environments

Login is the process during which a user identifies themselves in the system, usually by providing a **username** and **password**, a couple of pieces of information often referred to as *credentials*.

The system can be accessed in many ways. The initial case for this, which we are covering here, is how a user accesses it when they install a physical machine (such as a laptop) or via the virtualization software interface. In this case, we are accessing the system through a *console*.

During installation, the user was created with an assigned password, and no graphical interface was installed. We will access the system in this case via its *text console*. The first thing we are going to do is to log in to the system using it. Once we start the machine and the boot process is completed, we will enter, by default, the multi-user text mode environment in which we are being requested to provide our **login**:

Figure 3.1 – Login process, username request

The blinking cursor will let us know that we are ready to enter our username, in this case `user`, and then press *Enter*. A line requesting the password will appear:

Figure 3.2 – Login process, password request

We may now type the user's password to complete the log in and, by pressing *Enter* on your keyboard, start a session. Note that no character will be displayed on screen when typing the password to avoid eavesdropping on it. This would be the session running:

```
Red Hat Enterprise Linux 8.3 (Ootpa)
Kernel 4.18.0-240.el8.x86_64 on an x86_64

rhel8 login: user
Password:
Last login: Thu Dec 17 14:45:10 on tty1
[user@rhel8 ~]$
```

Figure 3.3 – Login process, login completed, session running

Now we are fully logged in to the system with the *credentials* for the user named `user`. This will define what we can do in the system, which files we can access, and even how much disk space we have assigned.

The console can have more than one session. To make that possible, we have different terminals through which we can log in. The default terminal can be reached by simultaneously pressing the *Ctrl* + *Alt* + *F1* keys. In our case, nothing will happen as we are already in that terminal. We could move to the second terminal by pressing *Ctrl* + *Alt* + *F2*, to the third one by pressing *Ctrl* + *Alt* + *F3*, and so on for the rest of the terminals (by default, six are allocated). This way, we can run different commands in different terminals.

Using the root account

Regular users will not be able to make changes to the system, such as creating new users or adding new software to the whole system. To do so, we need a user with administrative privileges and for that, the default user is `root`. This user always exists in the system and its identifier (**User Id** or **UID**) has the value `0`.

In the previous installation, we have configured the root password, making the account accessible through the console. To use it by logging in the system, we only need to type, in one of the terminals shown, right next to **login**, the user `root`, then hit *Enter*, and then provide its **Password**, which won't be displayed. This way, we will access the system as the administrator, `root`:

```
Red Hat Enterprise Linux 8.3 (Ootpa)
Kernel 4.18.0-240.el8.x86_64 on an x86_64

Hint: Num Lock on

rhel8 login: root
Password:
Last login: Wed Dec 16 11:18:02 from 192.168.122.1
[root@rhel8 ~]#
```

Figure 3.4 – Login process, login completed as root

Using and understanding the command prompt

The command line that appears once we have logged in and are waiting for our commands to be typed and run is called the **command prompt**.

In its default configuration, it will show the *username* and *hostname* between brackets to let us know with which user we are working. Next, we see the path, in this case ~, which is the shortcut for the **user's home directory** (in other words, `/home/user` for `user`, and `/root` for `root`)

The last part and, probably the most important one, is the symbol before the prompt:

- The `$` symbol is used for regular uses with no administrative privileges.
- The `#` symbol is used for root or once a user has acquired administrative privileges.

> **Important note**
>
> Be careful when using a prompt with the # sign as you will be running as an administrator and the system will, very likely, not stop you from damaging it.

Once we have identified ourselves in the system, we are logged in and have a running session. It is time to learn how to change from one user to the other in the next section.

Changing users with the su command

As we have entered a **multi-user system**, it is logical to think that we will be able to change between users. Even when this can be done easily by opening a session for each, sometimes we want to act as other users in the same session we are in.

To do so, we can use the `su` tool. The name of the tool is usually referred to as **Substitute User**.

Let's use that last session in which we logged in as `root` and turn ourselves into the `user` user.

Before doing so, we can always ask which user I am logged in with by running the `whoami` command:

```
[root@rhel8 ~]# whoami
root
```

Now we can make the change from `root` to `user`:

```
[root@rhel8 ~]# su user
[user@rhel8 root]$ whoami
user
```

Now we have a session as the `user` user. We could finish this session by using the `exit` command:

```
[user@rhel8 root]$ exit
exit
[root@rhel8 ~]# whoami
root
```

As you may have seen, when we are logged in as `root`, we can act as any user without knowing its password. But how can we impersonate `root`? We can do so by running the `su` command and specifying the `root` user. In this case, the root user's password will be requested:

```
[user@rhel8 ~]$ su root
Password:
[root@rhel8 user]# whoami
root
```

As `root` is the user with the ID `0` and the most important one, when running `su` without specifying the user we want to turn to, it will default to turning ourselves into `root`:

```
[user@rhel8 ~]$ su
Password:
[root@rhel8 user]# whoami
root
```

Each user can define several options in their own environment, such as, for example, their preferred editor. If we want to fully impersonate the other user and take their preferences (or **environment variables**, as they are declared and referred to on many occasions), we can do so by adding a `-` after the `su` command:

```
[user@rhel8 ~]$ su -
Password:
Last login: mar dic 22 04:57:29 CET 2020 on pts/0
[root@rhel8 ~]#
```

Also, we can switch from `root` to `user`:

```
[root@rhel8 ~]# su - user
Last login: Tue Dec 22 04:53:02 CET 2020 from 192.168.122.1 on
pts/0
[user@rhel8 ~]$
```

As you can observe, it behaves as if a new login was done, but within the same session. Now, let's move on to managing the permissions for the different users in the system, as addressed in the next section.

Understanding users, groups, and basic permissions

Multi-user environments are defined by being able to handle more than one user simultaneously. But in order to be able to administer the system resources, there are two capabilities that help with the tasks:

- **Groups**: Can aggregate users and provide permission for them in blocks.

  Each user has a *primary group*.

  By default, a group is created for each user and assigned to it as a primary with the same name as the username.
- **Permissions**: Are assigned to files and determine which users and groups can access each file.

  Standard Linux (and UNIX/POSIX) permissions include *user*, *group*, and *others* (`ugo`).

The whole system comes with a set of permissions assigned by default to each file and directory. Be careful when changing them.

There is a principle in UNIX that Linux inherited that is: *everything is a file*. Even when there may be some corner cases to this principle, it stays true on almost any occasion. It means that a disk is represented as a file in the system (in other words, like `/dev/sdb` mentioned in the installation), a process can be represented as a file (under `/proc`) and many other components in the system are represented as files.

This means that, when assigning permissions to files, we can also assign permissions to many other components and capabilities implemented by them by virtue of the fact that, in Linux, everything is represented as a file.

> **Tip**
>
> **POSIX** stands for **Portable Operating System Interface** and is a family of standards specified by the IEEE Computer Society: `https://en.wikipedia.org/wiki/POSIX`.

Users

Users are a way of providing security limits to people as well as programs running in a system. There are three types of users:

- **Regular users**: Assigned to individuals to perform their job. They have restrictions applied to them.
- **Superuser**: Also referred to as "root." This is the main administrative account in the system and has full access to it.
- **System users**: These are user accounts usually assigned to running processes or "daemons" to limit their reach within the system. System users are not intended to log in to the system.

Users have a number called the **UID (User Id)** that the system uses to internally identify each one of them.

We previously used the `whoami` command to reveal which user we were working with, but to get more information, we will use the `id` command:

```
[user@rhel8 ~]$ id
uid=1000(user) gid=1000(user) groups=1000(user),10(wheel) conte
xt=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

We can also check on the information related to other user accounts in the system, even to get info about `root`:

```
[user@rhel8 ~]$ id root
uid=0(root) gid=0(root) groups=0(root)
```

Now, let's take a look at the information we have received for `user` by running `id`:

- `uid=1000(user)`: The user ID is the numeric identifier of the user in the system. In this case, it is `1000`. Identifiers of 1000 and above are used in RHEL for regular users, whereas 999 and below are reserved for system users.
- `gid=1000(user)`: The group ID is the numeric identifier for the principal group assigned to the user.

- `groups=1000(user),10(wheel)`: These are the groups that the user belongs to, in this case, "user" with **Group ID (GID)** 1000 and "wheel" with GID 10. The "wheel" user group is a special one. It is used in RHEL and many other systems as the group for users that can become administrators by using the `sudo` tool (to be explained later).
- `context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023`: This is the SELinux context for the user. It will define several restrictions in the system by using **SELinux** (to be explained in depth in *Chapter 10*, *Keeping Your System Hardened with SELinux*).

ID-related data is stored in the system in the `/etc/passwd` file. Please note that this file is very sensitive and is better managed by using the tools related to it. In case we want to edit it, we will do so by using `vipw`, a tool that will ensure (among other things) that only one admin is editing the file at any one time. The `/etc/passwd` file contains the info of each user per line. This is the line for `user`:

```
user:x:1000:1000:user:/home/user:/bin/bash
```

Each field is separated by a colon, `:` in each line. Let's review what they mean:

- `user`: The username assigned to the user.
- `x`: The field for the encrypted password. In this case, it shows as `x` because it has moved to `/etc/shadow`, which is not directly accessible by regular users, to make the system more secure.
- `1000` (first one): *The UID* value.
- `1000` (second one): *The GID* value.
- `user`: A description of the account.
- `/home/user`: The home directory assigned to the user. This will be the default directory (or folder if you prefer) that the user will work on and where their preferences will be stored.
- `/bin/bash`: The command interpreter for the user. Bash is the default interpreter in RHEL. Other alternatives, such as `tcsh`, `zsh`, or `fish` are available to be installed in RHEL.

Groups

Groups are a way of assigning certain permissions to a subset of users in a dynamic way. As an example, let's imagine a scenario where we have a finance team. We can create the *finance* group and provide permission to access, read, and write the `/srv/finance` directory. When the finance team has a new hire, in order to provide them with access to that folder, we only need to add the user assigned to this person to the `finance` group (this also works if someone leaves the team; we will only have to remove their account from the `finance` group).

Groups have a number called the **GID** that the system uses to identify them internally.

The data for groups is stored in the system in the `/etc/group` file. To edit this file in a way that ensures consistency and avoids corruption, we must use the `vigr` tool. The file contains one group per line with different fields separated by a colon, `:`. Let's take a look at the line for the `wheel` group:

```
wheel:x:10:user
```

Let's review what each field means:

- `wheel`: This is the name of the group. In this case, this group is special as it is configured to be used, by default, as the one to provide admin privileges to regular users.
- `x`: This is the group password field. It's currently obsolete and should always contain `x`. It is kept for compatibility purposes.
- `10`: This is the GID value for the group itself.
- `user`: This is the list of the users belonging to that group (separated by commas, such as `user1`, `user2`, and `user3`).

The types of groups are as follows:

- **Primary group**: This is the group assigned to the files newly created by the user.
- **Private group**: This is a specific group, with the same name as the user, that is created for each user. When adding a new user account, a private group will be automatically created for it. It is very common that the "primary group" and "private group" are one and the same.
- **Supplementary group**: This is another group usually created for specific purposes. By way of an example, we can see the `wheel` group for enabling admin privileges to users, or the `cdrom` group for providing access to CDs and DVD devices in the system.

File permissions

To review **file permissions**, we are going to log in to the system as `root`. We will use the `ls` command to list files and we will review the permissions associated with them. We will learn more on how to change permissions in *Chapter 5*, *Securing Systems with Users, Groups, and Permissions*.

Once logged in to the system as `root`, we can run the `ls` command:

```
[root@rhel8 ~]# ls
anaconda-ks.cfg
```

This shows the files present in the *root user home directory*, represented by `~`. In this case, it shows the *kickstart* file created by *Anaconda* that we reviewed in the previous chapter.

We could get the long version of the list by appending the `-l` option to `ls`:

```
[root@rhel8 ~]# ls -l
total 4
-rw-------. 1 root root 1393 Dec  7 16:45 anaconda-ks.cfg
```

We see the following in the output:

- `total 4`: This is the total space, in kilobytes, occupied in the disk by the files (note that we are using 4K blocks, so every file under that size will occupy a minimum of 4K).
- `-rw-------.`: These are the permissions assigned to the file.

The structure of the permissions can be seen in the following diagram:

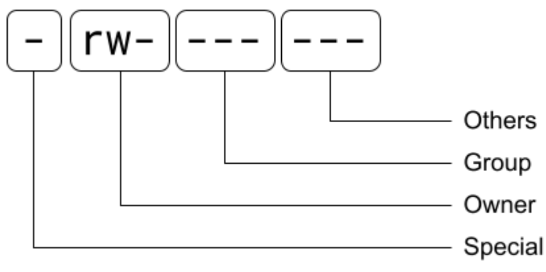


Figure 3.5 – Linux permissions structure

The first character is for the *special permissions* that the file may have. If it is a regular file, and has no special permission (as in this case), it will appear as `-`:

- Directories will appear with `d`. Consider that in Linux, everything is a file, and directories are a file with special permissions.
- Links, usually symbolic links, will appear with a `l`. These behave like a shortcut to a file from a different directory.
- Special permissions to run a file as a different user or group, called **setuid** or **setgid**, will appear as `s`.
- A special permission so that the owner can only remove or rename the file, called the **sticky bit**, will appear as `t`.

The next three characters, `rw-`, are the permissions for the *owner*:

- The first one, `r`, is the read permission assigned.
- The second one, `w`, is the write permission assigned.
- The third one, `x`, not present and shown as `-`, is the executable permission. Note that executable permission for directories means being able to enter them.

The next three characters, `---`, are for the *group* permissions and work the same way as the owner permission. In this case, no group access is granted.

The final three characters, `---`, are the permissions for *others*, which means users and/or groups do not show as the ones assigned to the file:

- `1`: This indicates the number of **links** (hard links) to this file. This is intended, among other things, so that we do not delete a file used in another folder.
- `root`: This indicates the (first-time) owner of the file.
- `root`: This indicates the (second-time) group assigned to the file.
- `1393`: This indicates the size in bytes.
- `Dec 7 16:45`: This indicates the date and time that the file was last modified.
- `anaconda-ks.cfg`: This indicates the filename.

When we list a directory (referred to in other systems as *folder*), the output will show the contents of the directory itself. We can list the info for the directory itself with the `-d option`. Let's now take a look at `/etc`, the directory that stores the system-wide configuration:

```
[root@rhel8 ~]# ls -l -d /etc
drwxr-xr-x. 81 root root 8192 Dec 23 17:03 /etc
```

As you can see, it's quite easy to obtain information pertaining to files and directories in the system. Let's now learn more about the command line and how to navigate the filesystem, in order to move around the system easily, in the next section.

Using the command line, environment variables, and navigating through the filesystem

As we have seen before, once we *log in* to the system, we have access to the command line. It's important to navigate the command line and the filesystem well in order to feel comfortable in the environment and make the most of it.

Command line and environment variables

The command line is provided by a program also known as *interpreter* or **shell**. It will behave differently depending on which shell we use, but in this section, we will cover the most widespread shell used in Linux and the one provided by default in RHEL: **bash**.

A simple trick to know which shell you are using is to run the following command:

```
[root@rhel8 ~]# echo $SHELL
/bin/bash
```

The `echo` command will show on screen the content of whatever we give to it. Some content needs to be *substituted* or *interpreted*, like environment variables. The content to be substituted starts with the `$` symbol. In this case, we are telling the system to `echo` the content of the `SHELL` variable. Let's use it for other variables:

```
[root@rhel8 ~]# echo $USER
root
[root@rhel8 ~]# echo $HOME
/root
```

These are **environment variables** that can be customized for every user. Let's now check these for a different user:

```
[root@rhel8 ~]# su - user
Last login: Wed Dec 23 17:03:32 CET 2020 from 192.168.122.1 on pts/0
[user@rhel8 ~]$ echo $USER
user
[user@rhel8 ~]$  echo $HOME
/home/user
```

As you can see, you can always refer to `$USER` and it will be substituted with the current user, or to `$HOME` and it will be substituted by the directory dedicated to the user, also known as the **home directory**.

These are some of the most common and important *environment variables*:

Variable	Value for user	Usage
`EDITOR`	(unset)	This establishes the default text editor for the user.
`HOME`	`/home/user`	This is the home directory for the current user.
`HOSTNAME`	`rhel8.example.com`	This is the hostname of the system we are logged in to.
`LANG`	`en_US.UTF-8`	This is the language configured for the current user. In this case, US American English with the UTF-8 extensions.
`PATH`	`/home/user/.local/bin:/home/user/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin`	This is a colon-separated list of directories that will be searched in order to run the command we are typing.
`PS1`	`'[\u@\h \W]\$ '`	This defines what information will be shown at the "prompt", in this case, the user (`\u`), the `@` hostname (`\h`), the working directory (`\W`), and the prompt symbol – `$` for user or # for root (`\$`).
`PWD`	`/home/user`	This is the path to the working directory, the directory we are currently in.
`SHELL`	`/bin/bash`	This indicates the shell in use.
`USER`	`user`	This is the username of the current user, in this case, "`user`".

The `~/.bashrc` file is the one that should be edited in order to change these values for the current user.

Navigating the filesystem

Now it's time to move ourselves into the **directory tree** of the system. In Linux and Unix (macOS is a Unix-like system), there are no drive letters but a single directory tree that starts with the *root directory*, represented by `/`. The rest of the content of the system will hang from that folder and any other disk or device to be accessed will be assigned a directory to be accessed.

> **Important note**
>
> The *root directory* and the *home directory* for the *root user* are two different things. The *root user* has assigned, by default, the home directory, `/root`, whereas the *root directory* is the mother of all directories in the system and is represented by `/`.

We can see which directory we are in by running the `pwd` command:

```
[user@rhel8 ~]$ pwd
/home/user
```

We can change directory by using the `cd` command:

```
[user@rhel8 ~]$ cd /var/tmp
[user@rhel8 tmp]$ pwd
/var/tmp
```

As you already know, there is a **shortcut** for the home directory of the current user, ~. We can use this shortcut to go to it:

```
[user@rhel8 tmp]$ cd ~
[user@rhel8 ~]$ pwd
/home/user
```

Some shortcuts for directories include the following:

- **"~":** This is the home of the current user.
- **".":** This is the current directory.
- **"..":** This is the parent directory.
- **"-":** This is the directory used previously.

More details on managing files and directories in Linux and RHEL are available in the *Listing, creating, copying, and moving files and directories, links, and hard links* section.

Bash autocomplete

Shortcuts are a faster way to reach commonly used directories or relative references to the current working directory. However, bash includes some capabilities to reach other directories in a fast way, which is called **autocompletion**. It relies on the *Tab* key (the one with two opposing arrows at the very left of your keyboard, right above *Caps Lock*).

When reaching a folder or a file, we can hit *Tab* to complete its name. For example, if we want to go to the `/boot/grub2` folder, we type the following:

```
[user@rhel8 ~]$ cd /bo
```

Then, when we hit the *Tab* key, this will autocomplete it to `/boot/`, even adding the final / as it is a directory:

```
[user@rhel8 ~]$ cd /boot/
```

Now we type the first letter of the directory we want to go to, `grub2`, which is `g`:

```
[user@rhel8 ~]$ cd /boot/g
```

Then, when we hit the *Tab* key, this will autocomplete it to `/boot/grub2/`:

```
[root@rhel8 ~]# cd /boot/grub2/
```

Now we can hit *Enter* and go there.

If we press *Tab* + *Tab* (pressing *Tab* twice during complete), this will show a list of available targets to complete, for example:

```
[root@rhel8 ~]# cd /r
root/ run/
```

It can also be used to complete commands. We can type a letter, for example, `h`, hit *Tab* + *Tab*, and this will show all the commands starting with `h`:

```
[root@rhel8 ~]# h
halt         hardlink     hash         h dparm      head
help         hexdump      history      hostid       hostname
hostnamectl  hwclock
```

This capability can be extended to help complete other parts of our commands by installing the `bash-completion` package:

```
[root@rhel8 ~]# yum install bash-completion -y
```

Previous commands

There is a way to recover the last commands run, which is referred to as **history**, in case you want to re-run them again. Just press the *up arrow* key (the one with an arrow pointing up) and the previous commands will appear on screen.

If there are too many commands in your history, you can search through them quickly by running the `history` command:

```
[user@rhel8 ~]$ history
    1  su root
    2  su
    3  su -
    4  id
    5  id root
    6  grep user /etc/passwd
    7  echo $USER
    8   echo $HOME
    9  declare
    10  echo $SHELL
    11  echo EDITOR
    12  echo $EDITOR
    13  grep wheel /etc/gro
    14  grep wheel /etc/group
    15  cat /etc/group
    16  grep nobody /etc/group /etc/passwd
```

You can run any of those commands again by using the `!` command. Just run `!` with the number of the command and it will run again:

```
[user@rhel8 ~]$ !5
id root
uid=0(root) gid=0(root) groups=0(root)
```

> **Tip**
>
> The command `!!` will run the very last command again, no matter which number.

Now it is time to enjoy your superfast command line. Let's learn more on the structure of directories in Linux, to know where to go in order to find things, in the next section.

Filesystem hierarchy

Linux has a standard, maintained by the *Linux Foundation*, that defines the **filesystem hierarchy** and is used in almost every Linux distribution, including *RHEL*. This standard is known as **FHS**, or **Filesystem Hierarchy Standard**. Let's review here the most important folders in the standard and the system itself:

Directory	Stands for	Use/Purpose
`/`	Root directory	The main directory of the filesystem hierarchy on which the other directories hang.
`/boot`	Bootable files	Files used by the system to boot. It usually has a dedicated partition for itself.
`/dev`	Devices	The directory that includes files representing the devices connected to it (whether disks, keyboards, or audio devices, they are represented here).
`/etc`	"Etcetera" or "editable text configuration"	Configuration files that apply system wide.
`/home`	Home	The directory to include home directories, where user personal settings and files are intended to be stored.
`/media`	External media	Intended to provide system-wide access to removable media.
`/mnt`	Mount	For temporarily mounted filesystems.
`/opt`	Optional	Optional application software packages. When building or installing your own software, for more than one system, land it here.
`/proc`	Processes	A special virtual filesystem representing the processes running in the system.
`/root`	Root's home	The home folder for the "root" super-user. This is not present in `/home` in case it gets full so that root can always log in to the system.
`/run`	Run	This is the runtime data for processes. It is intended to be cleaned upon reboot.
`/srv`	Service	This is the content to be served externally, such as web pages of files in an FTP service.
`/sys`	System	This contains system information related to kernel features and connected devices.
`/usr`	User binaries and libs	This contains the read-only user data, including libraries, binaries, headers, sources, and other shared data.
`/usr/bin`	Binaries	This is the directory for the regular binaries used in the system.
`/usr/lib`	Libraries	This is the directory for holding the libraries used in the system.
`/usr/local`	Local binaries	This indicates local data specific to this host. When building local scripts or binaries for this specific system, host them here.
`/usr/sbin`	Super binaries	This is the directory for the binaries to be used by the super-user only.
`/var`	Variable content	This is the directory to be used to host content to be managed by different programs, from virtual machines to logs.

> **Tip**
>
> Previous versions of RHEL used to have `/bin` for the essential binaries and `/usr/bin` for the non-essential ones. Now, the content of both resides in `/usr/bin`. They also used `/var/lock` and `/var/run` for what is running in `/run`. In addition, they used to have `/lib` for the essential libraries and `/usr/lib` for the non-essential ones, which were consolidated into a single directory, `/usr/lib`. And last but not least, `/sbin` is the directory for the essential super-user binaries, and `/usr/sbin` is the directory for the non-essential ones merged under `/usr/sbin`.

When partitioning, we may well be asking ourselves, where does the disk space go?

These are the allocation values for a "minimal" installation of RHEL 8 and the recommendations:

Directory	**Used Space**	**Recommendation**
`/boot`	194 MB	`boot` will contain booting files and will play an important role when upgrading the system. 1 GB is the minimum, while assigning 2 GB is recommended.
`/etc`	22 MB	`etc` hardly ever grows a lot in size (rarely beyond 50 MB for a server). It is very convenient to leave this folder as part of the root partition and not create a partition for itself.
`/home`	20 KB	`home` will be assigned to user files. In a workstation, it will hold all the working data of the user; however, in a server, it is usually kept with admin's temporary files. It is very convenient to assign a partition to it.
`/root`	36 KB	This is the home partition for the root user and again, hardly ever grows in size. It is convenient to keep it within the root partition.
`/run`	8.6 MB	This is usually mounted as a temporary filesystem (`tmpfs`) that does not use disk space but memory. No changes to the default install are recommended.
`/usr`	1.3 GB	`usr` is the largest folder when installing a machine. It is 1.3 GB in the minimal install, and reaches 4.6 GB for a full-blown workstation. Once the system is in production, the size of this directory won't change. On average, 10 GB is a very good size for it.
`/var`	168 MB	`var`, although holding very little data initially, will hold most of the data available in the system once in production. For servers storing large amounts of data, it can be common to split this partition into several others, such as `/var/log` or `/var/lib`, or even, for email servers, `/var/spool`.

It's important to become familiar with the main directories in the system in order to make the best of them. It is recommended to navigate through the different system directories and look at what's in them in order to become comfortable with the structure. In the next section, we will look at how to perform redirections on the command line to learn more about command and file interaction.

Understanding I/O redirection in the command line

We have already run several commands to ascertain information about the system, such as listing files with `ls`, and we have got some information, **output**, from the running command, including, for example, filenames and file sizes. That information, or *output*, can be useful, and we want to be able to work with it, store it, and manage it properly.

When talking about command *output* and also **input**, there are three sources or targets for them that need to be understood:

- **STDOUT**: Also known as **Standard Output**, this is where commands will put their regular messages to provide information on what they are doing. In a terminal, on an interactive shell (like the ones we are using so far), this output will show on screen. This will be the main output managed by us.
- **STDERR**: Also known as **Standard Error**, this is where the commands will put their error messages to be processed. In our interactive shells, this output will also be shown on screen together with the standard output unless we specifically redirect it.
- **STDIN**: Also known as **Standard Input**, this is where the commands get data to be processed.

We will mention these in the next paragraphs to better understand them.

The way in which command input and output is used requires the following operators:

- `|`: A **pipe** operator is used to get the output from one command and make it the input of the next command. It *pipes* data from one command to another.
- `>`: A **redirect** operator is used to put the output of a command into a file. If the file exists, it will be overwritten.
- `<`: **Reverse redirect** can be applied to use a file as input to a command. Using it won't delete the file used as input.
- `>>`: A **redirect and add** operator is used to append the output of a command into a file. If the file does not exist, it will be created with the output provided to it.
- `2>`: A **redirect STDERR** operator will only redirect the output sent to the error message handler. (Note, no space should be included between "2" and ">" in order for this to work!)
- `1>`: A **redirect STDOUT** operator will only redirect the output sent to the standard output and not to the error message handler.

- `>&2`: A **redirect to STDERR** operator will redirect the output to the standard error handler.
- `>&1`: A **redirect to STDOUT** operator will redirect the output to the standard output handler.

To better understand these, we will go through a number of examples in this section and the following one.

Let's get a list of files and put it in a file. First, we list the files in `/var`, using the `-m` option to separate entries with commas:

```
[root@rhel8 ~]# ls -m /var/
adm, cache, crash, db, empty, ftp, games, gopher, kerberos,
lib, local, lock, log, mail, nis, opt, preserve, run, spool,
tmp, yp
```

Now, we run the command again, redirecting the output to the `/root/var-files.txt` file:

```
[root@rhel8 ~]# ls -m /var/ > /root/var-files.txt
[root@rhel8 ~]#
```

As we can see, no output is shown on screen, but we will be able to find the new file in the current working directory, in this case, `/root`, the newly created file:

```
[root@rhel8 ~]# ls /root
anaconda-ks.cfg  var-files.txt
```

To see the content of the file on screen, we use the `cat` command, intended to concatenate the output for several files, but regularly used for this purpose:

```
[root@rhel8 ~]# ls -m /var/ > /root/var-files.txt
[root@rhel8 ~]#
[root@rhel8 ~]# cat var-files.txt
adm, cache, crash, db, empty, ftp, games, gopher, kerberos,
lib, local, lock,
log, mail, nis, opt, preserve, run, spool, tmp, yp
```

We can also add to this file the content of `/var/lib`. First, we can list it:

```
[root@rhel8 ~]# ls -m /var/lib/
alternatives, authselect, chrony, dbus, dhclient, dnf, games,
```

```
initramfs, logrotate, misc, NetworkManager, os-prober,
plymouth, polkit-1, portables, private, rhsm, rpm, rpm-state,
rsyslog, selinux, sss, systemd, tpm, tuned, unbound
```

Now, to append this content to the `/root/var-files.txt` file, we use the `>>` operator:

```
[root@rhel8 ~]# ls -m /var/lib/ >> var-files.txt
[root@rhel8 ~]# cat var-files.txt
adm, cache, crash, db, empty, ftp, games, gopher, kerberos,
lib, local, lock, log, mail, nis, opt, preserve, run, spool,
tmp, yp
alternatives, authselect, chrony, dbus, dhclient, dnf, games,
initramfs, logrotate, misc, NetworkManager, os-prober,
plymouth, polkit-1, portables, private, rhsm, rpm, rpm-state,
rsyslog, selinux, sss, systemd, tpm, tuned, unbound
```

The `/root/var-files.txt` file now contains both the comma-separated list for `/var` and for `/var/lib`.

Now we can try to list a non-existing directory to see an error being printed:

```
[root@rhel8 ~]# ls -m /non
ls: cannot access '/non': No such file or directory
```

The output we see is an error, and it is treated differently by the system than the regular messages. We can try to redirect the output to a file:

```
[root@rhel8 ~]# ls -m /non > non-listing.txt
ls: cannot access '/non': No such file or directory
[root@rhel8 ~]# cat non-listing.txt
[root@rhel8 ~]#
```

We see that using the standard redirect, with a command providing an error message, will show the error message, via `STDERR`, on screen and create an empty file. This is because the file contains the output of the common information messages which are shown via `STDOUT`. We can still capture the output of the error, redirecting `STDERR`, by using `2>`:

```
[root@rhel8 ~]# ls /non 2> /root/error.txt
[root@rhel8 ~]# cat /root/error.txt
ls: cannot access '/non': No such file or directory
```

Now we can redirect the standard output and the error output independently.

Now we want to count the number of files and directories in `/var`. For that, we are going to use the `wc` command, which stands for *word count,* with the option `-w` to focus on only counting words. To do so, we will redirect the output of `ls` to it by using a *pipe* represented by `|`:

```
[root@rhel8 ~]# ls -m /var/ | wc -w
21
```

We can also use it to count the entries in `/etc`:

```
  [root@rhel8 ~]# ls -m /etc/ | wc -w
174
```

Pipes, `|`, are great for reusing the output of one command, and sending it to another command to process that output. Now we know more about using the more common operators to redirect input and output. There are several ways to process that output and we will see more examples in the next section.

Filtering output with grep and sed

The `grep` command is heavily used (and commonly mistyped) in system administration. It helps when finding a pattern in a line, whether in a file or via **standard input** (**STDIN**).

Let's do a recursive search of the files in `/usr` with `find` and put it in `/root/usr-files.txt`:

```
[root@rhel8 ~]# find /usr/ > /root/usr-files.txt
[root@rhel8 ~]# ls -lh usr-files.txt
-rw-r--r--. 1 root root 1,9M dic 26 12:38 usr-files.txt
```

As you can see, it's a file 1.9 MB in size, and it isn't easy to go through it. There is a utility in the system called `gzip` and we want to know which files in `/usr` contain the `gzip` pattern. To do so, we run the following command:

```
[root@rhel8 ~]# grep gzip usr-files.txt
/usr/bin/gzip
/usr/lib64/python3.6/__pycache__/gzip.cpython-36.opt-2.pyc
/usr/lib64/python3.6/__pycache__/gzip.cpython-36.opt-1.pyc
/usr/lib64/python3.6/__pycache__/gzip.cpython-36.pyc
/usr/lib64/python3.6/gzip.py
/usr/share/licenses/gzip
```

```
/usr/share/licenses/gzip/COPYING
/usr/share/licenses/gzip/fdl-1.3.txt
/usr/share/doc/gzip
/usr/share/doc/gzip/AUTHORS
/usr/share/doc/gzip/ChangeLog
/usr/share/doc/gzip/NEWS
/usr/share/doc/gzip/README
/usr/share/doc/gzip/THANKS
/usr/share/doc/gzip/TODO
/usr/share/man/man1/gzip.1.gz
/usr/share/info/gzip.info.gz
/usr/share/mime/application/gzip.xml
```

As you can see, we have found all the files with `gzip` under the `/usr` directory by creating a file with all the content and searching though it with `grep`. Could we do the same without creating the file? Sure we can, by using a *pipe*. We can redirect the output of `find` to `grep` and get the same output:

```
[root@rhel8 ~]# find /usr/ | grep gzip
/usr/bin/gzip
/usr/lib64/python3.6/__pycache__/gzip.cpython-36.opt-2.pyc
/usr/lib64/python3.6/__pycache__/gzip.cpython-36.opt-1.pyc
/usr/lib64/python3.6/__pycache__/gzip.cpython-36.pyc
/usr/lib64/python3.6/gzip.py
/usr/share/licenses/gzip
/usr/share/licenses/gzip/COPYING
/usr/share/licenses/gzip/fdl-1.3.txt
/usr/share/doc/gzip
/usr/share/doc/gzip/AUTHORS
/usr/share/doc/gzip/ChangeLog
/usr/share/doc/gzip/NEWS
/usr/share/doc/gzip/README
/usr/share/doc/gzip/THANKS
/usr/share/doc/gzip/TODO
/usr/share/man/man1/gzip.1.gz
/usr/share/info/gzip.info.gz
/usr/share/mime/application/gzip.xml
```

In this command, the standard output from `find` was sent to `grep` to process it. We can even count the number of instances of files with `wc`, but this time using the `-l` option to count the lines:

```
[root@rhel8 ~]# find /usr/ | grep gzip | wc -l
18
```

We have now concatenated two pipes, one to filter the output and another one to count it. We will find ourselves doing this kind of plumbing often when searching for, and finding, information in the system.

Some very common options for `grep` are as follows:

- `-i`: for **ignore-case**. This will match the pattern whether it's uppercase or lowercase or a combination thereof.
- `-v`: for **invert match**. This will show all entries that do not match the pattern being searched for.
- `-r`: for **recursive**. We can tell grep to search for a pattern in all the files within a directory, while going through all of them (if we have permission).

There is a way to also filter columns in the output provided. Let's say we have a list of files in our home directory, and we want to see the size of it. We run the following command:

```
[root@rhel8 ~]# ls -l
total 1888
-rw-------. 1 root root    1393 dic  7 16:45 anaconda-ks.cfg
-rw-r--r--. 1 root root      52 dic 26 12:17 error.txt
-rw-r--r--. 1 root root       0 dic 26 12:08 non-listing.txt
-rw-r--r--. 1 root root 1917837 dic 26 12:40 usr-files.txt
-rw-r--r--. 1 root root     360 dic 26 12:12 var-files.txt
```

Let's say we only want the size, which is the fifth column, of the content that has `files` in its name. We can use `awk` for that:

```
[root@rhel8 ~]# ls -l | grep files | awk '{ print $5}'
1917837
360
```

The `awk` tool will help us to filter according to the correct column. It is very useful for finding identifiers in processes or for getting a specific list of data in from a long output.

Tip

Consider that awk is super powerful in processing output and that we will use the minimal capability for it.

We could replace the separator with -F and get a list of available users in the system:

```
[root@rhel8 ~]# awk -F: '{ print $1}' /etc/passwd
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
operator
games
ftp
nobody
dbus
systemd-coredump
systemd-resolve
tss
polkitd
unbound
sssd
chrony
sshd
rngd
user
```

The awk and grep tools are very common processing tools in the life of a Linux sysadmin, and it is important to understand them well in order to manage the output provided by the system. We have applied the base knowledge to filter the output received by row and column. Let's now move on to how to manage files in a system so that we can better handle the stored output we have just generated.

Listing, creating, copying, and moving files and directories, links, and hard links

It is important to know how to **manage files and directories** (also known as folders) in a system from the command line. It will serve as a basis for managing and copying important data such as configuration files or data files.

Directories

Let's start by creating a directory to keep some working files. We can do so by running `mkdir`, short for **make directory**:

```
[user@rhel8 ~]$ mkdir mydir
[user@rhel8 ~]$ ls -l
total 0
drwxrwxr-x. 2 user user 6 Dec 23 19:53 mydir
```

Folders can be deleted with the `rmdir` command, short for **remove directory**:

```
[user@rhel8 ~]$ ls -l
total 0
drwxrwxr-x. 2 user user 6 Dec 23 19:53 mydir
[user@rhel8 ~]$ mkdir deleteme
[user@rhel8 ~]$ ls -l
total 0
drwxrwxr-x. 2 user user 6 Dec 23 20:15 deleteme
drwxrwxr-x. 2 user user 6 Dec 23 19:53 mydir
[user@rhel8 ~]$ rmdir deleteme
[user@rhel8 ~]$ ls -l
total 0
drwxrwxr-x. 2 user user 6 Dec 23 19:53 mydir
```

However, `rmdir` will only delete empty directories:

```
[user@rhel8 ~]$ ls /etc/ > ~/mydir/etc-files.txt
[user@rhel8 ~]$ rmdir mydir
rmdir: failed to remove 'mydir': Directory not empty
```

How can we delete a directory and all the other files and directories it contains using the remove (`rm`) command? First, let's just create and remove a single file, `var-files.txt`:

```
[user@rhel8 ~]$ ls /var/ > ~/var-files.txt
[user@rhel8 ~]$ ls -l var-files.txt
-rw-rw-r--. 1 user user 109 Dec 26 15:31 var-files.txt
[user@rhel8 ~]$ rm var-files.txt
[user@rhel8 ~]$ ls -l var-files.txt
ls: cannot access 'var-files.txt': No such file or directory
```

To remove a full directory branch, including its contents, we may use the `-r` option, short for **recursive**:

```
[user@rhel8 ~]$ rm -r mydir/
[user@rhel8 ~]$ ls -l
total 0
```

> **Important note**
> Be very careful when using recursive mode when deleting as there is neither a recovery command for it nor a trash bin to keep files that have been removed in the command line.

Let's take a look at the review table:

Command	Usage
`mkdir`	Creates directories
`rmdir`	Deletes empty directories
`rm -r`	Deletes directories that contain files or other directories recursively

Now that we know how to create and delete directories in a Linux system, let's start copying and moving content.

Copying and moving

Now, let's copy some files to play with them using the `cp` (for **copy**) command. We may get some powerful `awk` examples copied to our home directory:

```
[user@rhel8 ~]$ mkdir myawk
[user@rhel8 ~]$ cp /usr/share/awk/* myawk/
[user@rhel8 ~]$ ls myawk/ | wc -l
26
```

To copy more than one file at the same time, we have used **globbing** with the `*` sign. This works in a way in which specifying the files one by one, we can just type `*` for everything. We can also type the initial characters and then `*`, so let's try it by copying some more files using globbing, first:

```
[user@rhel8 ~]$ mkdir mysystemd
[user@rhel8 ~]$ cp /usr/share/doc/systemd/* mysystemd/
[user@rhel8 ~]$ cd mysystemd/
[user@rhel8 mysystemd]$ ls
20-yama-ptrace.conf  CODING_STYLE  DISTRO_PORTING  ENVIRONMENT.
md  GVARIANT-SERIALIZATION  HACKING  NEWS  README  TRANSIENT-
SETTINGS.md  TRANSLATORS  UIDS-GIDS.md
```

You will see that running `ls TR*` shows only those files that start with `TR`:

```
[user@rhel8 mysystemd]$ ls TR*
TRANSIENT-SETTINGS.md  TRANSLATORS
```

It will work the same way with the file ending:

```
[user@rhel8 mysystemd]$ ls *.md
ENVIRONMENT.md  TRANSIENT-SETTINGS.md  UIDS-GIDS.md
```

As you can see, it shows only those files ending in `.md`.

We can copy a full branch of files and directories with the *recursive* option for `cp`, which is `-r`:

```
[user@rhel8 mysystemd]$ cd ~
[user@rhel8 ~]$ mkdir myauthselect
[user@rhel8 ~]$ cp -r /usr/share/authselect/* myauthselect
[user@rhel8 ~]$ ls myauthselect/
default  vendor
```

The recursive option is very useful for copying complete branches. We could also move directories or files easily with the `mv` command. Let's put all our new directories together into a newly created directory called `docs`:

```
[user@rhel8 ~]$ mv my* docs/
[user@rhel8 ~]$ ls docs/
myauthselect  myawk  mysystemd
```

You can see that with `mv`, you do not need to use the recursive option to manage a full branch of files and directories. It can also be used to rename files and/or directories:

```
[user@rhel8 ~]$ cd docs/mysystemd/
[user@rhel8 mysystemd]$ ls
20-yama-ptrace.conf  CODING_STYLE  DISTRO_PORTING  ENVIRONMENT.
md  GVARIANT-SERIALIZATION  HACKING  NEWS  README  TRANSIENT-
SETTINGS.md  TRANSLATORS  UIDS-GIDS.md
[user@rhel8 mysystemd]$ ls -l NEWS
-rw-r--r--. 1 user user 451192 Dec 26 15:59 NEWS
[user@rhel8 mysystemd]$ mv NEWS mynews
[user@rhel8 mysystemd]$ ls -l NEWS
ls: cannot access 'NEWS': No such file or directory
[user@rhel8 mysystemd]$ ls -l mynews
-rw-r--r--. 1 user user 451192 Dec 26 15:59 mynews
```

There is a special command for creating empty files, which is `touch`:

```
[user@rhel8 ~]$ ls -l  docs/
total 4
drwxrwxr-x. 4 user user   35 Dec 26 16:08 myauthselect
drwxrwxr-x. 2 user user 4096 Dec 26 15:51 myawk
drwxrwxr-x. 2 user user  238 Dec 26 16:21 mysystemd
[user@rhel8 ~]$ touch docs/mytouch
[user@rhel8 ~]$ ls -l  docs/
total 4
drwxrwxr-x. 4 user user   35 Dec 26 16:08 myauthselect
drwxrwxr-x. 2 user user 4096 Dec 26 15:51 myawk
drwxrwxr-x. 2 user user  238 Dec 26 16:21 mysystemd
-rw-rw-r--. 1 user user    0 Dec 26 16:27 mytouch
```

When applied to an existing file or folder, it will update its access time to the current one:

```
[user@rhel8 ~]$ touch docs/mysystemd
[user@rhel8 ~]$ ls -l  docs/
total 4
drwxrwxr-x. 4 user user   35 Dec 26 16:08 myauthselect
drwxrwxr-x. 2 user user 4096 Dec 26 15:51 myawk
drwxrwxr-x. 2 user user  238 Dec 26 16:28 mysystemd
-rw-rw-r--. 1 user user    0 Dec 26 16:27 mytouch
```

Let's check the review table:

Command	Usage
`cp`	Copies a set of files in the same source directory
`rm`	Deletes a set of files in the same directory
`cp -r`	Copies a full directory branch, recursively, to a target directory
`touch`	Creates empty files or sets the access time to a file to the current time
`mv`	Renames a file or directory
`mv`	Moves a full directory branch recursively to a target directory

Now we know how to copy, delete, rename, and move files and directories, even full directory branches. Let's now take a look at a different way to work with them – links.

Symbolic and hard links

We can have the same file in two places using **links**. There are two types of links:

- **Hard links**: There will be two entries (or more) to the same file in the filesystem. The content will be written once to disk. Hard links, for the same file, cannot be created in two different filesystems. Hard links cannot be created for directories.
- **Symbolic links**: A symbolic link is created pointing to a file or directory in any place in the system.

Both are created using the `ln`, for *link*, utility.

Let's now create hard links:

```
[user@rhel8 ~]$ cd docs/
[user@rhel8 docs]$ ln mysystemd/README MYREADME
[user@rhel8 docs]$ ls -l
total 20
```

```
drwxrwxr-x. 4 user user    35 Dec 26 16:08 myauthselect
drwxrwxr-x. 2 user user  4096 Dec 26 15:51 myawk
-rw-r--r--. 2 user user 13826 Dec 26 15:59 MYREADME
drwxrwxr-x. 2 user user   238 Dec 26 16:28 mysystemd
-rw-rw-r--. 1 user user     0 Dec 26 16:27 mytouch
[user@rhel8 docs]$ ln MYREADME MYREADME2
[user@rhel8 docs]$ ls -l
total 36
drwxrwxr-x. 4 user user    35 Dec 26 16:08 myauthselect
drwxrwxr-x. 2 user user  4096 Dec 26 15:51 myawk
-rw-r--r--. 3 user user 13831 Dec 26 16:32 MYREADME
-rw-r--r--. 3 user user 13831 Dec 26 16:32 MYREADME2
drwxrwxr-x. 2 user user   238 Dec 26 16:28 mysystemd
-rw-rw-r--. 1 user user     0 Dec 26 16:27 mytouch
drwxrwxr-x. 2 user user     6 Dec 26 16:35 test
```

Check the increasing number of references to the file (in bold in the previous example).

Now let's create a symbolic link to a directory with `ln -s` (*s for symbolic*):

```
[user@rhel8 docs]$ ln -s mysystemd mysystemdlink
[user@rhel8 docs]$ ls -l
total 36
drwxrwxr-x. 4 user user    35 Dec 26 16:08 myauthselect
drwxrwxr-x. 2 user user  4096 Dec 26 15:51 myawk
-rw-r--r--. 3 user user 13831 Dec 26 16:32 MYREADME
-rw-r--r--. 3 user user 13831 Dec 26 16:32 MYREADME2
drwxrwxr-x. 2 user user   238 Dec 26 16:28 mysystemd
lrwxrwxrwx. 1 user user     9 Dec 26 16:40 mysystemdlink -> 
mysystemd
-rw-rw-r--. 1 user user     0 Dec 26 16:27 mytouch
drwxrwxr-x. 2 user user     6 Dec 26 16:35 test
```

Check how the symbolic link created is treated as a different type when listing it, as it starts with `l` for *link* (in bold in the previous example) instead of `d` for *directory* (also in bold in the previous example).

> **Tip**
>
> When in doubt as to what to use, be it a hard link or a symbolic link, use the symbolic link as the default choice.

Let's check the review table:

Command	Usage
`ln`	Creates a hard link for a file in the same filesystem
`ln -s`	Creates a symbolic link to a file or directory that could cross different filesystems

As you can see, creating links and symbolic links is super simple and can help in having access to the same file or directory from different locations. In the next section, we will cover how to pack and compress a set of files and directories.

Using tar and gzip

Sometimes, we want to pack a full directory, including files, into a single file for backup purposes or simply to share it more easily. The command that can help aggregate files into one is `tar`.

First, we need to install `tar`:

```
[root@rhel8 ~]# yum install tar -y
```

We can try by creating, as `root`, a backup of the `/etc` directory branch:

```
[root@rhel8 ~]# tar -cf etc-backup.tar /etc
tar: Removing leading '/' from member names
[root@rhel8 ~]# ls -lh etc-backup.tar
-rw-r--r--. 1 root root 21M dic 27 16:08 etc-backup.tar
```

Let's check the options used:

- `-c`: Short for create. TAR can put files together but also unpack them.
- `-f`: Short for file. We specify that the next parameter will be working with a file.

We can try to unpack it:

```
[root@rhel8 ~]# mkdir tmp
[root@rhel8 ~]# cd tmp/
[root@rhel8 tmp]# tar -xf ../etc-backup.tar
```

```
[root@rhel8 tmp]# ls
etc
```

Let's check the new option used:

- `-x`: for extraction. It unpacks a TAR file.

Please realize that we created a directory called `tmp` to work on and that we pointed to the parent directory of `tmp` by using the `..` shortcut (which refers to the parent directory to the current working directory).

Let's `gzip` to compress a file. We can copy `/etc/services` and compress it:

```
[root@rhel8 etc]# cd ..
[root@rhel8 tmp]# cp /etc/services .
[root@rhel8 tmp]# ls -lh services
-rw-r--r--. 1 root root 677K dic 27 16:16 services
[root@rhel8 tmp]# gzip services
[root@rhel8 tmp]# ls -lh services.gz
-rw-r--r--. 1 root root 140K dic 27 16:16 services.gz
```

Please note that when using `gzip`, this will compress the specified file, adding the `.gz` extension to it and the original file will not be kept. Also, be aware that the newly created file is 1/5 of the size of the original file.

To recover it, we can run `gunzip`:

```
-rw-r--r--. 1 root root 140K dic 27 16:16 services.gz
[root@rhel8 tmp]# gunzip services.gz
[root@rhel8 tmp]# ls -lh services
-rw-r--r--. 1 root root 677K dic 27 16:16 services
```

Now we can combine the two of them, packing and compressing them:

```
[root@rhel8 ~]# tar cf etc-backup.tar /etc/
tar: Removing leading '/' from member names
[root@rhel8 ~]# ls -lh etc-backup.tar
-rw-r--r--. 1 root root 21M dic 27 16:20 etc-backup.tar
[root@rhel8 ~]# gzip etc-backup.tar
[root@rhel8 ~]# ls etc-backup.tar.gz
etc-backup.tar.gz
```

```
[root@rhel8 ~]# ls -lh etc-backup.tar.gz
-rw-r--r--. 1 root root 4,9M dic 27 16:20 etc-backup.tar.gz
```

This way, we pack and compress in two steps.

The `tar` command is smart enough to be able to perform packing and compression in a single step:

```
[root@rhel8 ~]# rm -f etc-backup.tar.gz
[root@rhel8 ~]# tar -czf etc-backup.tar.gz /etc/
tar: Removing leading '/' from member names
[root@rhel8 ~]# ls -lh etc-backup.tar.gz
-rw-r--r--. 1 root root 4,9M dic 27 16:22 etc-backup.tar.gz
```

Let's check the new option:

- `-z`: This compresses the newly created tar file with `gzip`. It is also applicable to decompress.

We may want to review that same option while decompressing:

```
[root@rhel8 ~]# cd tmp/
[root@rhel8 tmp]# rm -rf etc
[root@rhel8 tmp]# tar -xzf ../etc-backup.tar.gz
[root@rhel8 tmp]# ls
etc
```

As you can see, it's very easy to pack and compress files using `tar` and `gzip`. There are other available compression methods with higher rations, such as `bzip2` or `xz`, that you may want to try, too. Now, let's move on to combine all the commands that we have learned into a powerful way to automate – by creating shell scripts.

Creating basic shell scripts

As a system administrator, or sysadmin, there will be times when you want to run a series of commands more than once. You can do this manually by running each command every time; however, there is a more efficient way to do so, by creating a **shell script**.

A shell script is nothing more than a text file with a list of commands to be run, and a reference to the shell that will interpret it.

In this book, we will not cover how to use a **text editor**; however, we will provide three recommendations for text editors in Linux that could help:

- **Nano**: This is probably the easiest text editor to use for beginners. Lean, simple, and straightforward, you may want to start by installing it and giving it a try.
- **Vi** or **Vim**: Vi is the default text editor available in RHEL, included even in the minimal install, and in many Linux distributions. Even if you are not going to use it every day, it's good to familiarize yourselves with the basics of it as it will be present in almost any Linux system you will use. **Vim** stands for **vi-improved**.
- **Emacs**: This is probably the most advanced and complex text editor ever. It can do everything and beyond, including reading emails or helping with a bit of psychoanalysis via **Emacs Doctor**.

We can create our first shell script by editing a new file called `hello.sh` with the following line as its content:

```
echo ''hello world!''
```

Then we can run it by using the `bash` **command interpreter** with the following line:

```
[root@rhel8 ~]# bash hello.sh
hello world!
```

There is a different way to do this where we do not need to type `bash`. We can add an initial line referencing the interpreter, so the file content for `hello.sh` looks like this:

```
#!/bin/bash
echo ''hello world!''
```

Now we are changing the permissions so as to make it executable:

```
[root@rhel8 ~]# ls -l hello.sh
-rw-r--r--. 1 root root 32 dic 27 18:20 hello.sh
[root@rhel8 ~]# chmod +x hello.sh
[root@rhel8 ~]# ls -l hello.sh
-rwxr-xr-x. 1 root root 32 dic 27 18:20 hello.sh
```

And we run it just like this:

```
[root@rhel8 ~]# ./hello.sh
hello world!
```

We have created our first shell script. Congratulations!

> **Tip**
> The commands, in order to be run in any working directory, must be in the path, as stated by the `$PATH` variable. If our command (or shell script) is not in one of the directories specified in the path, we will specify the running directory, in this case, using the `.` shortcut for the current directory and the `/` separator.

Let's use some variables in it. We can define a variable by simply putting the name of it and the value we want for it. Let's try replacing the word `world` with a variable. To use it, we prepend the `$` symbol to the name of the variable and it will be used. The script will look like this:

```
#!/bin/bash
PLACE=''world''
echo ''hello $PLACE!''
```

We can run the script, obtaining the same output as before:

```
[root@rhel8 ~]# ./hello.sh
hello world!
```

To have more clarity, when using the value of the variable, we will put the name of it between curly braces, `{`"and "`}`, and take this as a good practice.

The previous script will look like this:

```
#!/bin/bash
PLACE=''world''
echo ''hello ${PLACE}!''
```

Now we know how to create a basic script, but we may want to have a deeper control of it by using some programmatic capabilities, starting with loops. Let's go for it!

for loops

What if we want to run the same command over a list of places? That's what a `for` **loop** is used for. It can help iterate over a set of elements, such as a list or a counter, for example.

The `for` loop syntax is as follows:

- `for`: To specify the iteration
- `do`: To specify the action
- `done`: To close the loop

We can define a space-separated list to try it and iterate through it with our first `for` loop:

```
#!/bin/bash
PLACES_LIST=''Madrid Boston Singapore World''
for PLACE in ${PLACES_LIST}; do
echo ''hello ${PLACE}!''
done
```

Let's run it. The output will look like this:

```
[root@rhel8 ~]# ./hello.sh
hello Madrid!
hello Boston!
hello Singapore!
hello World!
```

Using the `for` loop can be very interesting when **reading the list from an external command**. We can do so by putting the external command between `$(` and `)`.

> **Tip**
>
> Backticks, `'`, can also be used to run a command and get its output as a list, but we will stick to the previous expression for clarity.

One example of the external command to be used can be `ls`. Let's create the `txtfiles.sh` script with the following content:

```
#!/bin/bash
for TXTFILE in $(ls *.txt); do
  echo ''TXT file ${TXTFILE} found! ''
done
```

Make it executable and run it:

```
[root@rhel8 ~]# chmod +x txtfiles.sh
[root@rhel8 ~]# ./txtfiles.sh
TXT file error.txt found!
TXT file non-listing.txt found!
TXT file usr-files.txt found!
TXT file var-files.txt found!
```

You see how we can now iterate over a set of files, including, for example, changing their names, finding and replacing content in them, or simply making a specific backup of a selection of files.

We've seen several ways in which to iterate a list with the `for` loop, which can be very useful when it comes to automating tasks. Now, let's move on to another programmatic capability in scripts – conditionals.

if conditionals

Sometimes, we may want to execute something different for one of the elements in a list, or if a **condition** is happening. We can use the `if` conditional for this.

The `if` conditional syntax is `if`: to specify the condition.

Conditions are usually specified between brackets, `[` and `]`.

- `then`: To specify the action
- `fi`: To close the loop

Let's change our previous `hello.sh` script to say `hello to Madrid` in Spanish, like this:

```
#!/bin/bash
PLACES_LIST=''Madrid Boston Singapore World''
for PLACE in ${PLACES_LIST}; do
    if [ ${PLACE} = ''Madrid'' ]; then
        echo ''¡Hola ${PLACE}!''
    fi
done
```

Then, run it:

```
[root@rhel8 ~]# ./hello.sh
¡Hola Madrid!
```

We have a problem; it only says `hello to Madrid`. What happens if we want to run the previous code on the ones not matching the condition? That's when we extend the conditional using `else` for the items that do not match. The syntax is as follows:

- `else`: This is used as a `then` element when the condition is *not* matched.

And now we have an example of a conditional using `else`:

```
#!/bin/bash
PLACES_LIST=''Madrid Boston Singapore World''
for PLACE in ${PLACES_LIST}; do
    if [ ${PLACE} = ''Madrid'' ]; then
        echo ''¡Hola ${PLACE}!''
    else
        echo ''hello ${PLACE}!''
    fi
done
```

And now we can run it:

```
[root@rhel8 ~]# ./hello.sh
¡Hola Madrid!
hello Boston!
hello Singapore!
hello World!
```

As you see, it's simple to use the conditionals in a script and provide a lot of control on the conditions under which a command is run. We now need to control when something may not be running correctly. That's what the exit codes (or error codes) are for. Let's go for it!

Exit codes

When a program is run, it provides an **exit code**, specifying whether it ran OK or whether there was an issue. That *exit code* is stored in a special variable called `$?`.

Let's take a look at it by running `ls hello.sh`:

```
[root@rhel8 ~]# ls hello.sh
hello.sh
[root@rhel8 ~]# echo $?
0
```

When the program runs OK, the *exit code* is zero, `0`.

What happens when we try to list a file that doesn't exist (or run any other command incorrectly, or that is having issues)? Let's try listing a `nonexistent` file:

```
[root@rhel8 ~]# ls nonexistentfile.txt
ls: cannot access 'nonexistentfile.txt': No such file or
directory
[root@rhel8 ~]# echo $?
2
```

You see, the *exit code* is different to zero. We will go to the documentation and check the number associated with it to understand the nature of the issue.

When running a command in a script, check for the exit code and act accordingly. Let's now review where to find further information on the commands, such as exit codes or other options, in the next section.

Using system documentation resources

The system includes resources to help you while working with it and guide you to improve your sysadmin skills. This is referred to as the **system documentation**. Let's check three different resources available by default in your RHEL installation: man pages, info pages, and other documents.

Man pages

The most common resource used to obtain documentation is **manual pages**, also referred to by the command used to invocate them: `man`.

Almost any utility installed in the system has a man page to help you use it (in other words, specifying all the options for the tools and what do they do). You can run `man tar` and check the output:

```
[root@rhel8 ~]# man tar
TAR(1)                                         GNU TAR Manual
TAR(1)

NAME
       tar - an archiving utility

SYNOPSIS
   Traditional usage
       tar {A|c|d|r|t|u|x}[GnSkUWOmpsMBiajJzZhPlRvwo] [ARG...]

   UNIX-style usage
       tar -A [OPTIONS] ARCHIVE ARCHIVE

       tar -c [-f ARCHIVE] [OPTIONS] [FILE...]

       tar -d [-f ARCHIVE] [OPTIONS] [FILE...]
```

You can see in it (navigate with the *arrow* keys, space bar, and/or *Page Up* and *Page Down*) and exit it by hitting the letter `q` (for *quit*).

There are sections in the `man` page on related topics. It is pretty simple to search those by using the `apropos` command. Let's see this for `tar`:

```
[root@rhel8 ~]# apropos tar
dbus-run-session (1) - start a process as a new D-Bus session
dnf-needs-restarting (8) - DNF needs_restarting Plugin
dracut-pre-udev.service (8) - runs the dracut hooks before
udevd is started
gpgtar (1)           - Encrypt or sign files into an archive
gtar (1)             - an archiving utility
open (1)             - start a program on a new virtual
terminal (VT).
openvt (1)           - start a program on a new virtual
terminal (VT).
```

```
scsi_start (8)        - start one or more SCSI disks
setarch (8)           - change reported architecture in new
program environment and set personalit...
sg_reset (8)          - sends SCSI device, target, bus or host
reset; or checks reset state
sg_rtpg (8)           - send SCSI REPORT TARGET PORT GROUPS
command
sg_start (8)          - send SCSI START STOP UNIT command:
start, stop, load or eject medium
sg_stpg (8)           - send SCSI SET TARGET PORT GROUPS command
systemd-notify (1)    - Notify service manager about start-up
completion and other daemon status c...
systemd-rc-local-generator (8) - Compatibility generator for
starting /etc/rc.local and /usr/sbin...
systemd.target (5)    - Target unit configuration
tar (1)               - an archiving utility
tar (5)               - format of tape archive files
unicode_start (1)     - put keyboard and console in unicode mode
```

As you can see, it matches not only `tar` but also `start`. This isn't perfect, but it can provide helpful information related to tar, such as `gpgtar`.

Man pages have a section. As you can see in the previous example, for `tar`, there are manual pages in two sections, one for the command-line utility (section 1), and one for the archiving format (section 5):

```
tar (1)               - an archiving utility
tar (5)               - format of tape archive files
```

We can access the page in section 5 to understand the format by running the following command:

```
[root@rhel8 ~]# man 5 tar
```

Now we can see the `tar format` page:

```
TAR(5)                                      BSD File Formats Manua
l                                      TAR(5)

NAME
     tar — format of tape archive files
```

```

DESCRIPTION
     The tar archive format collects any number of files,
directories, and other file system objects (symbolic links,
device nodes, etc.) into a single stream of bytes.  The format
was ...
```

You can see that manual pages are a great resource for learning more about the typical commands being used. This is also a fantastic resource as regards the **Red Hat Certified System Administrator** exam. One recommendation is to review all man pages for the commands shown previously in this chapter, as well as for the forthcoming chapters. Consider man pages the main information resource in the system. Let's now review other information resources available.

Info pages

Info pages are usually more descriptive than man pages and are more interactive. They help more in getting started on a topic.

We can try to get `info` for the `ls` command by running the following:

```
[root@rhel8 ~]# info ls
```

We can see the info page for it:

```
Next: dir invocation,  Up: Directory listing

10.1 'ls': List directory contents
==================================

The 'ls' program lists information about files (of any type,
including
directories).  Options and file arguments can be intermixed
arbitrarily,
```

Info pages can *redirect to other topics, shown underlined,* and these can be followed by putting the cursor over them and hitting *Enter*.

As with man pages, press `q` to quit.

Please take some time to review the info pages for the main topics covered in this chapter (in several cases, info pages will not be available, but the ones that are could be very valuable).

What if we do not find a man or info page for a topic? Let's cover this in the next section.

Other documentation resources

For other documentation resources, you can go to the `/usr/share/doc` directory. There, you will find other documents that come with the tools installed in the system.

Let's see how many items we have:

```
[root@rhel8 doc]# cd /usr/share/doc/
[root@rhel8 doc]# ls | wc -l
219
```

You can see that there are 219 directories available under `/usr/share/doc`.

As a good example, let's enter the `bash` directory:

```
[root@rhel8 doc]# cd bash/
```

Then, let's take a look at the `INTRO` file using `less` to read it (remember, you use `q` to quit):

```
[root@rhel8 bash]# ls
bash.html  bashref.html  FAQ  INTRO  RBASH  README
[root@rhel8 bash]# less INTRO

                       BASH - The Bourne-Again Shell

Bash is the shell, or command language interpreter, that will
appear in the GNU operating system.  Bash is an sh-compatible
shell that
incorporates useful features from the Korn shell (ksh) and C
shell
(csh).  It is intended to conform to the IEEE POSIX P1003.2/
ISO 9945.2 Shell and Tools standard.  It offers functional
improvements
```

This is a good read for a better understanding of bash. Now you have a lot of documentation resources that you will be able to be use during your daily tasks as well as in the **RHCSA** exam.

Summary

We have learned in this chapter how to log in to a system with a user and with `root`, understanding the basics of permissions and security. We are now also more comfortable using the command line with autocomplete, navigating through the directories and files, packing, and unpacking them, redirecting command output and parsing it, and even automating processes with shell scripts. More importantly, we have a way to obtain information on what we are doing (or want to do) available in any RHEL system with the included documentation. These skills are the basis of the upcoming chapters. Don't hesitate to revisit this chapter if you feel stuck or if your progress is not as fast as you thought.

Now, it is time to extend your knowledge to encompass more advanced topics in the upcoming chapters. In the following chapter, you will be getting used to the *tools for regular operations*, in which you will review the most common actions taken when managing a system. Enjoy!

4
Tools for Regular Operations

At this point in this book, we've installed a system, and we've covered some of the scripts we can create to automate tasks, so we've reached the point where we can focus on the system itself.

Having a system properly configured requires not only installing it but understanding how to run tasks at specific times, keeping all the services running appropriately, and configuring time synchronization, service management, boot targets (runlevels), and scheduled tasks, all of which we will be covering in this chapter.

In this chapter, you will learn how to check the statuses of services, how to start, stop, and troubleshoot them, as well as how to keep the system clock in sync for your server or your whole network.

The list of topics that will be covered is as follows:

- Managing system services with systemd
- Scheduling tasks with cron and systemd
- Learning about time synchronization with chrony and ntp
- Checking for free resources – memory and disk (free and df)
- Finding logs, using journald, and reading log files, including log preservation and rotation

Technical requirements

It is possible for you to complete this chapter by using the virtual machine we created at the beginning of this book. Additionally, for testing the *NTP server*, it might be useful to create a second virtual machine that will connect to the first one as a client, following the same procedure we used for the first one. Additionally, required packages will be indicated within the text.

Managing system services with systemd

In this section, you will learn how to manage **System Services**, runtime targets, and all about the service status with **systemd**. You will also learn how to manage system boot targets and services that should start at system boot.

`systemd` (which you can learn a bit about at `https://www.freedesktop.org/wiki/Software/systemd/`) is defined as a system daemon that's used to manage the system. It came as a rework of how a system boots and starts, and it looks at the limitations related to the traditional way of doing it.

When we think about system starting, we have the initial **kernel** and **ramdisk** load and execution, but right after that, services and scripts take control to make filesystems available. This helps prepare the services that provide the functionality we want from our system, such as the following:

- Hardware detection
- Additional filesystem activation
- Network initialization (wired, wireless, and so on)
- Network services (time sync, remote login, printers, network filesystems, and so on)
- User-space setup

However, most of the tools that existed before `systemd` came into play and worked on this in a sequential way, causing the whole boot process (from boot to user login) to become lengthy and be subject to delays.

Traditionally, this also meant we had to wait for the required service to be fully available before the next one that depended on it could be started, increasing the total boot time.

Some approaches were attempted, such as using *monit* or other tools that allow us to define dependencies, monitor processes, and even recover from failures, but in general, it was reusing an existing tool to perform other functions, trying to win the race regarding the fastest-booting system.

> **Important Note**
>
> `systemd` redesigned the process to focus on simplicity: start less processes and do more parallel execution. The idea itself sounds easy but requires redesigning a lot of what was taken for granted in the past, to focus on the needs of a new approach to improve OS performance.
>
> This redesign, which has provided lot of benefits, also came with a cost: it drastically changed the way systems used to boot, so there has been a lot of controversy on the adoption of `systemd` by different vendors, and even some efforts by the community to provide systemd-free variants.

Rationalizing how services start so that only those that are required are started is a good way to accomplish efficiency, for example, there is no need to start Bluetooth, printer, or network services when the system is disconnected, there is no Bluetooth hardware, or no one is printing. With fewer services waiting to start, the system boot is not delayed by those waits and focuses on the ones that really need attention.

On top of that, parallel execution allows us to have each service taking the time it needs to get ready but not make others wait, so in general, running services initialization in parallel allows us to maximize the usage of CPU, disk, and so on, and the wait times for each service are used by other services that are active.

`systemd` also pre-creates the listening sockets before the actual daemon is started, so services that have requirements on other services can be started and be on a wait status until its dependencies are started. This is done without them losing any messages that are sent to them, so when the service is finally started, it will act on all the pending actions.

Let's learn a bit more about *systemd* as it will be required for several operations we're going to describe in this chapter.

Systemd comes with the concept of units, which are nothing but configuration files. These units can be categorized as different types, based on their file extension:

Unit type	File extension	Description
Timer	`.timer`	A systemd timer. Described later in this chapter.
Socket	`.socket`	An inter-process communication socket.
Service	`.service`	Describes a system service.
Target	`.target`	A group of system units.
Automount	`.automount`	Defines a filesystem automount point.
Device	`.device`	Defines a device file recognized by the kernel.
Scope	`.scope`	An externally created process.
Slice	`.slice`	Defines a group of hierarchical units that manage system processes.
Path	`.path`	A file or directory in the system.
Mount	`.mount`	Defines a mountpoint in the filesystem.
Swap	`.swap`	A swap device or swap file definition.

> **Tip**
>
> Don't feel overwhelmed by the different `systemd` unit types. In general, the most common ones are **Service**, **Timer**, **Socket**, and **Target**.

Of course, these unit files are expected to be found in some specific folders:

Folder	Description
`/etc/systemd/system/`	Created with the systemctl-enabled services. Takes preference over the runtime ones.
`/usr/lib/systemd/system/`	Units distributed with the packages we've installed on our system.
`/run/systemd/system/`	Units created at runtime. It takes preference over the ones defined by installed packages.

As we mentioned earlier about the sockets, unit files for path, bus, and more are activated when a system's access to that path is performed, allowing services to be started when another one is requiring them. This adds more optimization for lowering system startup times.

With that, we have learned about *systemd* unit types. Now, let's focus on the file structure of unit files.

Systemd unit file structure

Let's get our hands dirty with an example: a system has been deployed with `sshd` enabled, and we need to get it running once the network has been initialized in the **runlevels**, which provide connectivity.

As we mentioned previously, `systemd` uses unit files, and we can check the aforementioned folders or list them with `systemctl list-unit-files`. Remember that each file is a configuration file that defines what *systemd* should do; for example, `/usr/lib/systemd/system/chronyd.service`:

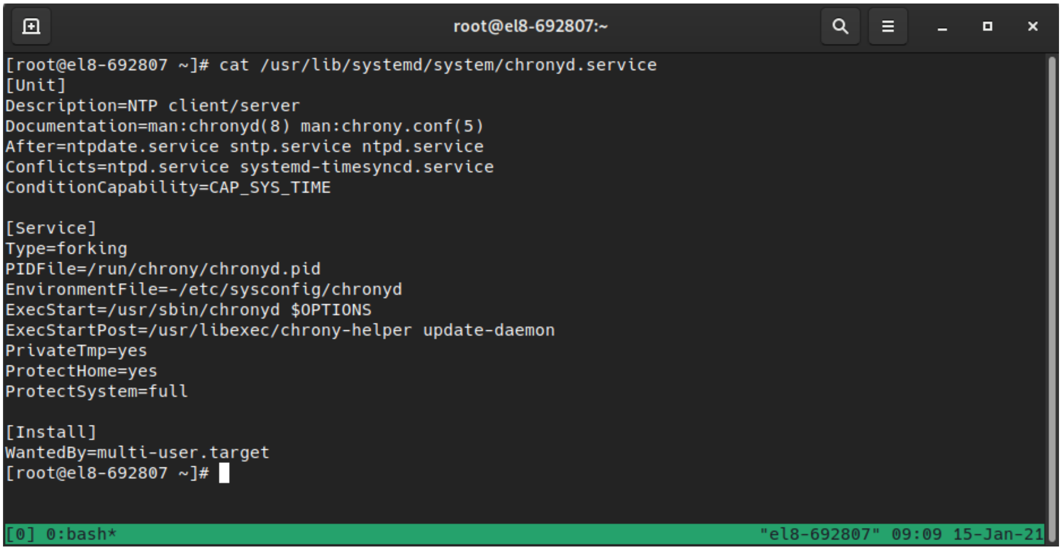


Figure 4.1 – chronyd.service contents

This file defines not only the traditional program to start and the PID file, but the dependencies, the conflicts, and soft dependencies, which provides enough information to `systemd` to decide on the right approach.

If you're familiar with "*inifiles*," this file uses that approach, in that, it uses square brackets, `[` and `]`, for sections and then pairs of `key=value` for the settings in each section.

Section names are case-sensitive, so they will not be interpreted correctly if the proper naming convention is not used.

Section directives are named like so:

- `[Unit]`
- `[Install]`

There are additional entries for each of the different types:

- `[Service]`
- `[Socket]`
- `[Mount]`
- `[Automount]`
- `[Swap]`
- `[Path]`
- `[Timer]`
- `[Slice]`

As you can see, we have specific sections for each type. If we execute `man systemd.unit` it will give you examples, along with all the supported values, for the *systemd* version you're using:

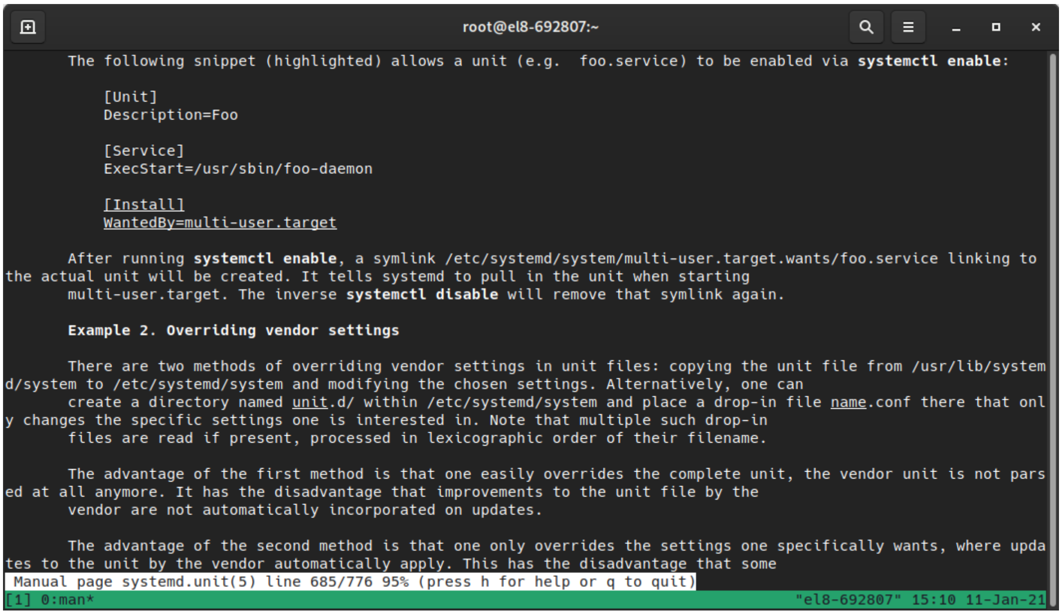


Figure 4.2 – man page of systemd.unit

With that, we have reviewed the file structure of unit files. Now, let's use *systemctl* to actually manage the service's status.

Managing services to be started and stopped at boot

Services can be enabled or disabled; that is, the services will or won't be activated on system startup.

If you're familiar with the previous tools available in RHEL, it was common to use `chkconfig` to define the status of the services based on their default `rc.d/` settings.

A service, such as `sshd`, can be enabled via the following command:

```
#systemctl enable sshd
```

It can also be disabled via the following command:

```
#systemctl disable sshd
```

This results in creating or removing `/etc/systemd/system/multi-user.target.wants/sshd.service`. Notice `multi-user.target` in the path, which is the equivalent of the runlevel we used to configure other approaches such as **initscripts**.

> **Tip**
>
> Although traditional usage of **chkconfig** is provided for compatibility so that `chkconfig sshd on/off` or `service start/stop/status/restart sshd` is valid, it is better to get used to the `systemctl` approach described in this chapter.

The previous commands enable or disable the service at boot, but for executing an immediate action, we need to issue different commands.

To start the `sshd` service, use the following command:

```
#systemctl start sshd
```

To stop it, use the following command:

```
#systemctl stop sshd
```

Of course, we can also check the service's status. The following is an example of looking at `systemd` via `systemctl status sshd`:

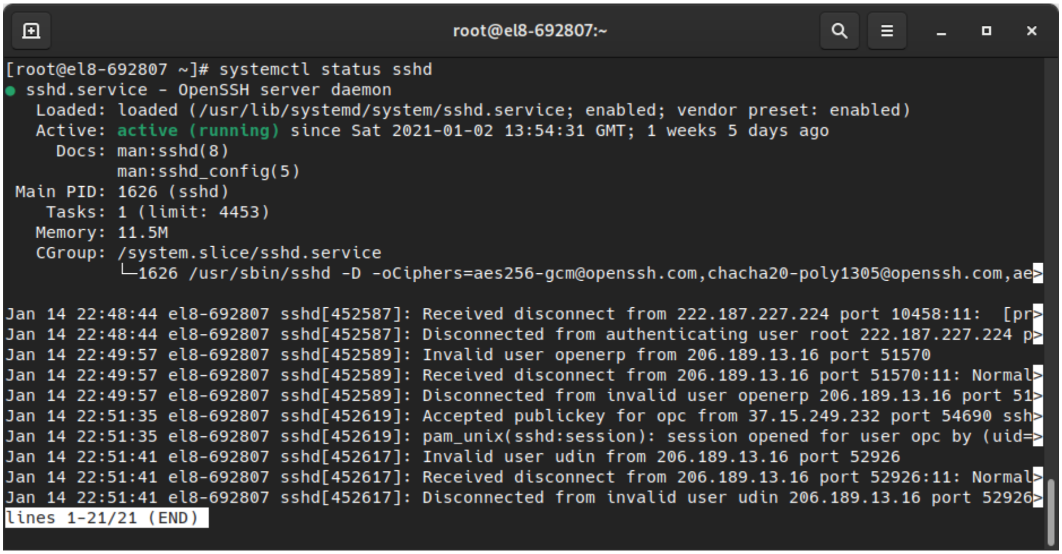


Figure 4.3 – Status of sshd daemon

This status information provides details about the unit file defining the service, its default status at boot, if it is running or not, its PID, some other details about its resource consumption, and some of the most recent log entries for the service, which are quite useful when you're debugging simple service start failures.

One important thing to check is the output of `systemctl list-unit-files` as it reports the defined unit files in the system, as well as the current status and the vendor preset for each one.

Now that we have covered how to start/stop and status check services, let's work on managing the actual system boot status itself.

Managing boot targets

The default status we have defined at boot is important when it comes to talking about **runlevels**.

A runlevel defines a predefined set of services based on usage; that is, they define which services will be started or stopped when we're using a specific functionality.

For example, there are runlevels that are used to define the following:

- **Halt mode**
- **Single user mode**
- **Multi-user mode**
- **Networked multiuser**
- **Graphical**
- **Reboot**

Each of those runlevels allows a predefined set of services to be started/stopped when the runlevel is changed with `init $runlevel`. Of course, levels used to be based on each other, and were very simple:

- Halt stopped all the services and then halted or powered off the system.
- Single user mode starts a shell for one user.
- Multi-user mode enables regular login daemons on the virtual terminals.
- Networked is like multi-user but with the network started.
- Graphical is like networked but with graphical login via display manager (`gdm` or others).
- Reboot is like halt, but at the end of processing services, it issues a reboot instead of a halt.

These runlevels (and the default one when the system is booted) used to be defined in `/etc/inittab`, but the file placeholder reminds us of the following:

```
# inittab is no longer used.
#
# ADDING CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
#
# Ctrl-Alt-Delete is handled by /usr/lib/systemd/system/ctrl-
alt-del.target
#
# systemd uses 'targets' instead of runlevels. By default,
there are two main targets:
#
# multi-user.target: analogous to runlevel 3
# graphical.target: analogous to runlevel 5
```

```
#
# To view current default target, run:
# systemctl get-default
#
# To set a default target, run:
# systemctl set-default TARGET.target
```

So, by making this change to `systemd`, a new way to check the available boot targets and define them is in place.

We can find the available system targets by listing this folder:

```
#ls -l /usr/lib/systemd/system/*.target
```

Or more, correctly, we can use `systemctl`, like so:

```
#systemctl list-unit-files *.target
```

When you examine the output on your system, you will find some compatibility aliases for runlevels 0 to 6 that provide compatibility with the traditional ones.

For example, for regular server usage, the default target will be `multi-user.target` when you're running without graphical mode or `graphical.target` when you're using it.

We can define, as instructed in the placeholder at `/etc/inittab`, the new runlevel to use by executing the following command:

```
#sysemctl set-default TARGET.target
```

We can verify the active one by using the following command:

```
#systemctl get-default
```

This brings us to the next question: *What does a target definition look like*? Let's examine the output in the following screenshot:

Figure 4.4 – Contents of runlevel 5 from its target unit definition

As you can see, it is set as a dependency of another target (**multi-user.target**) and has some requirements on other services, such as **display-manager.service**, and also other conflicts, and the target can only be reached when other targets have completed.

In this way, `systemd` can select the proper order of services to start and the dependencies to reach the configured boot target.

With that, we have covered the service's status, as well as how to start, stop, and enable it on boot, but there are other tasks we should execute in our system but in a periodic way. Let's get further into this topic.

Scheduling tasks with cron and systemd

The skills you will learn in this section will be concerned with scheduling periodic tasks in the system for business services and maintenance.

For regular system usage, there are tasks that need to be executed periodically, ranging from temporary folder cleanup, updating the cache's refresh rate, and performing check-in with inventory systems, among other things.

The traditional way to set them up is via **cron**, which is provided in RHEL8 via the `cronie` package.

Cronie implements a daemon that's compatible with the traditional *vixie cron* and allows us to define both user and system crontabs.

A crontab defines several parameters for a task that must be executed. Let's see how it works.

System-wide crontab

System-wide crontab can be defined in `/etc/crontab` or in individual files at `/etc/cron.d`. Other additional folders exist, such as `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly`, and `/etc/cron.monthly`.

In the folders for *hourly*, *daily*, *weekly*, or *monthly*, you can find scripts or symbolic links to them. When the period since the preceding execution is met (one hour, one day, one week, one month), the script will be executed.

In contrast, in `/etc/crontab` or `/etc/cron.d`, as well as in the user crontabs, the standard definition of jobs is used.

Jobs are defined by specifying the parameters that are relevant to the execution period, the user that will be executing the job (except for user crontabs), and the command to execute:

```
# Run the hourly jobs
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
01 * * * * root run-parts /etc/cron.hourly
```

By looking at the standard `/etc/crontab` file, we can check the meaning of each field:

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR
sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed
```

Based on this, if we check the initial example, `01 * * * * root run-parts /etc/cron.hourly`, we can deduce the following:

- Run at minute `01`.
- Run every hour.
- Run every day.

- Run every month.
- Run every day of the week.
- Run as `root`.
- Execute the `run-parts /etc/cron.hourly` command.

This, in brief, means that the job will run on the first minute of every hour as the `root` user.

Sometimes, it is possible to see an indication, such as */number*, which means that the job will be executed every multiple of that number. For example, */3* will run every 3 minutes if it is on the first column, every 3 hours if it's on the second, and so on.

Any command we might execute from the command line can be executed via cron, and the output will be, by default, mailed to the user running the job. It is a common practice to either define the user that will receive the email via the `MAILTO` variable in the crontab file, or to redirect them to the appropriate log files for the standard output and standard error (`stdout` and `stderr`).

User crontab

Like the system-wide **crontab**, users can define their own crontabs so that tasks are executed by the user. This is, for example, useful for running periodic scripts both for a human user or a system account for a service.

The syntax for user crontabs is the same as it is system-wide. However, the column for the username is not there, since it is always executed as the user is defining the crontab itself.

A user can check its crontab via `crontab -l`:

```
[root@el8-692807 ~]# crontab -l
no crontab for root
```

A new one can be created by editing it via `crontab -e`, which will open a text editor so that a new entry can be created.

Let's work with an example by creating an entry, like this:

```
*/2 * * * * date >> datecron
```

When we exit the editor, it will reply with the following:

```
crontab: installing new crontab
```

This will create a file in the `/var/spool/cron/` folder with the name of the user that created it. It is a text file, so you can check its contents directly.

After some time (at least 2 minutes), we'll have a file in our `$HOME` folder that contains the contents of each execution (because we're using the *append* redirect; that is, `>>`):

```
[root@el8-692807 ~]# cat datecron
Mon Jan 11 21:02:01 GMT 2021
Mon Jan 11 21:04:01 GMT 2021
```

Now that we've covered the traditional crontab, let's learn about the systemd way of doing things; that is, using timers.

Systemd timers

Apart from the regular **Cron Daemon**, a cron-style systemd feature is to use **timers**. A timer allows us to define, via a unit file, a job that will be executed.

We can check the ones that are already available in our system with the following code:

```
>systemctl list-unit-files *.timer
...
timers.target                       static
dnf-makecache.timer                 enabled
fstrim.timer                        disabled
systemd-tmpfiles-clean.timer        static
...
```

Let's see, for example, `fstrim.timer`, which is used on SSD drives to perform a trim at `/usr/lib/systemd/system/fstrim.timer`:

```
[Unit]
Description=Discard unused blocks once a week
Documentation=man:fstrim
..
[Timer]
OnCalendar=weekly
AccuracySec=1h
Persistent=true
…
```

```
[Install]
WantedBy=timers.target
```

The preceding timer sets a weekly execution for `fstrim.service`:

```
[Unit]
Description=Discard unused blocks

[Service]
Type=oneshot
ExecStart=/usr/sbin/fstrim -av
```

As the `fstrim -av` command shows, we are only executing this once.

One of the advantages of having the service timers as unit files, similar to the service itself, is that it can be deployed and updated via the `/etc/cron.d/` files with the regular *cron* daemon, which is handled by *systemd*.

We now know a bit more about how to schedule tasks, but to get the whole picture, scheduling always requires proper timing, so we'll cover this next.

Learning about time synchronization with chrony and NTP

In this section, you will understand the importance of **time synchronization** and how to configure the service.

With connected systems, it is important to keep a source of truth in regards to timing (think about bank accounts, incoming transfer wires, outgoing payments, and more that must be correctly timestamped and sorted). Also, consider tracing logs between users connecting, issues happening, and so on; they all need to be in sync so that we can diagnose and debug between all the different systems involved.

You might think that the system clock, which is defined when the system is provisioned, should be OK, but setting the system clock is not enough as the clocks tend to drift; internal batteries can cause the clock to drift or to even reset, and even intense CPU activity can affect it. To keep clocks accurate, they need to be regularly synced against a reference clock that fixes the drift and tries to anticipate future drifts before the local clock is compared against the remote reference.

The system clock can be synced against a *GPS* unit, for example, or more easily against other systems that have connections to more precise clocks (other GPS units, atomic clocks, and so on). The **Network Time Protocol** (**NTP**) is an internet protocol that's used over UDP to maintain communication between the clients and the servers.

> **Tip**
>
> NTP organizes servers by stratum. A stratum 0 device is a GPS device or an atomic clock that directly sends the signal to a server, a stratum 1 server (primary server) is connected to a stratum 0 device, a stratum 2 server is connected to stratum 1 servers, and so on... This hierarchy allows us to reduce the usage of higher stratum servers but also keep a reliable time source for our systems.

Clients connect to servers and compare the times that are received to reduce the effects of network latency.

Let's see how the NTP client works.

NTP client

In RHEL8, *chrony* acts as both the server (when enabled) and the client (via the `chronyc` command), and it comes with some features that make it suitable for current hardware and user needs, such as fluctuating networks (laptop is suspend/resumed or flaky connections).

One interesting feature is that *chrony* does not **step** the clock after its initial sync, which means that the time doesn't *jump*. Instead, the system clock runs faster or slower so that, after a period of time, it will be in sync with the reference clock it's using. This makes the time to be a continuum from the operating system and application's point of view: the seconds are going faster or slower than what they should be, if compared against a clock, until they match the reference clock.

Chrony is configured via `/etc/chrony.conf` and acts as a client, so it connects to servers to check if they're eligible to be the time source. The main difference between the traditional **server** directive and the **pool** is that the latter can receive several entries while the former only uses one. It is possible to have several servers and pool because, in effect, the servers will be added to the list of possible sources once the duplicates have been removed.

For *pool* or *server* directives, there are several options available (described in `man chrony.conf`), such as `iburst`, which enables faster checks so that they can quickly transition to a synchronized status.

The actual sources for time can be checked with `chronyc sources`:

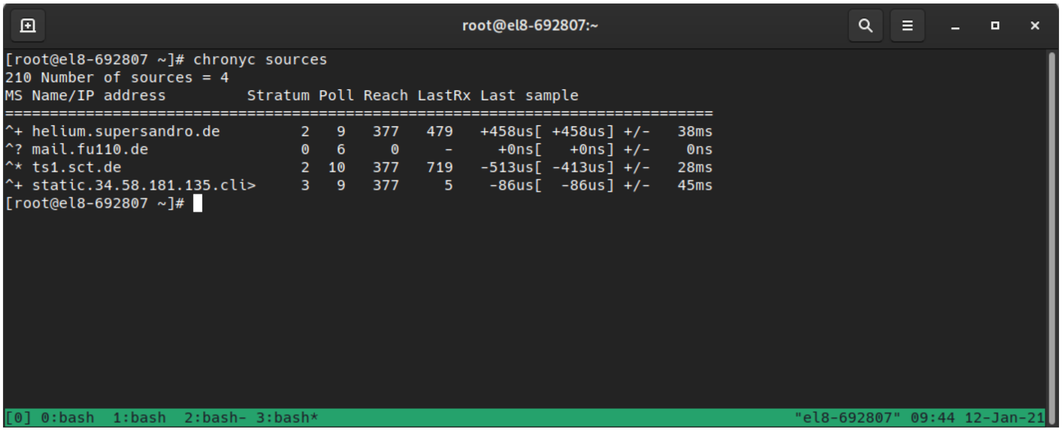


Figure 4.5 – chronyc sources output

As we can see, we know which status is for each server based on the first column (**M**):

- ^: This is a server
- =: This is a peer

In the second column (S), we can see the different statuses for each entry:

- *: This is our current synchronized server.
- +: This is another acceptable time source.
- **?**: This is used to indicate sources that has lost network connectivity.
- **x**: This server is considered a false ticker (its time is considered inconsistent compared to other sources).
- ~: A source that has a high variability (it also appears during daemon startup).

So, we can see that our system is connected to a server that is considering the reference at `ts1.sct.de`, which is a stratum 2 server.

More detailed information can be checked via the `chronyc tracking` command:

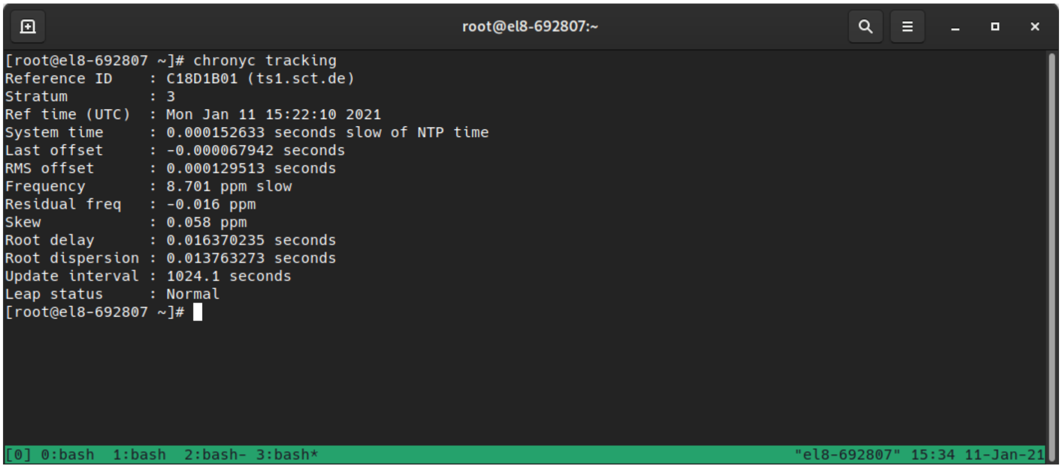


Figure 4.6 – Chronyc tracking output

This provides more detailed information about our clock and our reference clock. Each field in the preceding screenshot has the following meaning:

- **Field**: Description.
- **Reference ID**: ID and name/IP of the server that the system has synchronized.
- **Stratum**: Our stratum level. In this example, our synchronized server is a stratum 3 clock.
- **Ref time**: The last time the reference was processed.
- **System time**: When running in normal mode (without time skip), this references how far away or behind the system is from the reference clock.
- **Last offset**: Estimated offset on the last clock update. If it's positive, this indicates that our local time was ahead of our source.
- **RMS offset**: Long-term average of the offset value.
- **Frequency**: It is the rate at which the system clock would be wrong if *chronyd* is not fixing it, expressed in parts per million.
- **Residual freq**: Reflects any difference between the measurements for the current reference clock.

- **Skew**: Estimated error on the frequency.
- **Root delay**: Total network delays to the stratum -1 synchronized server.
- **Root dispersion**: Total dispersion accumulated through all the computers connected to the stratum -1 server we're synchronized to.
- **Update interval**: Interval between the last two clock updates.
- **Leap status**: It can be Normal, Insert, Delete, or Not synchronized. It reports the leap status.

> **Tip**
>
> Don't underestimate the information sources you have at your fingertips. Remember that when you're preparing for RHCSA exams, the information that's available in the system can be checked during the exam: man pages, documentation included with the program (`/usr/share/doc/program/`), and more. For example, more detailed information about each field listed here can be found via the `man chronyc` command.

To configure the client with additional options, other than the ones provided at install time or via the kickstart file, we can edit the `/etc/chrony.cnf` file.

Let's learn how to convert our system into an NTP server for our network.

NTP server

As we introduced earlier, *chrony* can also be configured as a server for your network. In this mode, our system will be providing accurate clock information to other hosts without consuming external bandwidth or resources from higher-stratum servers.

This configuration is also performed via the `/etc/chrony.conf` file, which is where we will be adding a new directive; that is, `allow`:

```
# Allow NTP client access from all hosts
allow all
```

This change enables *chrony* to listen on all host requests. Alternatively, we can define a subnet or host to listen to, such as `allow 1.1.1.1`. More than one directive can be used to define the different subnets. Alternatively, you can use the *deny* directive to block specific hosts or subnets from reaching our NTP server.

The serving time starts from the base that our server is already synchronized with, as well as an external NTP server, but let's think about an environment without connectivity. In this case, our server will not be connected to an external source and it will not serve time.

chrony allows us to define a fake stratum for our server. This is done via the `local` directive in the configuration file. This allows the daemon to get a higher local stratum so that it can serve the time to other hosts; for example:

```
local stratum 3 orphan
```

With this directive, we're setting the local stratum to 3 and we're using the **orphan** option, which enables a special mode in which all the servers with an equal local stratum are ignored unless no other source can be selected, and its reference ID is smaller than the local one. This means that we can set several NTP servers in our disconnected network but only one of them will be the reference.

Now that we have covered time synchronization, we are going to dive into resource monitoring. Later, we'll look at logging. All of this is related to our time reference for the system.

Checking for free resources – memory and disk (free and df)

In this section, you will check the availability of system **resources** such as **memory** and **disk**.

Keeping a system running smoothly means using monitoring so that we can check that the services are running and that the system provides the resources for them to do their tasks.

There are simple commands we can use to monitor the most basic use cases:

- Disk
- CPU
- Memory
- Network

This includes several ways of monitoring, such as one-shot monitoring, continuously, or even for a period of time to diagnose performance better.

Memory

Memory can be monitored via the `free` command. It provides details on how much *RAM* and *SWAP* are available and in use, which also indicates how much memory is used by shares, buffers, or caches.

Linux tends to use all available memory; any unused RAM is directed toward caches or buffers and memory pages that are not being used. These are swapped out to disk if available:

```
# free
              total        used        free      shared  buff/
cache   available
Mem:         823112      484884       44012        2976
294216       318856
Swap:       8388604      185856     8202748
```

For example, in the preceding output, we can see that the system has a total of 823 MB of RAM and that it's using some swap and some memory for buffers. This system is not swapping heavily as it's almost idle (we'll check the load average later in this chapter), so we should not be concerned about it.

When RAM usage gets high and there's no more swap available, the kernel includes a protection mechanism called **OOM-Killer**. It determines, based on time in execution, resources usage, and more which processes in the system should be terminated to recover the system so that it's functional. This, however, comes at a cost, as the kernel knows about the processes that may have gone out of control. However, the killer may kill databases and web servers and leave the system in an unstable way. For production servers, it is sometimes typical to, instead of letting the OOM-Killer start killing processes in an uncontrolled way, to either tune the values for some critical process so that those are not killed or to cause a system crash.

A system crash is used to collect debug information that can later be analyzed via a dump containing information about what caused the crash, as well as a memory dump that can be diagnosed.

We will come back to this topic in *Chapter 16*, *Kernel Tuning and Managing Performance Profiles with tuned*. Let's move on and check the disk space that's in use.

Disk space

Disk space can be checked via the **df** tool. `df` provides data as output for each filesystem. This indicates the filesystem and its size, available space, percent of utilization, and mount point.

Let's check this in our example system:

```
> df
Filesystem                       1K-blocks     Used Available Use%
Mounted on
devtmpfs                            368596        0    368596   0%
/dev
tmpfs                               411556        0    411556   0%
/dev/shm
tmpfs                               411556    41724    369832  11%
/run
tmpfs                               411556        0    411556   0%
/sys/fs/cgroup
/dev/mapper/rhel-root             40935908 11026516  29909392  27%
/dev/sda2                          1038336   517356    520980  50%
/boot
/dev/sda1                           102182     7012     95170   7%
/boot/efi
tmpfs                                82308        0     82308   0%
/run/user/1000
```

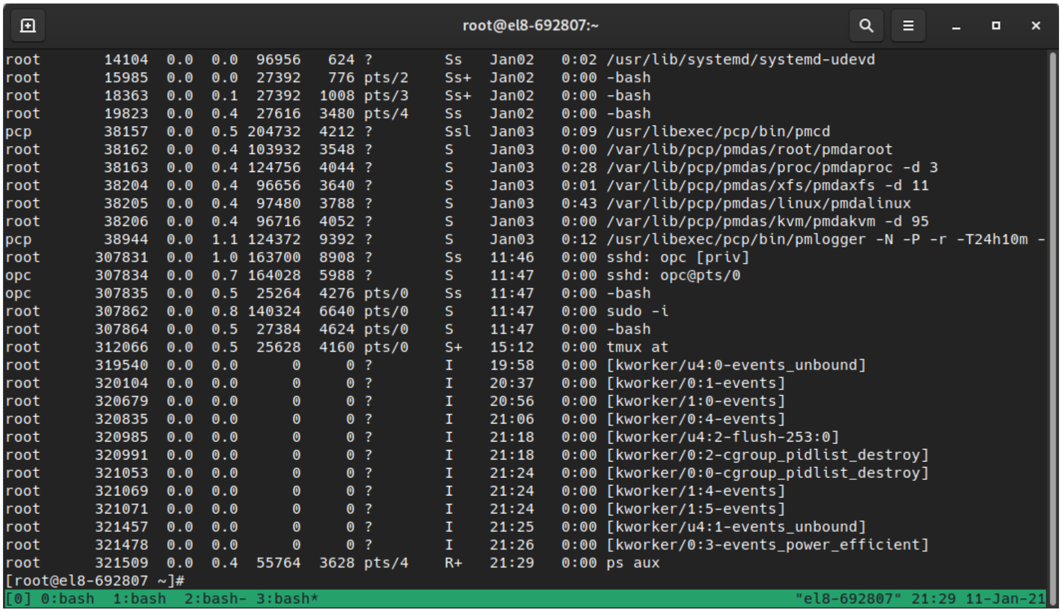
By using this, it's easy to focus on filesystems with higher utilization and less free space to prevent issues.

> **Important Note**
>
> If a file is being written, such as by a process logging its output, removing the file will just unlink the file from the filesystem, but since the process still has the file handle open, the space is not reclaimed until the process is stopped. In case of critical situations where disk space must be made available as soon as possible, it's better to empty the file via a redirect, such as `echo "" > filename`. This will recover the disk space immediately while the process is still running. Doing this with `rm` command will require the process to be finalized.

We'll check out CPU usage next.

CPU

When it comes to monitoring the CPU, we can make use of several tools, such as `ps`:



```
root@el8-692807:~
root       14104  0.0  0.0  96956   624 ?        Ss   Jan02   0:02 /usr/lib/systemd/systemd-udevd
root       15985  0.0  0.0  27392   776 pts/2    Ss+  Jan02   0:00 -bash
root       18363  0.0  0.1  27392  1008 pts/3    Ss+  Jan02   0:00 -bash
root       19823  0.0  0.4  27616  3480 pts/4    Ss   Jan02   0:00 -bash
pcp        38157  0.0  0.5 204732  4212 ?        Ssl  Jan03   0:09 /usr/libexec/pcp/bin/pmcd
root       38162  0.0  0.4 103932  3548 ?        S    Jan03   0:00 /var/lib/pcp/pmdas/root/pmdaroot
root       38163  0.0  0.4 124756  4044 ?        S    Jan03   0:28 /var/lib/pcp/pmdas/proc/pmdaproc -d 3
root       38204  0.0  0.4  96656  3640 ?        S    Jan03   0:01 /var/lib/pcp/pmdas/xfs/pmdaxfs -d 11
root       38205  0.0  0.4  97480  3788 ?        S    Jan03   0:43 /var/lib/pcp/pmdas/linux/pmdalinux
root       38206  0.0  0.4  96716  4052 ?        S    Jan03   0:00 /var/lib/pcp/pmdas/kvm/pmdakvm -d 95
pcp        38944  0.0  1.1 124372  9392 ?        S    Jan03   0:12 /usr/libexec/pcp/bin/pmlogger -N -P -r -T24h10m -
root      307831  0.0  1.0 163700  8908 ?        Ss   11:46   0:00 sshd: opc [priv]
opc       307834  0.0  0.7 164028  5988 ?        S    11:47   0:00 sshd: opc@pts/0
opc       307835  0.0  0.5  25264  4276 pts/0    Ss   11:47   0:00 -bash
root      307862  0.0  0.8 140324  6640 pts/0    S    11:47   0:00 sudo -i
root      307864  0.0  0.5  27384  4624 pts/0    S    11:47   0:00 -bash
root      312066  0.0  0.5  25628  4160 pts/0    S+   15:12   0:00 tmux at
root      319540  0.0  0.0      0     0 ?        I    19:58   0:00 [kworker/u4:0-events_unbound]
root      320104  0.0  0.0      0     0 ?        I    20:37   0:00 [kworker/0:1-events]
root      320679  0.0  0.0      0     0 ?        I    20:56   0:00 [kworker/1:0-events]
root      320835  0.0  0.0      0     0 ?        I    21:06   0:00 [kworker/0:4-events]
root      320985  0.0  0.0      0     0 ?        I    21:18   0:00 [kworker/u4:2-flush-253:0]
root      320991  0.0  0.0      0     0 ?        I    21:18   0:00 [kworker/0:2-cgroup_pidlist_destroy]
root      321053  0.0  0.0      0     0 ?        I    21:24   0:00 [kworker/0:0-cgroup_pidlist_destroy]
root      321069  0.0  0.0      0     0 ?        I    21:24   0:00 [kworker/1:4-events]
root      321071  0.0  0.0      0     0 ?        I    21:24   0:00 [kworker/1:5-events]
root      321457  0.0  0.0      0     0 ?        I    21:25   0:00 [kworker/u4:1-events_unbound]
root      321478  0.0  0.0      0     0 ?        I    21:26   0:00 [kworker/0:3-events_power_efficient]
root      321509  0.0  0.4  55764  3628 pts/4    R+   21:29   0:00 ps aux
[root@el8-692807 ~]#
[0] 0:bash  1:bash  2:bash- 3:bash*                                        "el8-692807" 21:29 11-Jan-21
```

Figure 4.7 – Output of the ps aux command (every process in the system)

The `ps` command is the de facto standard for checking which process is running, as well as resource consumption usage.

As for any other command, we could write a lot about all the different command arguments we could use (so, again, check the man page for details), but as a rule, try to learn about their basic usage or the ones that are more useful for you. For anything else, check the manual. For example, `ps aux` provides enough information for normal usage (every process in the system).

The `top` tool, as shown in the following screenshot, refreshes the screen regularly and can sort the output of running processes, such as CPU usage, memory usage, and more. In addition, `top` also shows a five-line summary of memory usage, `load average`, running processes, and so on:

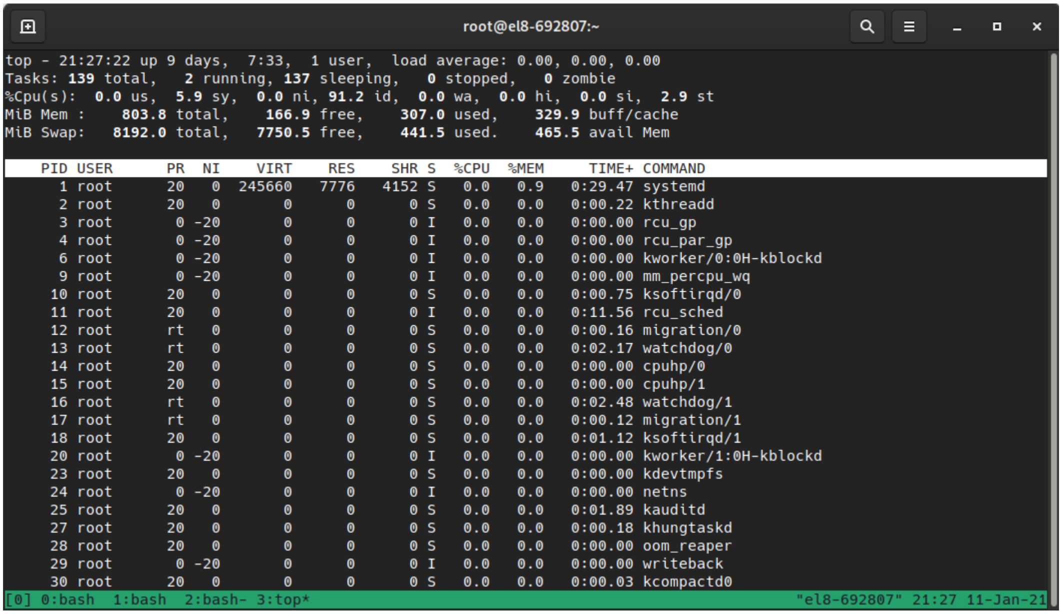


Figure 4.8 – top execution on our test system

CPU usage is not the only thing that may keep our system sluggish. Now, let's learn a bit about load average indicators.

Load average

Load average is usually provided as a group of three numbers, such as `load average: 0.81, 1.00, 1.17`, which is the average that's calculated for 1, 5, and 15 minutes, respectively. This indicates how busy a system is; the higher it is, the worse it will respond. The values that are compared for each time frame give us an idea of whether the system load is increasing (higher values in 1 or 5 and lower on 15) or if it is going down (higher at 15 mins, lower at 5 and 1), so it becomes a quick way to find out if something happened or if it is ongoing. If a system usually has a high load average (over 1.00), it would be a good idea to dig a bit deeper into the possible causes (too much demand for its power, not many resources available, and so on).

Now that we have covered the basics, let's move on and look at some extra checks we can perform on our system resource's usage.

Other monitoring tools

For **monitoring** network resources, we can check the packages that are sent/received for each card via `ifconfig`, for example, and match the values that are received for transmitted packages, received, errors, and so on.

When the goal Is to perform more complete monitoring, we should ensure that the **sysstat** package is installed. It includes some interactive tools such as **iostat**, which can be used to check disk performance, but the most important thing is that it also sets up a job that will collect system performance data on a periodical basis (the default is every 10 minutes). This will be stored in `/var/log/sa/`.

The historical data that's recorded and stored per day (`##`) at `/var/log/sa/sa##` and `/var/log/sa/sar##` can be queried so that we can compare against other days. By running the data collector (which is executed by a *systemd* timer) with a higher frequency, we can increase the granularity for specific periods while an issue is being investigated.

However, the appearance of the *sar* file is showing lots of data:

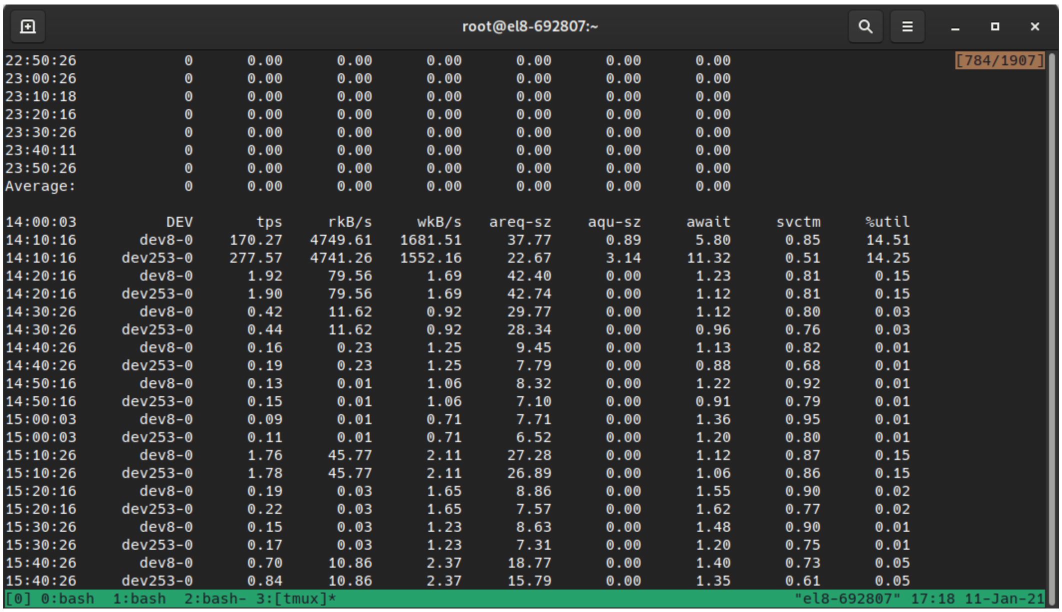


Figure 4.9 – Contents of /var/log/sar02 on the example system

Here, we can see that the 8-0 device had 170.27 transactions per second and 14.51% utilization. In this case, the device's name is using the values for the major/minor, which we can check in the `/dev/` folder. We can see this by running `ls -l /dev/*|grep 8,` as shown in the following screenshot:

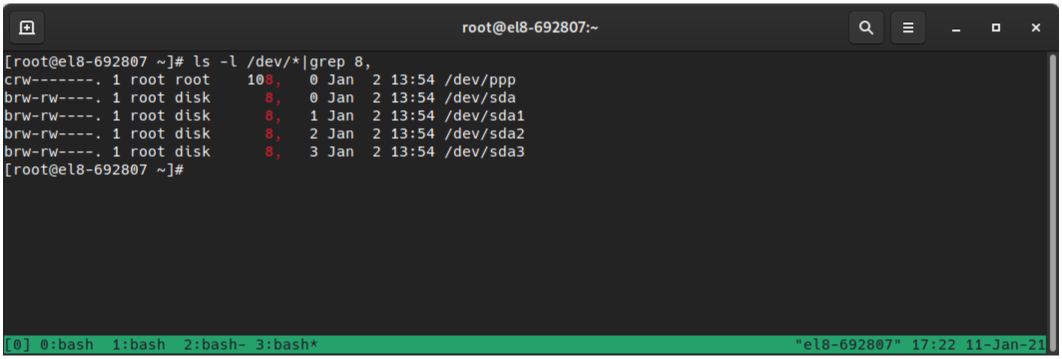


Figure 4.10 – Directory listing for /dev/ for locating the device corresponding to major 8 and minor 0

Here, we can see that this corresponds to the full hard drive statistics at `/dev/sda`.

> **Tip**
> Processing the data via **sar** is a good way to get insights on what's going on with our system, but since the *sysstat* package has been around for a long time in Linux, there are tools such as `https://github.com/mbaldessari/sarstats` that help us process the data that's recorded and present it graphically as a PDF file.

In the following graph, we can see the system service times for the different drives, along with a label at the time the system crashes. This helps us identify the system's activity at that point:

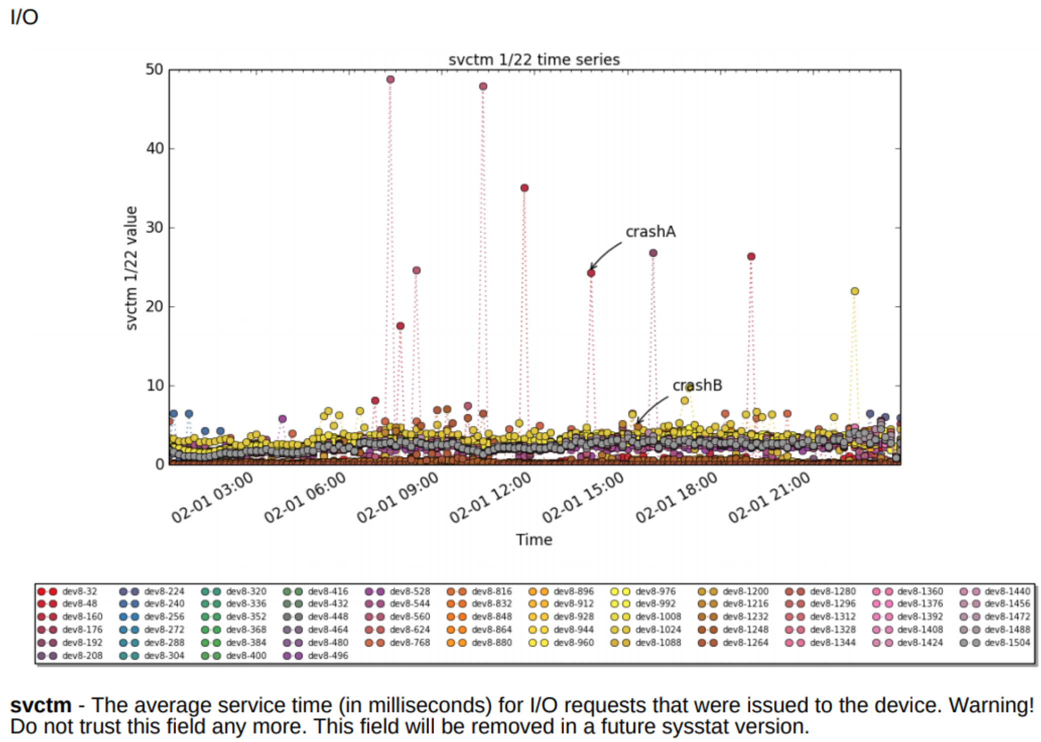


Figure 4.11 – Sarstats graphics for the disk service's time in their example PDF at https://acksyn.org/software/sarstats/sar01.pdf

Modern tooling for monitoring the system's resources has evolved, and **Performance Co-Pilot** (**pcp** and, optionally, the **pcp-gui** packages) can be set up for more powerful options. Just bear in mind that pcp requires us to also start the data collector on the system.

RHEL8 also includes **cockpit**, which is installed by default when we do a server installation. This package provides a set of tools that enable web management for the system, and it can also be made part of other products via plugins that extend its functionality.

The web service provided by cockpit can be reached at your host IP at port `9090`, so you should access `https://localhost:9090` to get a login screen so that we can use our system credentials to log in.

> **Important Tip**
>
> If cockpit is not installed or available, make sure that you execute `dnf install cockpit` to install the package and use `systemctl enable --now cockpit.socket` to start the service. If you are accessing the server remotely, instead of using `localhost`, use the server hostname or IP address after allowing the firewall to connect via `firewall-cmd --add-service=cockpit`, if you haven't done so previously.

After logging in, we will see a dashboard showing the relevant system information and links to other sections, as shown in the following screenshot:

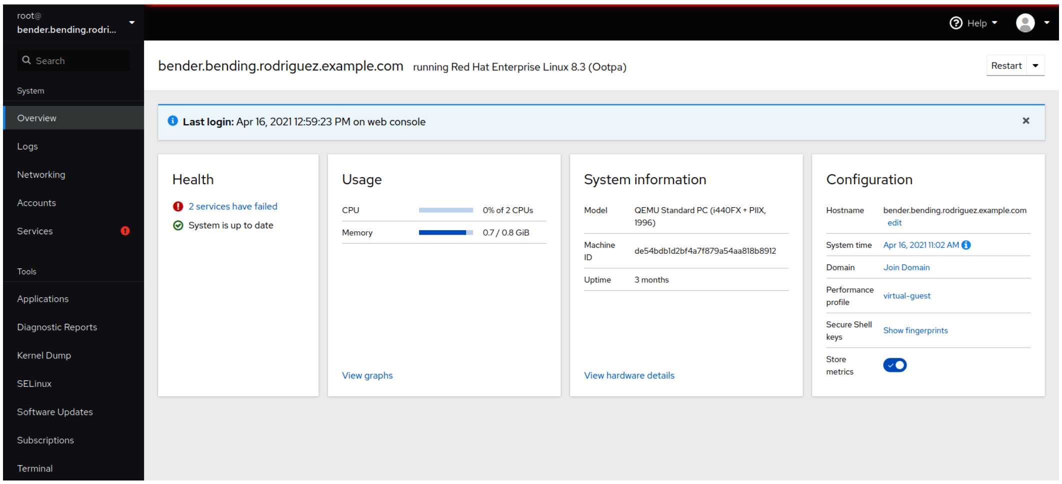


Figure 4.12 – Cockpit screen after logging in with a system dashboard

As you can see, *cockpit* includes several tabs that can be used to view the status of the system and even perform some administration tasks, such as **SELinux**, software updates, subscriptions, and more.

For example, we can check the system graphs on performance, as shown in the following screenshot:

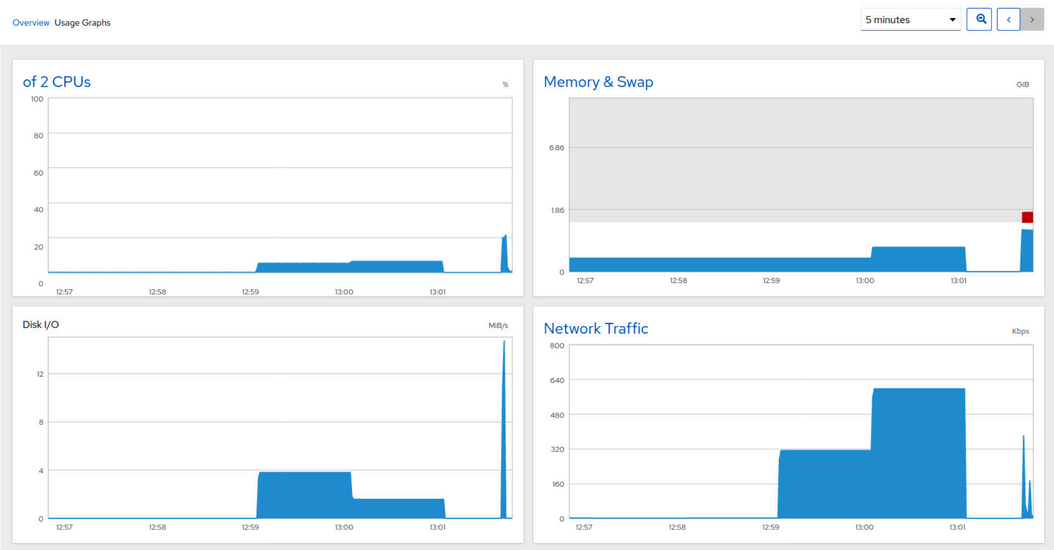


Figure 4.13 – Cockpit graphs in the dashboard for Usage Graphs

Cockpit allows us to check a service's status, package upgrade status, plus other configuration settings from a graphical interface that can also connect remotely to other systems. These can be selected from the lateral menu on the left.

There are better tools suited for large deployment monitoring and management, such as *Ansible* and *Satellite*, so it is important to get used to the tools we have for troubleshooting and simple scripts we can build. This allows us to combine what we've learned so far to quickly generate hints about things that require our attention.

With that, we have covered some of the basics of checking resource usage. Now, let's check out how to find information about the running services and errors we can review.

Finding logs, using journald, and reading log files, including log preservation and rotation

In this section, you will learn how to review a system's status via logs.

Previously in this chapter, we learned how to manage system services via *systemd*, check their status, and check their logs. Traditionally, the different daemons and system components used to create files under the `/var/log/` folder are based on the name of the daemon or service. If the service used to create several logs, it would do so inside a folder for the service (for example, **httpd** or **samba**).

The system log daemon, `rsyslogd`, has a new *systemd* partner, named `systemd-journald.service`, that also stores logs, but instead of using the traditional plain text format, it uses binary format, which can be queried via the `journalctl` command.

It's really important to get used to reading the log files as it's the basis for troubleshooting, so let's learn about general logging and how to use it.

Logs contain status information for the services that generate it. They might have some common formatting and can often be configured, but they tend to use several common elements, such as the following:

- Timestamp
- Module generating the entry
- Message

The following is an example:

```
Jan 03 22:36:47 el8-692807 sshd[50197]: Invalid user admin from
49.232.135.77 port 47694
```

In this case, we can see that someone attempted to log into our system as the `admin` user from IP address `49.232.135.77`.

We can correlate that event with additional logs, such as the ones for the login subsystem via `journalctl -u systemd-logind`. In this example, we cannot find any login for the `admin` user (this is expected as the `admin` user was not defined in this system).

Additionally, we can see the name of the host, `el8-692807`, the service generating it, `sshd`, a **PID** of `50197`, and the message that's been logged by that service.

In addition to *journalctl*, there are additional logs that we can look at when we wish to check the system's health. Let's look at an example with `/var/log/messages`:

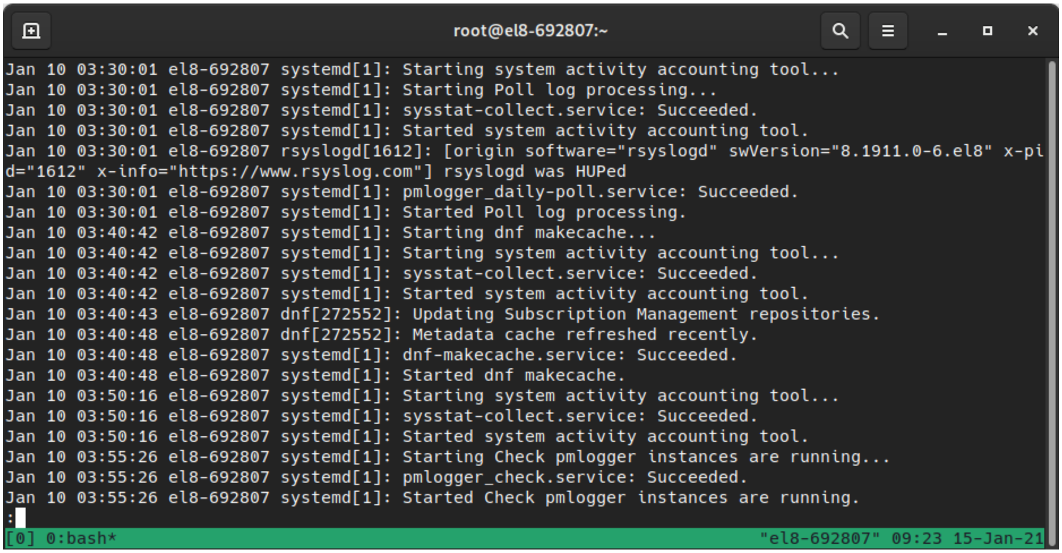


Figure 4.14 – Excerpt of /var/log/messages

In this example, we can see how the system ran some commands while following a similar output to the initial lines. For example, in the preceding example, we can see how `sysstat` has been executed every 10 minutes, as well as how the `dnf` cache has been updated.

Let's look at a list of important logs that are available in a standard system installation (note that the filenames are relative to `/var/log folder`):

- `boot.log`: Stores the messages that are emitted by the system during boot. It might contain escape codes that are used to provide colorized output.
- `audit/audit.log`: Contains the stored messages that have been generated by the Kernel audit subsystem.
- `secure`: Contains security-related messages, such as failed `sshd` login attempts.
- `dnf.log`: Logs generated by the DNF package manager, such as cache refreshes.
- `firewalld`: Output generated by the *firewalld* daemon.
- `lastlog`: This is a binary file that contains information about the last few users logging into the system (to be queried via the `last` command).
- `messages`: The default logging facility. This means that anything that is not a specific log will go here. Usually, this is the best place to start checking what happened with a system.

- `maillog`: The log for the mail subsystem. When enabled, it attempts to deliver messages. Any messages that are received will be stored here. It's common practice to configure outgoing mail from servers so that system alerts or script outputs can be delivered.
- `btmp`: Binary log for failed access to the system.
- `wtmp`: Binary log for access to the system.
- `sa/sar*`: Text logs for the *sysstat* utility (the binary ones, named *sa*, plus the day number, are converted via a *cron* job at night).

Additional log files might exist, depending on the services that have been installed, the installation method that was used, and so on. It is very important to get used to the available logs and, of course, review their contents to see how the messages are formatted, how many logs are created every day, and what kind of information they produce.

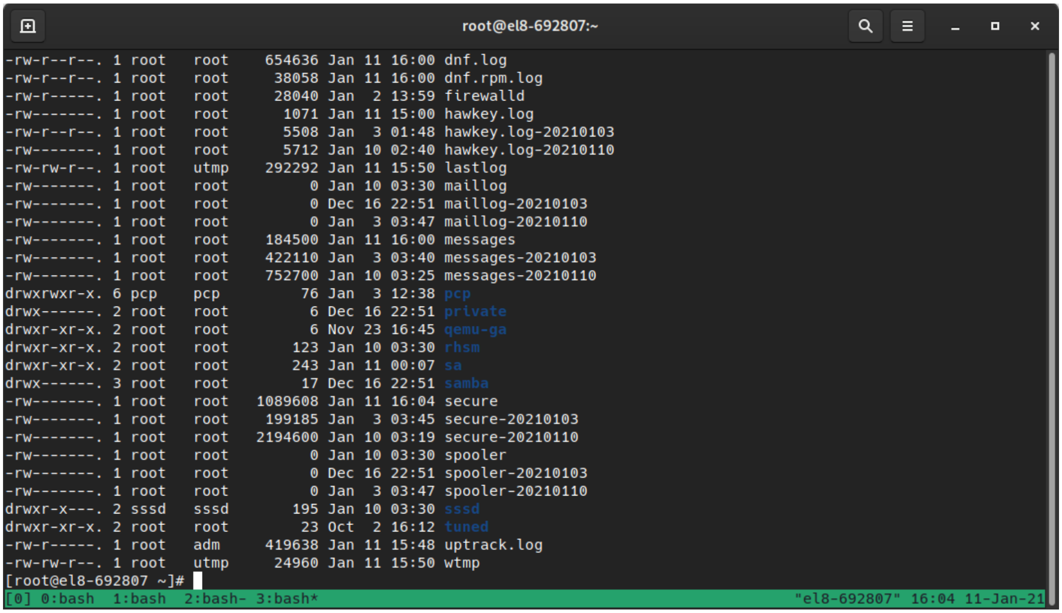
Using the information that's been logged, we will get hints on how to configure each individual daemon. This allows us to adjust the log level between being showing just errors or being more verbose about debugging issues. This means we can configure the required log rotation to avoid risking system stability because all the space has been consumed by logs.

Log rotation

During regular system operation, lots of daemons are in use, and the system itself generates the logs that are used for troubleshooting and system checks.

Some services might allow us to define the log file to write for based on the date, but usually, the standard is to log to a file named like the daemon in the `/var/log` directory; for example, `/var/log/cron`. Writing to the same file will cause the file to grow until the drive holding the logs is filled, which might not make sense as after a while (sometimes, under company-defined policies), logs are no longer useful.

The **logrotate** package provides a script with a `cron` entry that simplifies the log rotation process. It is configured via `/etc/logrotate.conf` and is executed on a daily basis, as shown here:



```
root@el8-692807:~
-rw-r--r--. 1 root   root    654636 Jan 11 16:00 dnf.log
-rw-r--r--. 1 root   root     38058 Jan 11 16:00 dnf.rpm.log
-rw-r-----. 1 root   root     28040 Jan  2 13:59 firewalld
-rw-------. 1 root   root      1071 Jan 11 15:00 hawkey.log
-rw-r--r--. 1 root   root      5508 Jan  3 01:48 hawkey.log-20210103
-rw-------. 1 root   root      5712 Jan 10 02:40 hawkey.log-20210110
-rw-rw-r--. 1 root   utmp    292292 Jan 11 15:50 lastlog
-rw-------. 1 root   root         0 Jan 10 03:30 maillog
-rw-------. 1 root   root         0 Dec 16 22:51 maillog-20210103
-rw-------. 1 root   root         0 Jan  3 03:47 maillog-20210110
-rw-------. 1 root   root    184500 Jan 11 16:00 messages
-rw-------. 1 root   root    422110 Jan  3 03:40 messages-20210103
-rw-------. 1 root   root    752700 Jan 10 03:25 messages-20210110
drwxrwxr-x. 6 pcp    pcp         76 Jan  3 12:38 pcp
drwx------. 2 root   root         6 Dec 16 22:51 private
drwxr-xr-x. 2 root   root         6 Nov 23 16:45 qemu-ga
drwxr-xr-x. 2 root   root       123 Jan 10 03:30 rhsm
drwxr-xr-x. 2 root   root       243 Jan 11 00:07 sa
drwx------. 3 root   root        17 Dec 16 22:51 samba
-rw-------. 1 root   root   1089608 Jan 11 16:04 secure
-rw-------. 1 root   root    199185 Jan  3 03:45 secure-20210103
-rw-------. 1 root   root   2194600 Jan 10 03:19 secure-20210110
-rw-------. 1 root   root         0 Jan 10 03:30 spooler
-rw-------. 1 root   root         0 Dec 16 22:51 spooler-20210103
-rw-------. 1 root   root         0 Jan  3 03:47 spooler-20210110
drwxr-x---. 2 sssd   sssd       195 Jan 10 03:30 sssd
drwxr-xr-x. 2 root   root        23 Oct  2 16:12 tuned
-rw-r-----. 1 root   adm     419638 Jan 11 15:48 uptrack.log
-rw-rw-r--. 1 root   utmp     24960 Jan 11 15:50 wtmp
[root@el8-692807 ~]#
[0] 0:bash  1:bash  2:bash- 3:bash*                    "el8-692807" 16:04 11-Jan-21
```

Figure 4.15 – Example listing of logs and rotated logs (using date extension)

If we check the contents of the configuration file, we will see that it includes some file definitions either directly there or via drop-in files in the `/etc/logrotate.d/` folder, which allows each program to drop their own requirements without it affecting others when packages are installed, removed, or updated.

Why is this important? Because, if you remember from of the tips earlier in this chapter (while speaking about disk space), if `logrotate` just deleted the files and created a new one, the actual disk space would not be freed, and the daemon writing to the log will continue to write to the file it was writing to (via the file handle). To overcome this, each definition file can define a post-rotation command. This signals the process of log rotation so that it can close and then reopen the files it uses for logging. Some programs might require a signal such as `kill -SIGHUP PID` or a special parameter on execution such as `chronyc cyclelogs`.

With these definitions, `logrotate` will be able to apply the configuration for each service and, at the same time, keep the service working in a sane state.

Configuration can also include special directives, such as the following:

- `missingok`
- `nocreate`
- `nopytruncate`
- `notifempty`

You can find out more about them (and others) on the **man page** for `logrotate.conf` (yes, some packages also include a man page for the configuration files, so try checking `man logrotate.conf` to get the full details!).

The remaining general configuration in the main file allows us to define some common directives, such as how many days of logs to keep, if we want to use the date in the file extension for the rotated log files, if we want to use compression on the rotated logs, how frequently we want to have the rotation executed, and so on.

Let's look at some examples.

The following example will rotate on a `daily` basis, keep `30` rotated logs, `compress` them, and use an extension with `date` as part of its trailing filename:

```
rotate 30
daily
compress
dateext
```

In this example, it will keep `4` logs rotated on a `weekly` basis (so 4 weeks) and will `compress` the logs, but use a sequence number for each rotated log (this means that each time a rotation happens, the sequence number is increased for the previously rotated logs too):

```
rotate 4
weekly
compress
```

One of the advantages of this approach (not using `dateext`) is that the log naming convention is predictable as we have `daemon.log` as the current one, `daemon.1.log` as the prior one, and so on. This makes it easier to script log parsing and processing.

Summary

In this chapter, we learned about `systemd` and how it takes care of booting the required system services in an optimized way. We also learned how to check a service's status, how to enable, disable, start, and stop them, and how to make the system boot into the different targets that we boot our system into.

Time synchronization was introduced as a must-have feature and it ensures our service functions properly. It also allows us to determine the status of our system clock and how to act as a clock server for our network.

We also used system tools to monitor resource usage, learned how to check the logs that are created by our system to find about the functional status of the different tools, and how to can ensure that logs are maintained properly so that older entries are discarded when they are no longer relevant.

In the next chapter, we will dive into securing the system with different users, groups, and permissions.

5
Securing Systems with Users, Groups, and Permissions

Security is a key part of managing a system, and understanding the security concepts in order to provide the right access to the right resource to the right user, or group of users, is required for any system administrator.

In this chapter, we will review the basics of security in **Red Hat Enterprise Linux** (**RHEL**). We will add new users to a system and change their attributes. We will also add a user to a group, and reviewing groups before making changes will be seen in this chapter. We shall review how to handle user passwords and changing the age requirements for them, locking and/or restricting user access. We will use `sudo` as a way to assign admin privileges to different users in the system (and even disable the root account). We will also take a deeper look into file permissions and how to change them, using an extended capability to enable commands to be run with a different user or group or simplify group collaboration in directories.

We will cover the following topics:

- Creating, modifying, and deleting local user accounts and groups
- Managing groups and reviewing assignments
- Adjusting password policies
- Configuring sudo access for administrative tasks
- Checking, reviewing, and modifying file permissions
- Using special permissions

Let's get started in the world of permissions and security with user accounts and groups.

Creating, modifying, and deleting local user accounts and groups

One of the first tasks that a system administrator has to do when preparing a system to be accessed by users is to create new user accounts for the people accessing the system. In this section, we will review how local accounts are created and deleted, as well as how they are assigned to groups.

The first step is to create a new user account in the system. That is done by using the `useradd` command. Let's add `user01` to the system by running the following:

```
[root@rhel8 ~]# useradd user01
[root@rhel8 ~]# grep user01 /etc/passwd
user01:x:1001:1001::/home/user01:/bin/bash
[root@rhel8 ~]# id user01
uid=1001(user01) gid=1001(user01) groups=1001(user01)
```

With that, the user is created.

> **Important note**
> To be able to add users, we need administrative privileges. In the current configuration, we do that by running the commands as `root`.

The account is created using the default options configured in the system, such as the following:

- **No password assigned**: The new user will not be able to log in using a password. However, we can switch to that account by using `su` as `root`. We will see how to add a password to the user next.
- **User ID (UID)**: The first number over 999 available. In the command we ran before, for `user01`, the UID is `1001`.
- **Group ID (GID)**: The same number as the UID. In this case, the GID is `1001`.
- **Description**: No description is added when creating the user. This field is left empty.
- **Home**: A `home` directory is created in `/home/$USER`, in this case `/home/user01`. This will be default and main directory for the user and is where their personal preferences and files will be stored. Initial contents are copied from `/etc/skel`.
- **Shell**: The default shell is `bash`.

> **Tip**
> The default options applied when a new user is created are defined in the `/etc/default/useradd` file.

Once the user is created, we can add (or change) the password by running, as `root`, the command `passwd` followed by the username to change it:

```
[root@rhel8 ~]# passwd user01
Changing password for user user01.
New password: redhat
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: redhat
passwd: all authentication tokens updated successfully
```

Now the user has the new password assigned. Note two things:

- User `root` can change the password to any user without knowing the previous one (a full password reset). This is useful when a user comes back from their holidays and doesn't remember their password.
- In the example, we show the password assigned, `redhat`, but that is not shown on the screen. The password is too simple and does not meet the default complexity criteria, however, as `root` we can still assign it.

Let's check the new user with the `id` command we learned before:

```
[root@rhel8 ~]# id user01
uid=1001(user01) gid=1001(user01) groups=1001(user01)
```

After the steps taken in this section, we now have the user in the system and ready to be used. The main options we could have used to customize the user creation with `useradd` are the following:

- `-u` or `--uid`: Assign a specific UID to the user.
- `-g` or `--gid`: Assign a main group to the user. It can be specified by number (GID) or by name. The group needs to be created first.
- `-G` or `--groups`: Make the user part of other groups by providing a comma-separated list of them.
- `-c` or `--comment`: Provide a description for the user, specified between quotes if you want to use spaces.
- `-d` or `--home-dir`: Define the home directory for the user.
- `-s` or `--shell`: Assign a custom shell to the user.
- `-p` or `--password`: A way to provide a password to the user. The password should be already encrypted to use this method. It is recommended *not* to use this option as there are ways to capture the encrypted password. Please use `passwd` instead.
- `-r` or `--system`: To create a system account instead of a user account.

What if we need to change any of the user's properties, such as, for example, the description? The tool for that is `usermod`. Let's modify the description to `user01`:

```
[root@rhel8 ~]# usermod -c "User 01" user01
[root@rhel8 ~]# grep user01 /etc/passwd
user01:x:1001:1001:User 01:/home/user01:/bin/bash
```

The `usermod` command uses the same options as `useradd`. It will be easy to customize your current users now.

Let's create `user02` as an example of how to use the options:

```
[root@rhel8 ~]# useradd --uid 1002 --groups wheel \
--comment "User 02" --home-dir /home/user02 \
--shell /bin/bash user02
[root@rhel8 ~]# grep user02 /etc/passwd
```

```
user02:x:1002:1002:User 02:/home/user02:/bin/bash
[root@rhel8 ~]# id user02
uid=1002(user02) gid=1002(user02) groups=1002(user02),10(wheel)
```

> **Tip**
>
> When the command line is too long, the character \ can be added, then press *Enter* and continue the command on a new line.

Now we know how to create a user, but we may need to create a group too and add our users to it. Let's create the `finance` group, with the `groupadd` command:

```
[root@rhel8 ~]# groupadd finance
[root@rhel8 ~]# grep finance /etc/group
finance:x:1003:
```

We can add the `user01` and `user02` users to the `finance` group:

```
[root@rhel8 ~]# usermod -aG finance user01
[root@rhel8 ~]# usermod -aG finance user02
[root@rhel8 ~]# grep finance /etc/group
finance:x:1003:user01,user02
```

> **Important note**
>
> We are using the `-aG` option to add the user to the group, instead of modifying the groups the user belongs to.

Once we know how to create users and groups, let's check how to delete them with the `userdel` command:

```
[root@rhel8 ~]# userdel user01
[root@rhel8 ~]# grep user01 /etc/passwd
[root@rhel8 ~]# id user01
id: 'user01': no such user
[root@rhel8 ~]# grep user02 /etc/passwd
user02:x:1002:1002:User 02:/home/user02:/bin/bash
[root@rhel8 ~]# id user02
uid=1002(user02) gid=1002(user02)
groups=1002(user02),10(wheel),1003(finance)
[root@rhel8 ~]# ls /home/
```

```
user  user01  user02
[root@rhel8 ~]# rm -rf /home/user01/
```

As you can see, we needed to manually delete the `home` directory. This way of removing a user is good if we want to keep its data for future use.

To fully remove a user, we shall apply the option `-r`. Let' try it with `user02`:

```
[root@rhel8 ~]# userdel -r user02
[root@rhel8 ~]# ls /home/
user  user01
[root@rhel8 ~]# grep user02 /etc/passwd
[root@rhel8 ~]# id user02
id: 'user02': no such user
```

Now let's remove the `finance` group with the `groupdel` command:

```
[root@rhel8 ~]# groupdel finance
[root@rhel8 ~]# grep finance /etc/group
```

As we have seen, it's simple and easy to create users and groups in RHEL and make simple assignments. In the next section, let's check in more depth how to manage groups and assignments to them.

Managing groups and reviewing assignments

We have seen how to create a group with `groupadd` and delete it with `groupdel`. Let's see how to modify a created group with `groupmod`.

Let's create a group to work with. We will create the misspelled `acounting` group by running the following:

```
[root@rhel8 ~]# groupadd -g 1099 acounting
[root@rhel8 ~]# tail -n1 /etc/group
acounting:x:1099:
```

You see we made a mistake in the name by not spelling it `accounting`. We may even have added some user accounts to it, and we need to modify it. We can do so using `groupmod` and running the following:

```
[root@rhel8 ~]# groupmod -n accounting acounting
[root@rhel8 ~]# tail -n1 /etc/group
accounting:x:1099:
```

Now we've seen how we modify the group name. We can modify not just the name but the GID by using the `-g` option:

```
[root@rhel8 ~]# groupmod -g 1111 accounting
[root@rhel8 ~]# tail -n1 /etc/group
accounting:x:1111:
```

We can see which groups are assigned to a user by running the `groups` command%:

```
[root@rhel8 ~]# groups user
user : user wheel
```

With this, we are ready to manage groups and users in a Linux system. Let's move on to password policies.

Adjusting password policies

As was mentioned in *Chapter 3*, *Basic Commands and Simple Shell Scripts*, users are stored in the `/etc/passwd` file while the encrypted passwords, or **password hashes**, are stored in the `/etc/shadow` file.

> **Tip**
>
> A hashing algorithm is made so that it generates a precise string of characters, or a hash, from a provided piece of data (that is, a file or a word). It does it in a way so that it will always generate the same hash from the same original data, but the original data is almost impossible to recreate from the hash. That's why they are used to store passwords or verify the integrity of a downloaded file.

Let's take a look at one example by running `grep` user as `root` against `/etc/shadow`:

```
user:$6$tOT/cvZ4PWRcl8XX$0v3.ADE/
ibzlUGbDLer0ZYaMPNRJ5gK17LeKnoMfKK9
.nFz8grN3IafmHvoHPuh3XrU81nJu0.
is5znztB64Y/:18650:0:99999:7:3:19113:
```

As with the password file, the data stored in `/etc/shadow` has an entry per line and the fields are separated by colons (`:`).

- `user`: Account name. It should be the same one as in `/etc/passwd`.
- `$6$tOT/cvZ4PWRcl8XX$0v3.ADE/ibzlUGbDLer0ZYaMPNRJ5gK17LeKnoMfKK9.nFz8grN3IafmHvoHPuh3XrU81nJu0.is5znztB64Y/`: Password hash. It contains three parts separated by `$`:
    - `$6`: Algorithm used to encrypt the file. In this case, the value `6` indicates that it is SHA-512. The number `1` is for the old, now insecure, MD5 algorithm.
    - `$tOT/cvZ4PWRcl8XX`: The password **salt**. This token is used to improve password encryption.
    - `$0v3.ADE/ibzlUGbDLer0ZYaMPNRJ5gK17LeKnoMfKK9.nFz8grN3IafmHvoHPuh3XrU81nJu0.is5znztB64Y/`: Encrypted password hash. Using the salt and the SHA-512 algorithm, this token is created. When the user validates, the process is run again and if the same hash is generated, the password is validated and access is granted.
- `18650`: The time and date when the password was last changed. The format is the number of days since 1970-01-01 00:00 UTC (this date is also known as **the epoch**).
- `0`: Minimum number of days until the user can change the password again.
- `99999`: Maximum number of days until the user has to change the password again. If empty, it won't expire.
- `7`: Number of days the user will be warned that the password is about to expire.
- `3`: Number of days the user can still log in even when the password has expired.
- `19113`: Date on which the password should expire. If empty, it won't expire on a specific date.
- `<empty>`: The last colon is left to allow us to add new fields easily.

> **Tip**
>
> To convert the `date` field to a human-readable date, you can run the following command: `date -d '1970-01-01 UTC + 18650 days'`.

How do we change the expiration dates for passwords? The tool to do so is `chage`, for **change age**. Let's first review the options that can be used in the same order as they are stored in `/etc/shadow`:

- `-d` or `--lastday`: The time and date when the password was last changed. The format for it is `YYYY-MM-DD`.
- `-m` or `--mindays`: Minimum number of days until the user can change the password again.
- `-W` or `--warndays`: Number of days the user will be warned that the password is about to expire.
- `-I` or `--inactive`: Number of days, once the password has expired, that will have to pass before the account is locked.
- `-E` or `--expiredate`: Date after which the user's account will be locked. The date should be expressed in the format `YYYY-MM-DD`.

Let's try it. First, we create the `usertest` account:

```
[root@rhel8 ~]# adduser usertest
[root@rhel8 ~]# grep usertest /etc/shadow
usertest:!!:18651:0:99999:7:::
```

> **Important note**
>
> The tools `adduser` and `useradd` are the same tool in RHEL 8. Feel free to type it the way you feel most comfortable with.

You'll notice in the previous example, from the two exclamation marks, `!!`, in bold that the password is not set, and we are using the defaults. Let's change the password and check the difference. Use any password you like:

```
[root@rhel8 ~]# passwd usertest
Changing password for user usertest.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

```
[root@rhel8 ~]# grep usertest /etc/shadow
usertest:$6$4PEVPj7M4GD8CH.4$VqiYY.
IXetwZA/g54bFP1ZJwQ/yc6bnaFauHGA1 1eFzsGh/
uFbJwxZCQTFHIASuamBz.27gb4ZpywwOA840eI.:18651:0:99999:7:::
```

The password hash is created and the date for the last change is just kept the same as the current date. Let's establish some options:

```
[root@rhel8 ~]# chage --mindays 0 --warndays 7 --inactive 3
--expiredate 2030-01-01 usertest
[root@rhel8 ~]# grep usertest /etc/shadow
usertest:$6$4PEVPj7M4GD8CH.4$VqiYY.IXetwZA/g54bFP1ZJwQ/
yc6bnaFauHGA1 1eFzsGh/uFbJwxZCQTFHIASuamBz.27gb4ZpywwOA
840eI.:18651:0:99999:7:3:21915:
[root@rhel8 ~]# date -d '1970-01-01 UTC + 21915 days'
mar ene  1 01:00:00 CET 2030
```

Please notice the changes in the `/etc/shadow` file corresponding to the values specified for `chage`. We can check the changes with the option `-l` of `chage`:

```
[root@rhel8 ~]# chage -l usertest
Last password change                                    : ene 24, 2021
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : ene 01, 2030
Minimum number of days between password change          : 0
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
```

To change the default values, we shall edit `/etc/login.defs`. Let's check the section for the most common changes:

```
# Password aging controls:
#
#    PASS_MAX_DAYS    Maximum number of days a password may be
used.
#    PASS_MIN_DAYS    Minimum number of days allowed between
password changes.
#    PASS_MIN_LEN    Minimum acceptable password length.
#    PASS_WARN_AGE    Number of days warning given before a
password expires.
```

```
#
PASS_MAX_DAYS     99999
PASS_MIN_DAYS     0
PASS_MIN_LEN      5
PASS_WARN_AGE     7
```

Please take some minutes to review the options in `/etc/login.defs`.

Now, we could have a situation in which a user has left the company. How can we lock the account so the user cannot access the system? The `usermod` command has the `-L` option, for **lock**, to do so. Let's try it. First, let's log into the system:

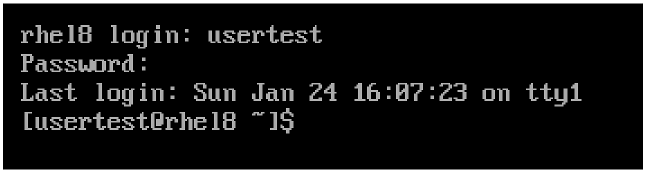


Figure 5.1 – User account usertest logging into the system

Now let's lock the account:

```
[root@rhel8 ~]# usermod -L usertest
[root@rhel8 ~]# grep usertest /etc/shadow
usertest:!$6$4PEVPj7M4GD8CH.4$VqiYY.IXetwZA/g54bFP1ZJwQ/
yc6bnaFauHGA 11eFzsGh/uFbJwxZCQTFHIASuamBz.27gb4ZpywwOA840eI.
:18651:0:99999:7:3:21915:
```

Notice that there is a `!` character added before the password hash. This is the mechanism used to lock it. Let's try to log in again:

Figure 5.2 – User account usertest not being able to log into the system

The account can be unlocked by using the `-U` option:

```
[root@rhel8 ~]# usermod -U usertest
[root@rhel8 ~]# grep usertest /etc/shadow
usertest:$6$4PEVPj7M4GD8CH.4$VqiYY.IXetwZA/g54bFP1ZJwQ/
yc6bnaFauHGA1 1eFzsGh/uFbJwxZCQTFHIASuamBz.27gb4ZpywwOA840eI.
:18651:0:99999:7:3:21915:
```

Now you can see that the `!` character is removed. Feel free to try logging in again.

> **Important note**
> To fully lock the account from access, not just from logging in with a password (there are other mechanisms), we should set the expiry date to `1`.

Another common use case is when you want users to access the system, such as having a network shared directory (that is, via NFS or CIFS, as explained in *Chapter 12*, *Managing Local Storage and Filesystems*) but you do not want them to be able to run commands in the system. For that, we can use a very special shell, the `nologin` shell. Let's assign that shell to the `usertest` user account using `usermod`:

```
[root@rhel8 ~]# usermod -s /sbin/nologin usertest
[root@rhel8 ~]# grep usertest /etc/passwd
usertest:x:1001:1001::/home/usertest:/sbin/nologin
[root@rhel8 ~]# su - usertest
Last login: sun jan 24 16:18:07 CET 2021 on pts/0
This account is currently not available.
[root@rhel8 ~]# usermod -s /bin/bash usertest
[root@rhel8 ~]# su - usertest
Last login: sun jan 24 16:18:15 CET 2021 on pts/0
[usertest@rhel8 ~]$
```

Note that we are reviewing the changes in `/etc/passwd` this time as it is where the modification is applied.

As you can see, it's easy to set the values for password aging for any user, lock them, or restrict access to the system. Let's move on to more administrative tasks and how to delegate admin access.

Configuring sudo access for administrative tasks

There is a way to delegate administrative access to users in RHEL, and it is done so with a tool called **sudo**, which stands for **Super User Do**.

It not only allows you to grant full administrative privileges to users or groups but also be very granular on the privileged commands that some users may be able to execute.

Let's start by understanding the default configuration and how to change it.

Understanding sudo configuration

The tool has its main configuration file in `/etc/sudoers` and includes this part in the default configuration:

```
root ALL=(ALL)  ALL
%wheel    ALL=(ALL)  ALL
## Read drop-in files from /etc/sudoers.d (the # here does not mean a comment)
#includedir /etc/sudoers.d
```

Let's analyze the lines one by one to understand what they do.

The first line enables the `root` user to use `sudo` for any command that they want to run:

```
root ALL=(ALL)  ALL
```

The second line enables the user in the `wheel` group to use `sudo` for any command that they want to run. We will explain the details of the syntax later:

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

> **Important note**
>
> Please do not disable the `wheel` group directive unless there is an important reason to do so. This behavior is expected by other programs to be available, and disabling it may cause some problems.

The third line, and all the lines starting with `#`, are considered comments, and they are intended only to add descriptive content with no effect on the final configuration:

```
 ## Read drop-in files from /etc/sudoers.d (the # here does not mean a comment)
```

The fourth line is the only exception to the previous rule. This line enables the directory `/etc/sudoers.d` as a source for configuration files. We can drop a file in that folder, and it will be used by `sudo`:

```
#includedir /etc/sudoers.d
```

The exception to this last rule is files that end with ~ or contain a `.` (dot) character.

As you have seen, the default configuration enables `root` and the members of the `wheel` group to run any command as an administrator using `sudo`.

The easiest way to use it is to add a user to the `wheel` group to grant that user full admin privileges. An example of how to modify the `usertest` account to make it an admin account is as follows:

```
[root@rhel8 ~]# usermod -aG wheel usertest
[root@rhel8 ~]# groups usertest
usertest : usertest wheel
```

> **Important note**
>
> For cloud instances, the account root does not have a valid password assigned. To be able to manage the mentioned cloud instance, in some clouds such as **Amazon Web Services** (**AWS**), a user is created by default and added to the `wheel` group. In the case of AWS, the default user account is `ec2-user`. In other clouds, a custom user is also created and also added to the `wheel` group.

To edit the `/etc/sudoers` file, as happens with other sensitive files, there is a tool that helps not only ensure that two admins are not editing it simultaneously but also that the syntax is correct. In this case, the tool to edit it is `visudo`.

Using sudo to run admin commands

We will use the `user` account in these examples. As you may remember, in *Chapter 1*, *Installing RHEL8*, we enabled the checkbox in which we requested the account to be the administrator. Under the hood, the account was added to the `wheel` group, so we can start using `sudo` to run admin commands.

Let's log in with the `user` account and try to run an administrative command such as `adduser`:

```
[root@rhel8 ~]# su - user
Last login: dom ene 24 19:40:31 CET 2021 on pts/0
[user@rhel8 ~]$ adduser john
adduser: Permission denied.
adduser: cannot lock /etc/passwd; try again later.
```

As you can see, we receive a `Permission denied` error message. To be able to run it with `sudo`, we only need to add it to the beginning of the command line:

```
[user@rhel8 ~]$ sudo adduser john

We trust you have received the usual lecture from the local
System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for user:
[user@rhel8 ~]$ id john
uid=1002(john) gid=1002(john) groups=1002(john)
```

In this case, we see that we have been shown a warning message that is shown the first time we run `sudo` successfully. Then we are asked for *our own password* – not the admin password as there may not even be one, but the one we have for the user running `sudo`. Once the password is correctly typed, the command is run and registered in the system journal:

```
jan 24 19:44:26 rhel8.example.com sudo[2879]: user : TTY=pts/0
; PWD=/home/user ; USER=root ; COMMAND=/sbin/adduser john
```

Important note

Once you have run `sudo` successfully, it will remember that validation for 15 minutes (as the default behavior). This is done so you don't have to type your password again and again if you need to run more than one administrative command in a session. To increase it to 30 minutes, we can add the following line using `visudo`: `Defaults:USER timestamp_timeout=30`.

Sometimes you want to have an interactive session so that there is no need to type `sudo` again and again. For that, the `-i` option is really useful. Let's try it:

```
[user@rhel8 ~]$ sudo -i
[sudo] password for user:
[root@rhel8 ~]#
```

Let's now move on to customizing the configuration of sudo in the sudoers file.

Configuring sudoers

We have seen the details of the default /etc/sudoers file in the previous section. Let's see a couple of examples of how to make a more granular configuration.

Let's start by making sudo run admin commands without requesting a password for the users in the wheel group. We can run visudo and make the line that starts with %wheel look like this:

```
%wheel          ALL=(ALL)       NOPASSWD: ALL
```

Save it. Note that there is a commented line in the configuration file with that configuration. Now let's try it:

```
[user@rhel8 ~]$ sudo adduser ellen
[user@rhel8 ~]$ id ellen
uid=1003(ellen) gid=1003(ellen) groups=1003(ellen)
```

We can now create a file, with your favorite editor, to make the new user account, ellen, able to run admin commands. Let's create the /etc/sudoers.d/ellen file with this content:

```
ellen ALL=(ALL)  ALL
```

With this, we are using the /etc/sudoers.d directory to extend the sudo configuration.

We will review the detailed configuration of sudoers here despite it not being part of the RHCSA exam. As you can see, there are three fields, separated by spaces or tabs, to define policies in the configuration files. Let's review them:

- The first field is to specify who is affected by the policy:
  - We can add users by simply putting the username in the first field.
  - We can add groups by using the % character before the name of the group in the first field.
- The second field is for where the policy applies:
  - We have so far used ALL=(ALL) to specify everything.
  - In the first part of this field, we can define a group of computers to be run such as SERVERS=10.0.0.0/255.255.255.0.

- In the second part, we can specify commands such as `NETWORK=/usr/sbin/ip`.
- Between parentheses is the user account that can be used to run the command.

- The third field is to specify which commands will use the password and which won't.

The syntax goes like this:

```
user  hosts = (run-as) commands
```

Let's see an example:

```
Runas_AliasDB = oracle
Host_Alias SERVERS=10.0.0.0/255.255.255.0
Cmnd_Alias NETWORK=/ust/sbin/ip

pete  SERVERS=NETWORK
julia SERVERS=(DB)ALL
```

We have already seen how to provide administrative access to users in RHEL, and even how to do it in a very granular manner. Let's move on now to the section on working with file permissions.

Checking, reviewing, and modifying file permissions

We have learned so far how to create users and groups, and even provide administrative capabilities to them. It's time now to see how permissions work at the files and directories level.

As you'll remember, in *Chapter 3*, *Basic Commands and Simple Shell Scripts*, we already saw how to see the permissions that are applied to a file. Let's review them now and dive deeper.

Let's get the permissions info for some example files by listing it with the `-l`, for long, option. Remember to run this as the `root` user (or using `sudo`):

```
[root@rhel8 ~]# ls -l /usr/bin/bash
-rwxr-xr-x. 1 root root 1150704 jun 23  2020 /usr/bin/bash
[root@rhel8 ~]# ls -l /etc/passwd
```

```
-rw-r--r--. 1 root root 1324 ene 24 21:35 /etc/passwd
[root@rhel8 ~]# ls -l /etc/shadow
----------. 1 root root 1008 ene 24 21:35 /etc/shadow
[root@rhel8 ~]# ls -ld /tmp
drwxrwxrwt. 8 root root 172 ene 25 17:35 /tmp
```

Remember that, in Linux, *everything is a file*.

Now let's review the five different blocks of information that the permissions include by using the ones for `/usr/bin/bash`:

```
-rwxr-xr-x.
```

The blocks are as follows:

Block 1	Block 2	Block 3	Block 4	Block 5
-	rwx	r-x	r-x	.

Let's review them again, as they are very important.

Block 1 is for the special permissions that the file may have. If it is a regular file and has no special permissions (as in this case), it will appear as `-`:

- Directories will appear with `d`.
- Links, usually symbolic links, will appear with an `l`.
- Special permissions to run a file as a different user or group, called **setuid** or **setgid**, will appear as `s`.
- A special permission for directories, so that the owner can only remove or rename the file, called the **sticky bit** will appear as `t`.

Block 2 is permissions for the *user* owning the file, and consists of three characters:

- The first one, `r`, is the read permission assigned.
- The second one, `w`, is the write permission assigned.
- The third one, `x`, is the executable permission. (Note that the executable permission for directories means being able to enter them.)

Block 3 is permissions for the *group*. It consists of the same three characters for read, write, and execute (`rwx`). In this case, write is missing.

Block 4 is the permissions for *others*. It also consists of the same three characters for read, write, and execute (`rwx`) as before. As in the previous block, write is missing.

Block 5 indicates that there is an **SELinux** context applied to the file. More on this topic in *Chapter 10*, *Keeping Your System Hardened with SELinux*.

To change permissions for a file, we will use the `chmod` command.

First, let's create a file:

```
[root@rhel8 ~]# touch file.txt
[root@rhel8 ~]# ls -l file.txt
-rw-r--r--. 1 root root 0 ene 27 18:30 file.txt
```

As you can see, the file is created with your username as the owner, your main group as the group, and a default set of permissions. The default set of permissions are defined by `umask`, and in RHEL, the defaults for newly created file permissions are as follows:

- **User**: Read and write
- **Group**: Read
- **Others**: Read

To change permissions using `chmod`, we specify the changes with three characters:

- The first one, which determines whom the change affects:

  - `u`: User

  - `g`: Group

  - `o`: Others
- The second one to add or remove permissions:

  - `+`: Add

  - `-`: Remove
- The third one, which determines the permission to be changed:

  - `r`: Read

  - `w`: Write

  - `x`: Execute

So, to add write permissions to the group, we can run the following:

```
[root@rhel8 ~]# chmod g+w file.txt
[root@rhel8 ~]# ls -l file.txt
-rw-rw-r--. 1 root root 0 ene 27 18:30 file.txt
```

And to remove read permissions from others, we run the following:

```
[root@rhel8 ~]# chmod o-r file.txt
[root@rhel8 ~]# ls -l file.txt
-rw-rw----. 1 root root 0 ene 27 18:30 file.txt
```

The permissions are stored in four octal digits. This means that special permissions are stored in a number from 0 to 7, the same way user, group, and other permissions are stored, each one of them with a number from 0 to 7.

Some examples are shown as follows:

Alphabetic	Octal	Description
`-rwxr-xr--`	0754	User: read, write, execute. Group: read, execute. Others: read.
`-rw-r-----`	0640	User: read, write. Group: read. Others: nothing.
`-r--------`	0400	User: read Group: nothing Others: nothing.

How does it work? We assign a number (power of 2) for each permission:

- **Nothing**: 0
- **Execute**: 2^0 = 1
- **Write**: 2^1 = 2
- **Read**: 2^2 = 4

We add them:

```
rwx = 4 + 2 + 1 = 7
rw- = 4 + 2 = 6
r-x = 4 + 1 = 5
r-- = 4
--- = 0
```

This is how we can assign permissions using numbers. Now let's try it:

```
[root@rhel8 ~]# chmod 0755 file.txt
[root@rhel8 ~]# ls -l file.txt
-rwxr-xr-x. 1 root root 0 ene 27 18:30 file.txt
[root@rhel8 ~]# chmod 0640 file.txt
[root@rhel8 ~]# ls -l file.txt
-rw-r-----. 1 root root 0 ene 27 18:30 file.txt
[root@rhel8 ~]# chmod 0600 file.txt
[root@rhel8 ~]# ls -l file.txt
-rw-------. 1 root root 0 ene 27 18:30 file.txt
```

As we said before, the default configuration of permissions is set by `umask`. We can very easily see the value:

```
[root@rhel8 ~]# umask
0022
```

All the newly created files have `execute` permissions removed (`1`).

With this `umask`, `0022`, the one provided by default in RHEL, we will have `write` permissions for `group` and `others` also removed (`2`).

Even when it is not recommended to change `umask`, we could give it a try to learn how it works. Let's start by using the most permissive `umask`, `0000`, to see how all `read` and `write` permissions are assigned to newly created files:

```
[root@rhel8 ~]# umask 0000
[root@rhel8 ~]# touch file2.txt
[root@rhel8 ~]# ls -l file2.txt
-rw-rw-rw-. 1 root root 0 ene 27 18:33 file2.txt
```

Now let's use the more restrictive `umask` for `group` and `others` permissions:

```
[root@rhel8 ~]# umask 0066
[root@rhel8 ~]# touch file3.txt
[root@rhel8 ~]# ls -l file3.txt
-rw-------. 1 root root 0 ene 27 18:33 file3.txt
```

If we try a higher number, it won't work and will return an error:

```
[root@rhel8 ~]# umask 0088
-bash: umask: 0088: octal number out of range
```

You can see that the effect of `0066` and `0077` is the same:

```
[root@rhel8 ~]# umask 0077
[root@rhel8 ~]# touch file4.txt
[root@rhel8 ~]# ls -l file4.txt
-rw-------. 1 root root 0 ene 27 18:35 file4.txt
```

Let's re-establish `umask` in our session, to the defaults, to continue practicing:

```
[root@rhel8 ~]# umask 0022
```

Now we may find ourselves with the need to create a directory for a specific user or group, or to change the owner of a file. To be able to change the ownership of a file or directory, the `chown` or `chgrp` tools are used. Let's see how it works. Let's move to `/var/tmp` and create the folders for `finance` and `accounting`:

```
[root@rhel8 ~]# cd /var/tmp/
[root@rhel8 tmp]# mkdir finance
[root@rhel8 tmp]# mkdir accounting
[root@rhel8 tmp]# ls -l
total 0
drwxr-xr-x. 2 root root 6 ene 27 19:35 accounting
drwxr-xr-x. 2 root root 6 ene 27 19:35 finance
```

Now let's create the groups for `finance` and `accounting`:

```
[root@rhel8 tmp]# groupadd finance
[root@rhel8 tmp]# groupadd accounting
groupadd: group 'accounting' already exists
```

In this example, the `accounting` group was already created. Let's change the group for each directory with `chgrp`:

```
[root@rhel8 tmp]# chgrp accounting accounting/
[root@rhel8 tmp]# chgrp finance finance/
[root@rhel8 tmp]# ls -l
```

```
total 0
drwxr-xr-x. 2 root accounting 6 ene 27 19:35 accounting
drwxr-xr-x. 2 root finance    6 ene 27 19:35 finance
```

Now we create users for `sonia` and `matilde`, and assign them to `finance` and `accounting` respectively:

```
[root@rhel8 tmp]# adduser sonia
[root@rhel8 tmp]# adduser matilde
[root@rhel8 tmp]# usermod -aG finance sonia
[root@rhel8 tmp]# usermod -aG accounting matilde
[root@rhel8 tmp]# groups sonia
sonia : sonia finance
[root@rhel8 tmp]# groups matilde
matilde : matilde accounting
```

Now we can create a personal folder for each under their group folder:

```
[root@rhel8 tmp]# cd finance/
[root@rhel8 finance]# mkdir personal_sonia
[root@rhel8 finance]# chown sonia personal_sonia
[root@rhel8 finance]# ls -l
total 0
drwxr-xr-x. 2 sonia root 6 ene 27 19:44 personal_sonia
[root@rhel8 finance]# chgrp sonia personal_sonia/
[root@rhel8 finance]# ls -l
total 0
drwxr-xr-x. 2 sonia sonia 6 ene 27 19:44 personal_sonia
```

There is a way to specify a user and group to `chown`, using the `:` separator. Let's use it with `matilde`:

```
[root@rhel8 tmp]# cd ../accounting
[root@rhel8 accounting]# mkdir personal_matilde
[root@rhel8 accounting]# chown matilde:matilde \
personal_matilde
[root@rhel8 accounting]# ls -l
total 0
drwxr-xr-x. 2 matilde matilde 6 ene 27 19:46 personal_matilde
```

If we want to change the permissions for a full branch, we can use `chown` with the `-R`, for recursive, option. Let's copy a branch and change its permissions:

```
[root@rhel8 accounting]# cp -rv /usr/share/doc/audit personal_
matilde/
'/usr/share/doc/audit' -> 'personal_matilde/audit'
'/usr/share/doc/audit/ChangeLog' -> 'personal_matilde/audit/
ChangeLog'
'/usr/share/doc/audit/README' -> 'personal_matilde/audit/
README'
'/usr/share/doc/audit/auditd.cron' -> 'personal_matilde/audit/
auditd.cron'
[root@rhel8 accounting]# chown -R matilde:matilde \
personal_matilde/audit
[root@rhel8 accounting]# ls -l personal_matilde/audit/
total 20
-rw-r--r--. 1 matilde matilde  271 ene 28 04:56 auditd.cron
-rw-r--r--. 1 matilde matilde 8006 ene 28 04:56 ChangeLog
-rw-r--r--. 1 matilde matilde 4953 ene 28 04:56 README
```

With this, we have a good understanding of permissions in RHEL, their default behaviors, and how to work with them.

Let's move on to some more advanced topics about permissions.

Using special permissions

As we've seen in the previous section, there are special permissions that could be applied to files and directories. Let's start by reviewing Set-UID (or **suid**) and Set-GUID (or **sgid**).

Understanding and applying Set-UID

Let's review how Set-UID applies to files and directories:

- **Set-UID permission applied to a file**: When applied to an executable file, this file will run as if the owner of the file was running it, applying the permissions.
- **Set-UID permission applied to a directory**: No effect.

Let's check a file with Set-UID:

```
[root@rhel8 ~]# ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 33544 dic 13  2019 /usr/bin/passwd
```

The `passwd` command requires `root` permissions to change hashes in the `/etc/shadow` file.

To apply this permission, we can use the `chmod` command, applying `u+s` permissions:

```
[root@rhel8 ~]# touch testsuid
[root@rhel8 ~]# ls -l testsuid
-rw-r--r--. 1 root root 0 ene 28 05:16 testsuid
[root@rhel8 ~]# chmod u+s testsuid
[root@rhel8 ~]# ls -l testsuid
-rwsr--r--. 1 root root 0 ene 28 05:16 testsuid
```

> **Tip**
>
> Be very careful when assigning `suid` to files as `root`. If you leave write permissions on the file, any user will be able to change the content and execute anything as `root`.

Understanding and applying Set-GID

Let's review how Set-GID applies to files and directories:

- **Set-GID permission applied to a file**: When applied to an executable file, this file will run with the group permissions of the file.
- **Set-GID permission applied to a directory**: New files created in that directory will have the group of the directory applied to them.

Let's check a file with Set-GID:

```
[root@rhel8 ~]# ls -l /usr/bin/write
-rwxr-sr-x. 1 root tty 21232 jun 26  2020 /usr/bin/write
```

We can try applying the permission to a file with `chmod` using `g+s`:

```
[root@rhel8 ~]# touch testgid
[root@rhel8 ~]# chmod g+s testgid
[root@rhel8 ~]# ls -l testgid
-rw-r-sr--. 1 root root 0 ene 28 05:23 testgid
```

Now let's try it with a directory. Let's go to our previous example:

```
[root@rhel8 ~]# cd /var/tmp/
[root@rhel8 tmp]# ls
accounting  finance
[root@rhel8 tmp]# chmod g+s accounting finance
[root@rhel8 tmp]# ls -l
total 0
drwxr-sr-x. 3 root accounting 30 ene 27 19:46 accounting
drwxr-sr-x. 3 root finance    28 ene 27 19:44 finance
[root@rhel8 tmp]# touch finance/testfinance
[root@rhel8 tmp]# ls -l finance/testfinance
-rw-r--r--. 1 root finance 0 ene 28 05:27 finance/testfinance
[root@rhel8 tmp]# touch accounting/testaccounting
[root@rhel8 tmp]# ls -l accounting/testaccounting
-rw-r--r--. 1 root accounting 0 ene 28 05:27 accounting/
testaccounting
```

You can see how, after applying Set-GID to the folders, they show the `s` permission for the group (in bold). Also, when creating new files in those directories the group assigned to them is the same as the group that the parent directory has (also in bold). This way we ensure group permissions are properly assigned.

Using the sticky bit

The last of the permissions to be used is the **sticky bit**. It only has an effect on directories and what it does is simple: when a user creates a file in a directory with the sticky bit, only that user can edit or delete that file.

Let's check an example:

```
[root@rhel8 ~]# ls -ld /tmp
drwxrwxrwt. 8 root root 172 ene 28 04:31 /tmp
```

We could apply those to the previous example, also with `chmod` using `o+t`:

```
[root@rhel8 ~]# cd /var/tmp/
[root@rhel8 tmp]# ls -l
total 0
drwxr-sr-x. 3 root accounting 52 ene 28 05:27 accounting
drwxr-sr-x. 3 root finance    47 ene 28 05:27 finance
[root@rhel8 tmp]# chmod o+t accounting finance
[root@rhel8 tmp]# ls -l
total 0
drwxr-sr-t. 3 root accounting 52 ene 28 05:27 accounting
drwxr-sr-t. 3 root finance    47 ene 28 05:27 finance
```

Let's give it a try. We will add the user `sonia` to the `accounting` group. We will grant a write permission to the group for the `/var/tmp/accounting` directory. Then, we will create a file with the user `matilde` and try to delete it with the user `sonia`. Let's go:

```
[root@rhel8 ~] # usermod -aG accounting sonia
[root@rhel8 ~]# cd /var/tmp/
[root@rhel8 tmp]# chmod g+w accounting
[root@rhel8 tmp]# ls -l
total 0
drwxrwsr-t. 3 root accounting 52 ene 28 05:27 accounting
drwxr-sr-t. 3 root finance    47 ene 28 05:27 finance
[root@rhel8 tmp]# su - matilde
Last login: jue ene 28 05:41:09 CET 2021 on pts/0
[matilde@rhel8 ~]$ cd /var/tmp/accounting/
[matilde@rhel8 accounting]$ touch teststickybit
[matilde@rhel8 accounting]$ exit
logout
[root@rhel8 tmp]# su - sonia
[sonia@rhel8 ~]$ cd /var/tmp/accounting/
[sonia@rhel8 accounting]$ ls -l teststickybit
-rw-rw-r--. 1 matilde accounting 0 Jan 28 05:43 teststickybit
[sonia@rhel8 accounting]$ rm -f teststickybit
rm: cannot remove 'teststickybit': Operation not permitted
```

> **Tip**
>
> The numeric values for special permissions are: `suid` = `4`; `sgid` = `2`; `sticky bit` = `1`.

With this, we have completed how to manage permissions in RHEL.

Summary

In this chapter, we have reviewed the permission management system in RHEL, implemented using traditional permissions. We have learned how to create user accounts and groups, and how to ensure that passwords are managed correctly. We have also learned how passwords are stored in the system, and even how to block shell access to a user. We have created files and folders, assigning permissions to them and ensuring that users can collaborate with an enforced set of rules.

These are the basics of managing access in RHEL and will be very useful to avoid security issues when managing systems. As this is such an important topic, we recommend reviewing this chapter carefully, reading the `man` pages for the commands shown, and making an effort to have a really good understanding of the topic as it will avoid any uncomfortable situations in the future.

Now you are ready to start providing services to users and managing their access, which is what we will cover in the next chapter. Remember to practice and test thoroughly the lessons learned here.

6 Enabling Network Connectivity

When we installed our system in the first chapter, we enabled the network interface. However, network configuration is, or can be, even more than that.

A server connected to a network might require additional interfaces for configuring other networks; for example, for reaching backup servers, performing internal services from other servers, or even accessing storage that is not presented directly via a Storage Array Network (SAN) as local drives but as, for example, **Internet Small Computer System Interface** (**iSCSI**) drives.

Additionally, a server might use redundant network capabilities to ensure that, in the event of a failure in one of the cards, switches, and so on, the server can still be reached and perform properly.

In this chapter, we will learn about how to define network configuration for our RHEL machine using different methods and perform some basic network troubleshooting.

This knowledge will be key since servers are commonly used to provide services to other systems, and we need networking for that purpose.

In this chapter, we will cover the following topics:

- Exploring network configuration in RHEL
- Config files and NetworkManager
- Configuring network interfaces with IPv4 and IPv6
- Configuring hostname and hostname resolutions (DNS)
- Overview of firewall configuration
- Testing connectivity

Let's get hands-on with networking!

Technical requirements

You can continue using the virtual machine we created at the beginning of this book in *Chapter 1*, *Installing RHEL8*. Additionally, to test network communication, it might be useful to create a second virtual machine or reuse the one we created in the previous chapters for testing the **Network Time Protocol** (**NTP**) configuration as we will use it to check for connectivity. Any additional packages that are required will be indicated in the text. Any additional files that are required for this chapter can be downloaded from `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration`.

Exploring network configuration in RHEL

A network is made of different devices that have been interconnected so that information and resources can be shared among them; for example, internet access, printers, files, and more.

Networks have been present since the beginning computing. Initially, the most common were non-IP-based ones, which were generally used for sharing data across computers in the local network, but with the expansion of internet services and the requirement for applications or remote services, IP networks were expanded and the concept of intranet was introduced, where the **Transmission Control Protocol/Internet Protocol** (**TCP/IP**) was used as transport, and the applications started to be more like internet services (or even be based on them).

The migration to IP-based networks has also adapted other protocols such as **Network Basic Input/Output System** (**NetBIOS**) so that they can run on top of it (it was working on top of **NetBIOS Extended User Interface** (**NetBEUI**), and even if other networks such as **InfiniBand** or **Remote Direct Memory Access** (**RDMA**) are still in use, they are not as common as TCP/IP).

TCP/IP, of course, is built on top of other protocols. You can check the OSI layer definition at `https://www.redhat.com/sysadmin/osi-model-bean-dip`. However, some concepts are still involved. We will cover these when we become familiar with TCP/IP and networks.

Before we get into the actual details, we need to clarify a few common TCP/IP and networking keywords we'll be using from now on:

- **IP address**: This is the address that's used for interacting with other devices on the network.
- **Netmask**: This is used to determine which devices are in the neighborhood. It can be expressed via a mask or via a network size, such as `255.255.255.0` or `/24`.
- **Gateway**: This the IP address of the device that will get all our traffic when the target device is outside our netmask, so that we cannot reach it directly.
- **DNS**: This is the IP address of a server or servers that translate **domain names** into IP addresses so that the hosts can connect to them.
- **MAC address**: This is the physical interface address. It is unique for each card and helps identify the card in the network so that the proper traffic is sent to it.
- **Network Interface Card (NIC)**: This card allows our device to connect to the network. It might be wireless, wired, and so on.
- **Extended Service Set Identification (ESSID)**: This is how a wireless network is named.
- **Virtual Private Network (VPN)**: This is a virtual network that is created between the client and the server. Once established, it allows you to direct connection to the services as if they were local, even if the client and the server are in different places. For example, a VPN network is used to allow remote workers to connect to their corporate network using their private internet connection.
- **Virtual Local Area Network (VLAN)**: This allows us to define virtual networks on top of the actual wiring. We can then use a specific header field to have them correctly understood and processed by the network equipment.
- **IPv6**: This is the replacement protocol for **IPv4**, which is still the predominant protocol in networks today.

In the following sections, we will use some of these terms when we explain how a network is set up and defined in **Red Hat Enterprise Linux** (**RHEL**) systems.

In general, when systems are connected, some relationships between the devices on the network are established. Sometimes, some hosts are providers of services, often called servers, and the consumers are known as clients. When the systems in the network perform roles, these networks are known as **Peer-To-Peer** (**p2p**) networks.

In the next section, we'll become familiar with the configuration files and the different approaches for configuring networking in our system.

Getting to know the configuration files and NetworkManager

Now that we have learned about some of the keywords and concepts of networking, it's the time to look at where we can use them to get our system networked.

Traditionally, network interfaces were configured via text files in the system, under the `/etc/sysconfig/network-scripts/` folder. Those scripts were processed via the utilities provided via the `network-scripts` package, which took care of getting the networking stack up and running with the defined configuration.

> **Important note**
>
> Although the `network-scripts` package is available and can be installed, it is considered to be **deprecated**, which means that the package is provided and available but might disappear in future major versions of the operating system, so they will only be provided to ease the transition toward the updated methods.

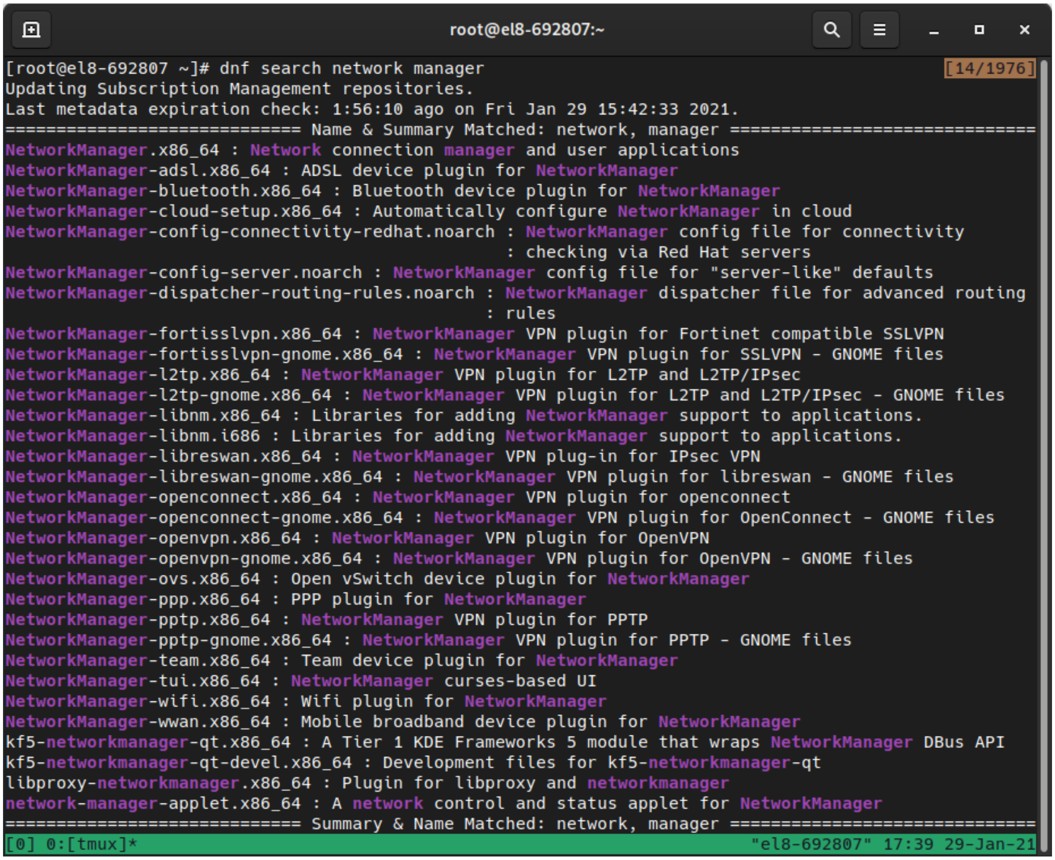
NetworkManager is a utility that was created in 2004 to make network configuration and its usage easier for desktop users. At that point, all configuration was done via text files and it was more or less static. Once a system was connected to a network, the information barely changed at all. With the adoption of wireless networks, more flexibility was required to automate and ease the connection to different networks via different profiles, VPNs, and more.

NetworkManager was created to cover those gaps and aimed to be a component that would be used in many distributions but from a new standpoint, for example, it queries the **Hardware Abstraction Layer** (**HAL**) at startup to learn about available network devices and their changes.

Imagine a laptop system; it can be connected to a wired cable, disconnected when you're moving it to another location or cubicle, can connect to a wireless network, and so on. All those events are relayed to NetworkManager, which takes care of reconfiguring network interfaces, routes, authenticating with the wireless network, and making the user's life a lot easier than it traditionally was.

> **Tip**
>
> The hardware that is connected to the system can be queried with several commands, depending on how the hardware is connected; for example, via utilities such as `lsusb`, `lspci`, or `lshw` (provided by installing the `usbutils`, `pciutils`, and `lshw` packages, respectively).

In the following screenshot, we can see the available packages related to NetworkManager, as obtained via the `dnf search network manager` command:



```
root@el8-692807:~
[root@el8-692807 ~]# dnf search network manager                                    [14/1976]
Updating Subscription Management repositories.
Last metadata expiration check: 1:56:10 ago on Fri Jan 29 15:42:33 2021.
============================= Name & Summary Matched: network, manager ==============================
NetworkManager.x86_64 : Network connection manager and user applications
NetworkManager-adsl.x86_64 : ADSL device plugin for NetworkManager
NetworkManager-bluetooth.x86_64 : Bluetooth device plugin for NetworkManager
NetworkManager-cloud-setup.x86_64 : Automatically configure NetworkManager in cloud
NetworkManager-config-connectivity-redhat.noarch : NetworkManager config file for connectivity
                                                  : checking via Red Hat servers
NetworkManager-config-server.noarch : NetworkManager config file for "server-like" defaults
NetworkManager-dispatcher-routing-rules.noarch : NetworkManager dispatcher file for advanced routing
                                                : rules
NetworkManager-fortisslvpn.x86_64 : NetworkManager VPN plugin for Fortinet compatible SSLVPN
NetworkManager-fortisslvpn-gnome.x86_64 : NetworkManager VPN plugin for SSLVPN - GNOME files
NetworkManager-l2tp.x86_64 : NetworkManager VPN plugin for L2TP and L2TP/IPsec
NetworkManager-l2tp-gnome.x86_64 : NetworkManager VPN plugin for L2TP and L2TP/IPsec - GNOME files
NetworkManager-libnm.x86_64 : Libraries for adding NetworkManager support to applications.
NetworkManager-libnm.i686 : Libraries for adding NetworkManager support to applications.
NetworkManager-libreswan.x86_64 : NetworkManager VPN plug-in for IPsec VPN
NetworkManager-libreswan-gnome.x86_64 : NetworkManager VPN plugin for libreswan - GNOME files
NetworkManager-openconnect.x86_64 : NetworkManager VPN plugin for openconnect
NetworkManager-openconnect-gnome.x86_64 : NetworkManager VPN plugin for OpenConnect - GNOME files
NetworkManager-openvpn.x86_64 : NetworkManager VPN plugin for OpenVPN
NetworkManager-openvpn-gnome.x86_64 : NetworkManager VPN plugin for OpenVPN - GNOME files
NetworkManager-ovs.x86_64 : Open vSwitch device plugin for NetworkManager
NetworkManager-ppp.x86_64 : PPP plugin for NetworkManager
NetworkManager-pptp.x86_64 : NetworkManager VPN plugin for PPTP
NetworkManager-pptp-gnome.x86_64 : NetworkManager VPN plugin for PPTP - GNOME files
NetworkManager-team.x86_64 : Team device plugin for NetworkManager
NetworkManager-tui.x86_64 : NetworkManager curses-based UI
NetworkManager-wifi.x86_64 : Wifi plugin for NetworkManager
NetworkManager-wwan.x86_64 : Mobile broadband device plugin for NetworkManager
kf5-networkmanager-qt.x86_64 : A Tier 1 KDE Frameworks 5 module that wraps NetworkManager DBus API
kf5-networkmanager-qt-devel.x86_64 : Development files for kf5-networkmanager-qt
libproxy-networkmanager.x86_64 : Plugin for libproxy and networkmanager
network-manager-applet.x86_64 : A network control and status applet for NetworkManager
============================= Summary & Name Matched: network, manager ==============================
[0] 0:[tmux]*                                                    "el8-692807" 17:39 29-Jan-21
```

Figure 6.1 – NetworkManagermanager-related packages available for installation in a Red Hat Enterprise Linux 8 system

`NetworkManagermanager` is configured with the files located in the `/etc/NetworkManager` folder, especially `NetworkManager.conf` and the files available in that folder:

- `conf.d`
- `dispatcher.d`
- `dnsmasq-shared.d`
- `dnsmasq.d`
- `system-connections`

Can't remember what a dispatcher is? Remember to use `man networkmanager` to get details on this!

The man page of NetworkManager explains that those scripts are executed in alphabetical order based on network events and will receive two arguments: the name of the device for the event and the action.

There are several actions you can perform, as follows:

- `pre-up`: The interface gets connected to a network but is not activated yet. The script must be executed before the connection can be notified as activated.
- `up`: The interface has been activated.
- `pre-down`: The interface is being deactivated but hasn't been disconnected from the network yet. In the case of forced disconnections (lost wireless connection or lost carrier), this will not be executed.
- `down`: The interface has been deactivated.
- `vpn-up`/`vpn-down`/`vpn-pre-up`/`vpn-pre-down`: Similar to the preceding interfaces but for VPN connections.
- `hostname`: Hostname has been changed.
- `dhcp4-change`/`dhcp6-change`: The DHCP lease has changed (renewed, rebounded, and so on).
- `connectivity-change`: Connectivity transitions such as no connection, system went online, and so on.

Now that we have learned a bit about NetworkManager and how it works and was designed, let's learn how to configure network interfaces.

Configuring network interfaces with IPv4 and IPv6

There are several approaches to configuring network interfaces and several network configurations. These will help us determine what we need to do and the required parameters and settings.

Let's look at some examples:

- A server might have two or more **Network Interface Cards** (**NIC**) for redundancy, but only one of them is active at a time.
- A server might use a trunk network and require that we define VLANs on top for accessing or providing the different services in the networks.
- Two or more NICS might be combined to provide increased output and redundancy via teaming.

Configuration can be performed in several ways, too:

- `nmtui`: Text-based interface for configuring a network
- `nmcli`: The command-line interface for NetworkManager
- `nm-connection-editor`: The graphical tool available for graphical environments
- Via text configuration files

> **Important note**
>
> Before editing your network configuration, ensure that you can reach the system being configured in another way. In the case of a server, this can be done via a remote management card or physical console access. A mistake in the configuration might leave the system unreachable.

Before we move on, let's learn a bit about IPv4 and IPv6

IPv4 and IPv6… what does that mean?

IPv4 was created in 1983 and uses a 32-bit address space, which provides 2^{32} unique addresses (`4,294,967,296`), but from those possible ones, large blocks are reserved for special usage. IPv6, ratified as Internet Standard in 2017, is the latest version at the time of writing and uses a 128-bit address space instead; that is, 2^{128} (3.4×10^{38} addresses).

Long story short, the number of IPv4 addresses seemed huge at the time, but today, where phones, tablets, computers, laptops, servers, lightbulbs, smart plugs, and all of the other **Internet of Things** (**IoT**) devices require an IP address, that number has been depleted for public IP addresses, meaning that it's not possible to assign more. This has caused some **Internet Service Providers** (**ISP**) to use techniques such as **Carrier-Grade Network Address Translation** (**CGNAT**), similar to what private networks do, which causes all the traffic from several devices to appear as coming from only one IP, and having the device interacting on both networks (a router), to do the proper routing from outgoing and incoming packages to the original requestors.

Why no IPv6, then? The main problem is that IPv4 and IPv6 are not interoperable, and even if IPv6 was a draft in 1998, not all network equipment is compatible with it and might not have been yet tested. Check out `https://www.ripe.net/support/training/videos/ipv6/transition-mechanisms` for more details.

In the next section, we will learn about how to configure network interfaces using a text-based user interface to NetworkManager named `nmtui`.

Configuring interfaces with nmtui

`nmtui` provides a text-based interface for configuration. This is the initial screen you'll see when it is executed by running `nmtui` on a Terminal:

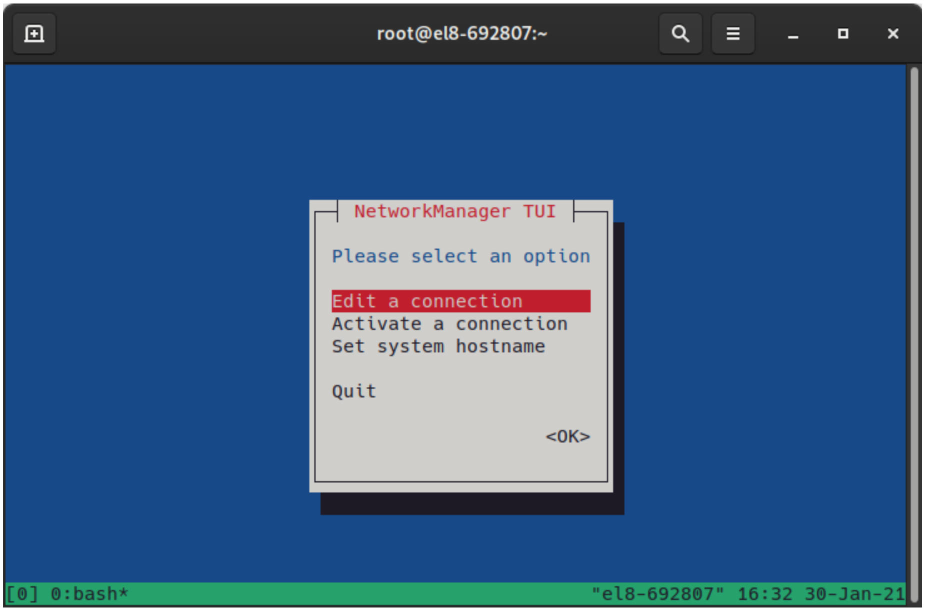


Figure 6.2 – The nmtui welcome screen showing a menu of possible actions that can be performed

Let's explore the available options for our interface. In this case, let's select **Edit a connection**. On the screen that appears, move down and edit the **Wired Connection** option that we have in our system to get to the following screen:

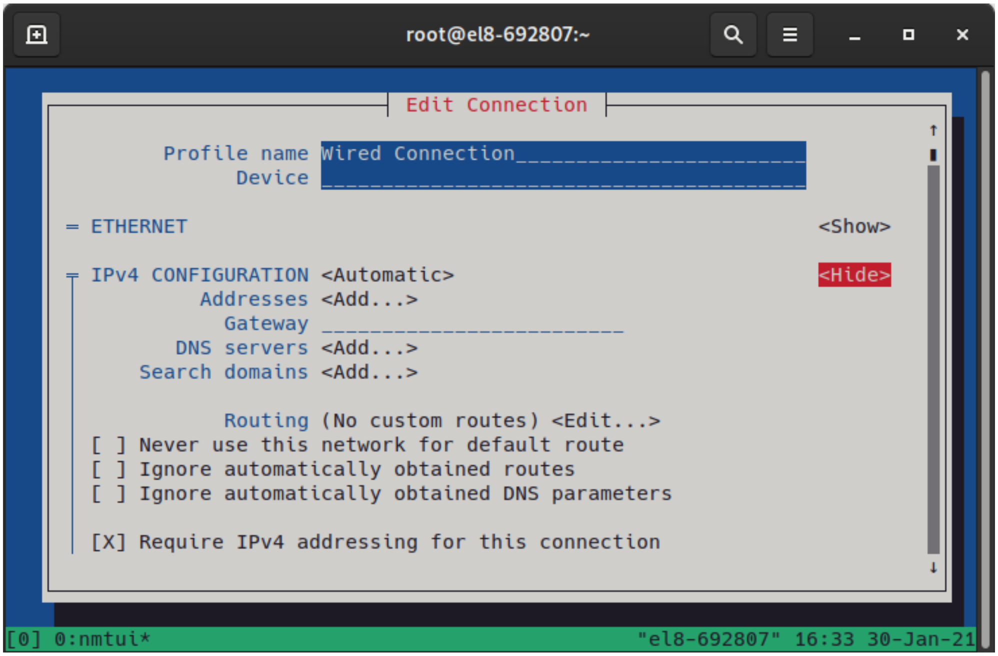


Figure 6.3 – The Edit Connection page with the IPv4 options expanded

It will be hard to show screenshots for each step, as one of the advantages of the text interface is that we can condense a lot of options into a simple screen. However, the preceding screenshot makes it easy to understand each of the required parameters:

- IP address
- Netmask
- Gateway
- Search domain
- Routing

As you can see, there are checkboxes for ignoring routes or DNS parameters that are obtained when the connection is set to `Automatic`. Additionally, there are other options for interfaces: `Disabled`, `Link-Local`, `Manual`, and `Shared`.

Let's discuss the `Automatic` option, which means that the interface will be set to autoconfiguration. This is one of the most common settings for configuration. It does not mean that everything is done magically, though. Let's dig into this a bit more.

In a network (corporate, private, and others), it is typical to have a special service or server doing **Dynamic Host Routing Protocol** (**DHCP**). DHCP is a protocol that runs on top of TCP/IP that allows you to configure hosts dynamically, using the configuration that was made previously either by the network administrator or some appliance and its default settings.

DHCP allows you to autoconfigure (from the client side) many aspects of network configuration, such as IP, netmask, gateway, DNS, search domain, time server, and so on. The configuration that's received is given a lease that is valid for a period of time. After that, the system attempts to renew it or if the system is being powered off or disconnected, the lease is released.

Usually, the DHCP configuration is considered to be tied to dynamic IPs, but keep in mind that a DHCP server can use two different approaches: a pool of IPs that can be reused by different systems connecting and fixed mappings of MAC addresses to static IPs.

Let's, for example, think about a **Small Office – Home Office** (**SOHO**) network with a private IP range in the `192.168.1.0/24` subnet.

We can define our ISP router to be at IP `192.168.1.1` because of the subnet (`/24`), which means that the last part of the IPv4 address can range from 0 to 255.

Using that IP range, we can set up hosts to get dynamic configuration and a dynamic IP from a pool in the last 100 IPs and leave the ones at the beginning for fixed equipment (even if they get the configuration dynamically), such as printers, storage devices, and more.

As we mentioned previously, we can create reservations for servers, but in general, for devices that are always going to have the same address, it is also common practice to configure static addressing. In this way, if the DHCP server becomes unavailable, the servers will be still reachable from other services with either a valid lease or other servers/devices with static addresses configured.

> **Tip**
>
> Just to get familiar with this concept, IP addresses are represented in IPv4 with a dotted notation separating four groups of numbers, such as `192.168.2.12`, while in IPv6, numbers are separated with `:`; for example, `2001:db8:0:1::c000:207`.

Configuring interfaces with nm-connection-editor

If our system has the graphical environment installed, which is not the case for our test system, we can use the graphical configuration tool instead. If it is not installed, proceed to execute `dnf install nm-connection-editor` in a shell console inside your graphical session.

> **Tip**
>
> To install the graphical interface, you can run the `dnf groupinstall "Server with GUI" -y` command or select it during installation.

In the following screenshot, we can see the window that was opened by executing `nm-connection-editor`. It's similar to the text interface shown by `nmtui` earlier in this chapter:

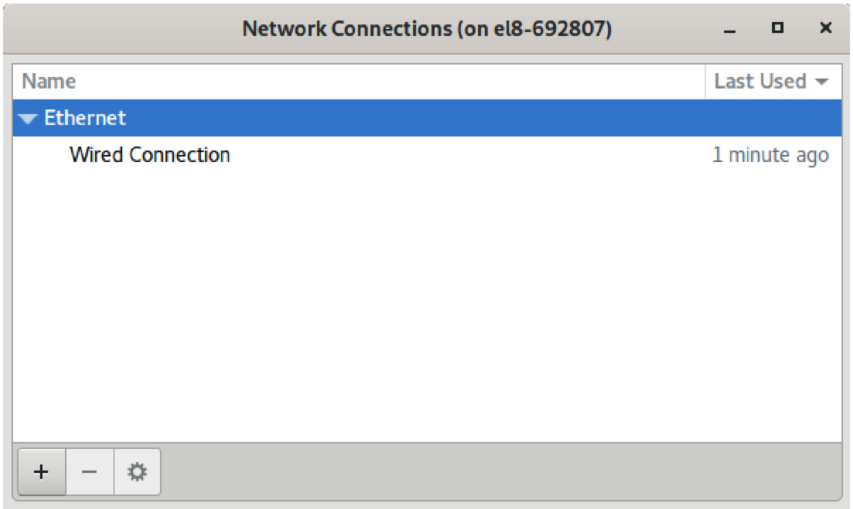


Figure 6.4 – Initial screen for nm-connection-editor

Here, we can see the +, -, and *gear* buttons, which are used to add/remove or configure the connection that's highlighted, respectively.

Let's click on our **Wired Connection** option and then click on the **gear** icon to open the details:

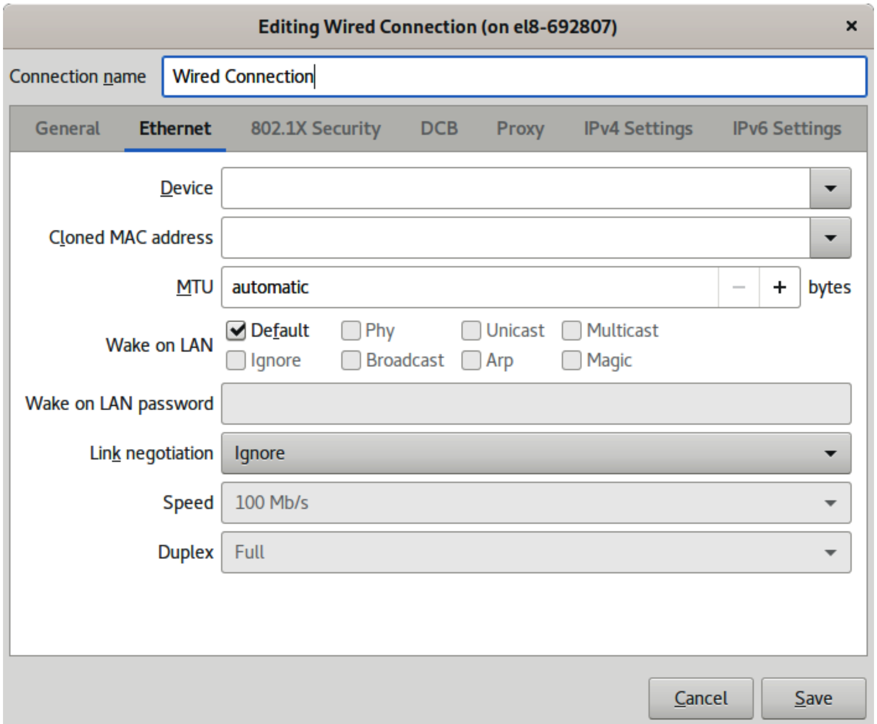


Figure 6.5 – Dialog for editing a network connection

In the dialog, we can see the fields we had in the simpler command-line configuration tool, plus extra fields and different tabs for each group of options.

The important fields to remember are those that are used to **Connect automatically with priority** in the **General** tab. This enables our system to automatically enable that NIC when a connection is available.

As you can find by examining the different tabs, there are lots of choices, such as marking a connection to be metered. This means that, for example, if a connection is made via a mobile phone, extra charges may be specified if network usage is not controlled.

When we're creating extra networks, we can define physical or virtual devices based on the packages we have installed in our system (if you recall the list of packages we saw when searching for NetworkManager, we had packages for different VPNs, Wi-Fi, and others), as we can see in the following screenshot:

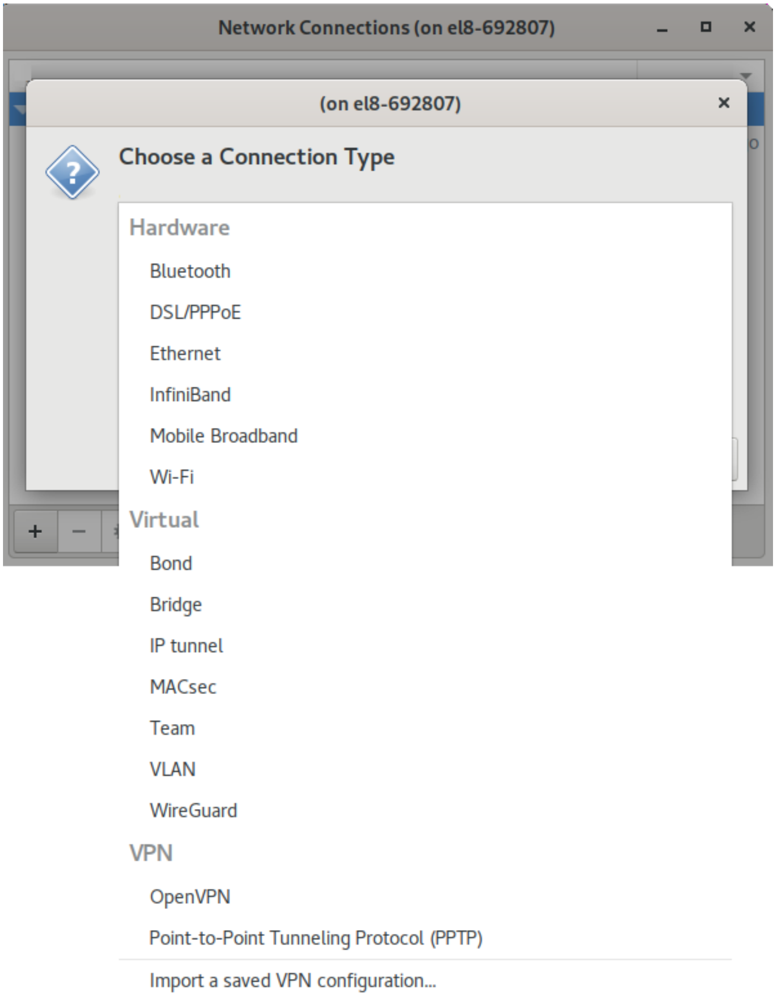


Figure 6.6 – nm-connection-editor with plugins for Wi-Fi, OpenVPN, PPTP, Bluetooth, and more installed

For server environments, the most common network types are **Bonds**, **Bridge**, and **Team** (a part of **Ethernet**), while for desktops, the most common network types are **Ethernet**, **Wi-Fi**, and **Broadband**.

Each type of connection has some requirements. For example, for bonds, bridges, and teams, we need more than one network interface that can be combined.

Now, let's move on a review `nmcli` usage in the next section.

Configuring interfaces with nmcli

`nmcli` is the command-line interface for NetworkManager. It allows us to not only check but also configure the network interfaces in our system, and even if using it might require more memory skills than what `nmtui` required, it empowers users and administrators with scripting capabilities to automate the network setup of our system.

> **Tip**
>
> Most commands allow us to use autocompletion; that is, pressing the *Tab* key will use the autocompletion lists on the command line to suggest the syntax. For example, typing `nmcli dev` on the command line and pressing *Tab* will autocomplete the command to `nmcli device`. In this case, it might not be as critical as `nmcli` takes both arguments as valid, but for others, it's mandatory to spell it properly for the code to work.

Let's start checking the available connections in our system with `nmcli dev`, and then use `nmcli con show` to check out its details:

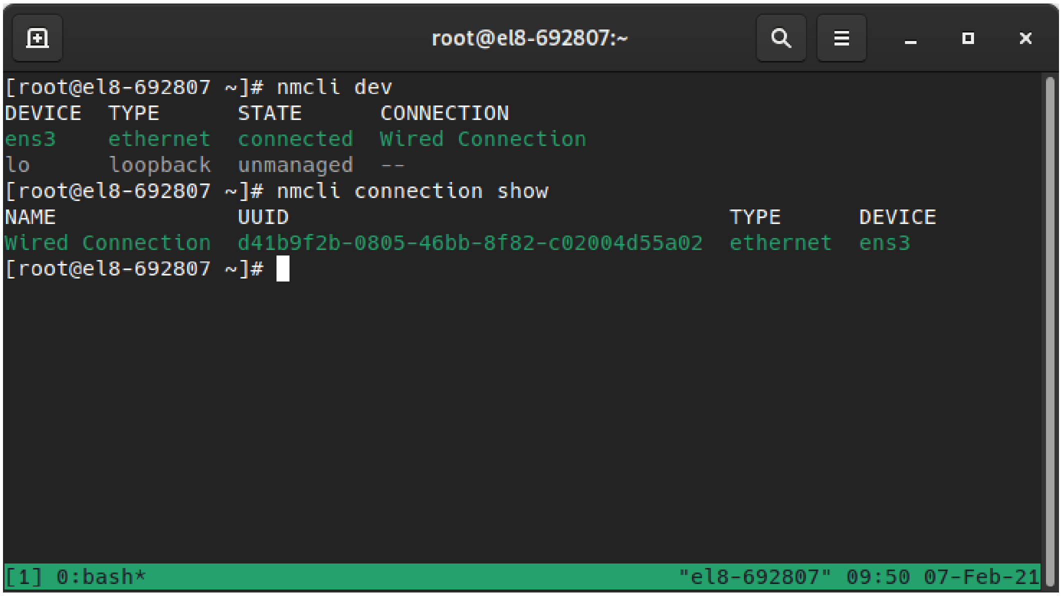


Figure 6.7 – nmcli dev and nmcli con show

When controlling a network connection, for example, when using `nmcli con up "Wired Connection"` or when disabling it with `nmcli con down ens3`, we should bear in mind what we explained about NetworkManager: if the connection is available in the system, NetworkManager might reactivate it just after being disconnected because the connection and the devices required are available in our system.

Now, let's create a new interface to illustrate the process of adding a new connection via IPv4:

```
nmcli con add con-name eth0 type ethernet \
 ifname eth0 ipv4.address 192.168.1.2/24 \
 ipv4.gateway 192.168.1.254
```

We can do the same with IPv6:

```
nmcli con add con-name eth0 type ethernet \
 ifname eth0 ipv6.address 2001:db8:0:1::c000:207/64 \
 ipv6.gateway 2001:db8:0:1::1 ipv4.address \
 192.0.1.3/24 ipv4.gateway 192.0.1.1
```

Once the preceding commands have been executed, we can check the network connections that have been defined with `nmcli connection show eth0` and validate that the proper settings were applied (or, of course, via `nmtui`, `nm-connection-editor`, or the text files that were created on disk as the information is shared and stored in the system).

When we reviewed the output of `nmcli connection show interface`, the output contained some keys separated by dots, such as the following:

- `ipv4.address`
- `ipv4.gateway`
- `ipv6.address`
- `ipv6.gateway`
- `connection.id`

We can use these keys to define new values via `nmcli con mod $key $value`, as shown in the following example:

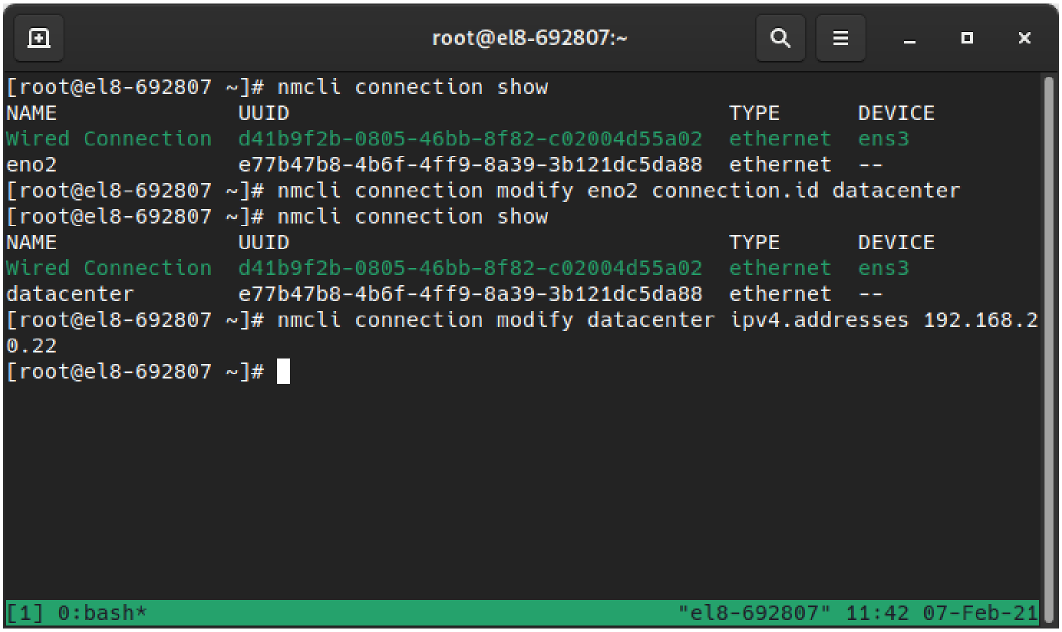


Figure 6.8 – Example of modifying a network connection to change the name of the connection ID and IP address

Of course, after doing the preceding tests, we can also remove the connection to avoid problems in our system with `nmcli con del datacenter`.

The following commands can be used to modify connections with the `nmcli` tool:

- `nmcli con show`: Shows the status of the connections.
- `nmcli con show NAME`: Shows the details of the connection named `NAME`.
- `nmcli dev status`: Shows the statuses of the devices in the system. Note that this means **devices**, not connections that might be using those devices.
- `nmcli con add con-NAME`: Adds a new connection.
- `nmci con mod NAME`: Modifies a connection.
- `nmcli con up NAME`: Brings up a connection.
- `nmcli con down NAME`: Brings down a connection (that can still be reenabled by NetworkManager).
- `nmcli con del NAME`: Removes a connection definition from the system.

> **Tip**
>
> Check `man nmcli-examples` to find more examples that are included in the documentation for the system.

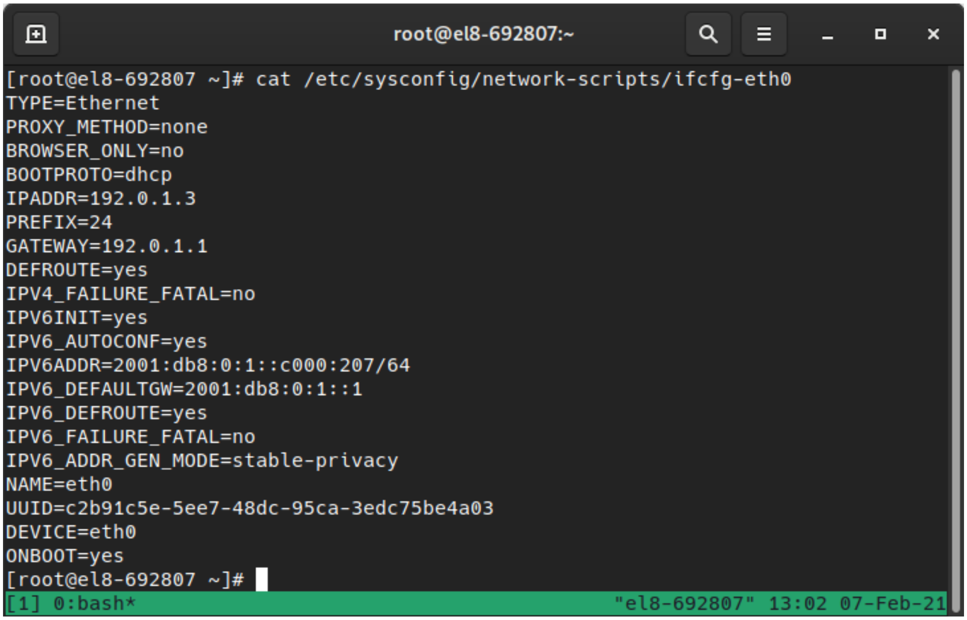
Configuring interfaces with text files

In prior subsections, we explored how to configure the network using different approaches, but in the end, all those configurations end up being written to disk as interface definition files (which also provides backward compatibility with the previously mentioned `network-scripts`).

Instead of starting with creating an interface definition from scratch, let's examine what `nmcli` did when we created the interface with the following command:

```
nmcli con add con-name eth0 type ethernet ifname eth0 ipv6.
address 2001:db8:0:1::c000:207/64 ipv6.gateway 2001:db8:0:1::1
ipv4.address 192.0.1.3/24 ipv4.gateway 192.0.1.1
```

The preceding command will generate the `/etc/sysconfig/network-scripts/ifcfg-eth0` file, which we can see in the following screenshot:



```
root@el8-692807:~
[root@el8-692807 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
IPADDR=192.0.1.3
PREFIX=24
GATEWAY=192.0.1.1
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6ADDR=2001:db8:0:1::c000:207/64
IPV6_DEFAULTGW=2001:db8:0:1::1
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=eth0
UUID=c2b91c5e-5ee7-48dc-95ca-3edc75be4a03
DEVICE=eth0
ONBOOT=yes
[root@el8-692807 ~]#
[1] 0:bash*                                   "el8-692807" 13:02 07-Feb-21
```

Figure 6.9 – Contents of the /etc/sysconfig/network-scripts/ifcfg-eth0 connection definition

As we can see, by default, we have specified a network interface of the `Ethernet` (`TYPE`) type using the `eth0` device, along with the provided values for IPv4 and IPv6 addressing and gateways. The names of the keys differ from the ones defined with `nmcli`, and the reason for this is that we have backward compatibility.

Note that in the preceding example, the `ONBOOT` field has already been set to `yes`, which means that the interface will be enabled automatically when the system is booting. If we're using `nmcli`, we can check the status via the `connection.autoconnect` configuration key, which will also, by default, make the connection automatically enabled at boot.

We can edit those files directly, but to let NetworkManager be aware of the changes that will be introduced, `nmcli con reload` must be executed. This will sync the changes that are made to the individual files.

For example, we can correct one of the settings in the preceding file because for statically defined IPs, it is a common practice to define `BOOTPROTO=none`. Use your preferred method to modify the `/etc/sysconfig/network-scripts/ifcfg-eth0` file (`vim`, `nano`, `sed`, or others). To get other details we can check with `nmcli` and also change the IP address.

Note how, in the following screenshot, the changes don't appear in `nmcli` until we issue the `reload` command:

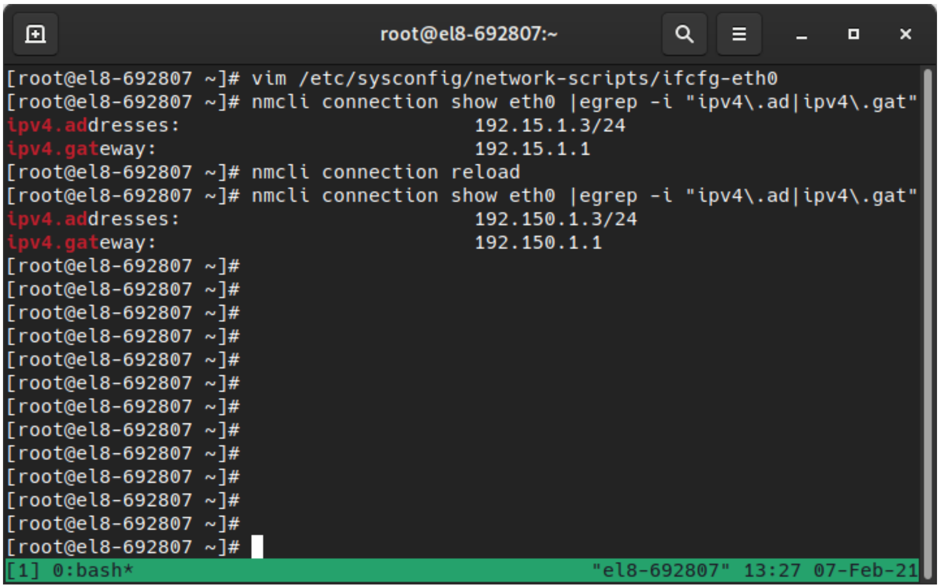


Figure 6.10 – The process of editing an interface definition doesn't show up on nmcli until we reload the connections

Of course, we can also create network definitions from scratch, and until the arrival and spread of NetworkManager, this method was used in scripting, including Anaconda automated installations via kickstart files.

Let's create a simple network definition in IPv4 with the commands show in the following screenshot:

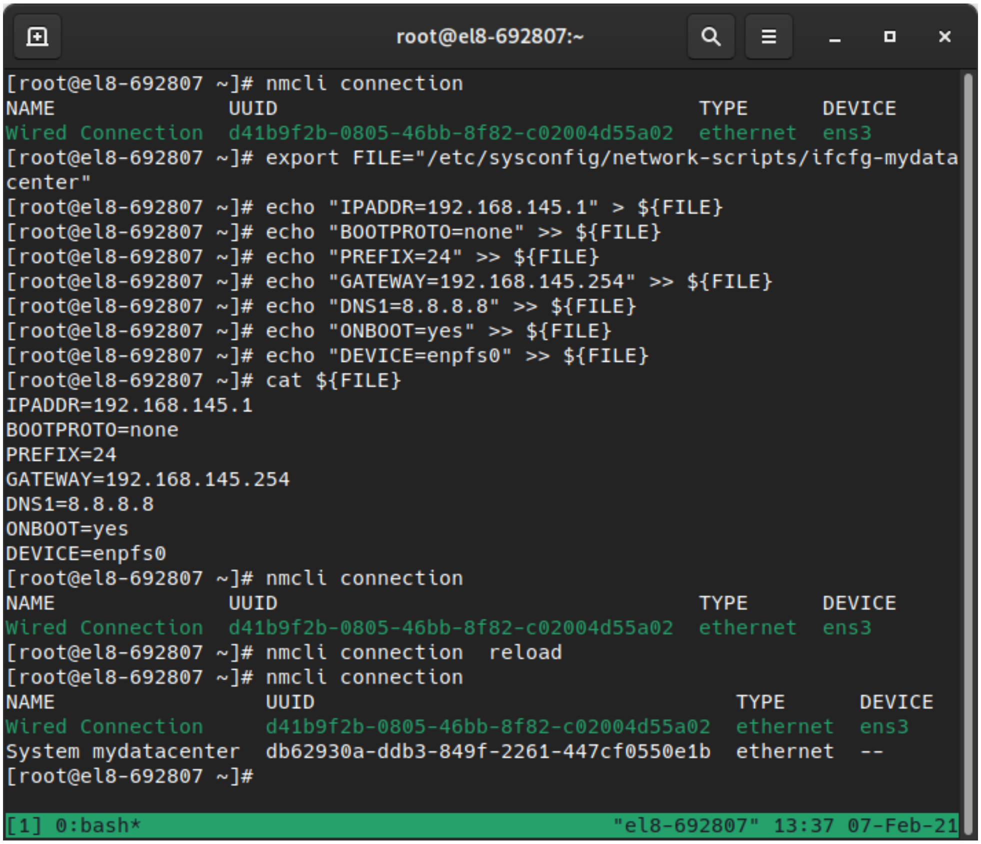


Figure 6.11 – Creating a connection using a configuration file (that can be part of a script)

Here, you can see not only the creation of the connection but the previous status, the interface definition, the NetworkManager view of the system, and a comparison of the configuration files that were reloaded. Note that the device column is empty as we have defined an interface for that connection that is not present in our system.

> **Important note**
>
> The network interface definition can become a nightmare as the interface name itself is subject to several rules, such as the location of the interface in the bus, if it was previously seen, and so on. Generally, once the network cards have been detected in the system, a custom rule is written that matches the MAC address of the interface to a custom naming convention. This happens so that it does not change across reboots or if a new software update changes the way we must enumerate the cards. You can read more about this topic by looking at the official RHEL8 manual at `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/consistent-network-interface-device-naming_configuring-and-managing-networking`.

Now that we have reviewed the different ways to configure networking in our system, let's learn about naming resolutions.

Configuring hostname and hostname resolutions (DNS)

Remembering IP addresses, whether they are IPv4 or IPv6 addresses, can become a nightmare. To make things easier, a more human approach was used for the hostnames and the DNS, in that we can translate those easier-to-remember names into the IP addresses that systems use for connecting.

Hostnames are the names we assign to a host to have them identified, but when they're used in addition to a DNS server, we must have other hosts that can *resolve* them into IP addresses they can connect to.

We can use the `hostname` command to see or to temporarily modify the current hostname, as shown in the following screenshot:

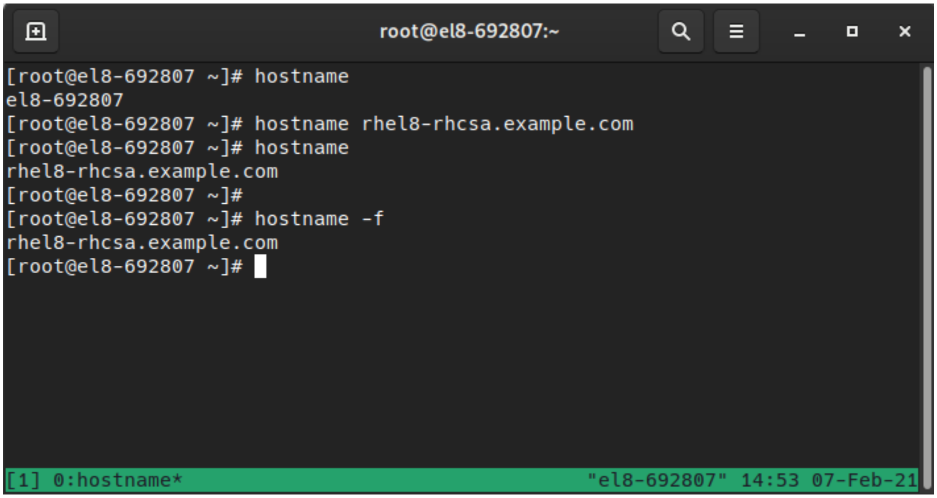


Figure 6.12 – Querying and changing the hostname for our host

Bear in mind that this change is only temporary; as long as we restart the server, it will use the configured one.

To define a new configured hostname, we will use the `hostnamectl set-hostname` command, as shown in the following screenshot:

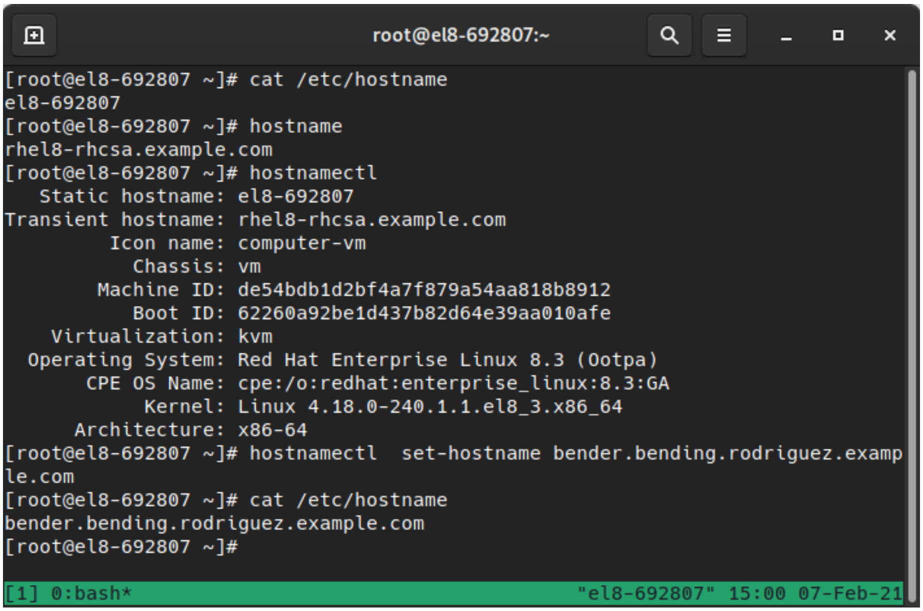


Figure 6.13 – Checking the previously configured hostname and the definition of a new one via hostnamectl

Note in the preceding example how we have `Transient hostname` versus `Static hostname`, which refers to the temporary status of the name that was defined with `hostname` instead of `hostnamectl`.

When it comes to name resolution, there are several approaches we can take. One, of course, is to use DNS servers, which we will explain later in this section, but there are other ways.

In general, systems have several resolvers, and those are defined in the `/etc/nsswitch.conf` configuration file. Those resolvers are not only for network naming, but, for example, for resolving users, where, for example, a corporate **LDAP** server might be used for defining users, passwords, and so on. By default, `nsswitch.conf` instructs our system to use, for hosts resolving this entry: `hosts:        files dns myhostname`.

This means that we are using the files in our `/etc/` directory as our first source. In the case of hostnames, this refers to the `/etc/hosts` file. If an entry is defined in that file, the value that was specified will be used; if not, the `/etc/resolv.conf` file will determine how to proceed with its resolution. Those files, and especially `resolv.conf`, are configured when the system is deployed and when a connection is activated. NetworkManager takes care of updating the values that were obtained via DHCP if autoconfiguration was used, or the specified DNS servers if manual configuration was performed.

In the following screenshot, we can see the entries that have been defined in our `/etc/hosts` file, how pinging a host fails because the name does not exist, and how, after manually adding an entry to the `/etc/hosts` file, our system is able to reach it:

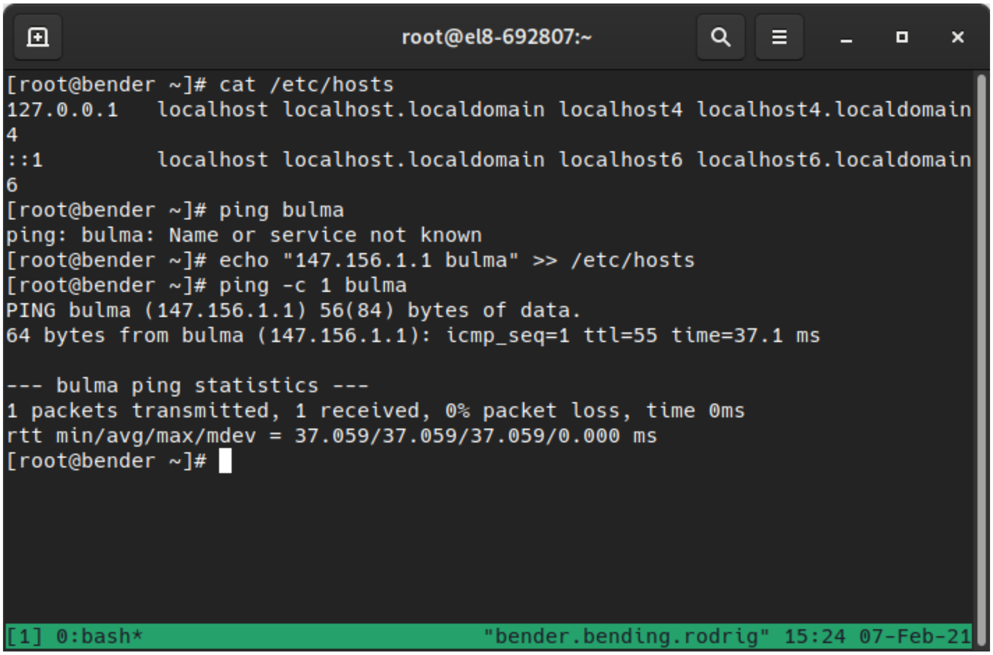


Figure 6.14 – Adding a static host entry to our local system

As we mentioned previously, DNS resolution is done via the configuration at `/etc/resolv.conf`, which, by default, contains a `search` parameter and a `nameserver` parameter. If we check the man page of `resolv.conf`, we can obtain descriptions for the common parameters:

- `nameserver`: Contains the IP of the nameserver to use. Currently, only a maximum of three entries (each on its own line) will be used by the `resolv` library in the system. The resolution is performed in order each time, so if one server fails, it will time out, try with the next one, and so on.
- `domain`: The local domain name. It allows us to use short names to hosts that are relative to the local domain in our host. If it's not listed, it's calculated based on the hostname of our system (everything after the first "`.`").
- `search`: By default, this contains the local domain name, and it's the list of domains we can attempt to use to resolve the short name that's provided. It's limited to 6 domains and 256 characters. Domain and search are mutually exclusive, since the last one in the file is the one to be used.

> **Tip**
>
> DNS resolution works by asking special servers (DNS) for the relevant data for a domain. This happens in a hierarchical way, with the top-most general servers being called **root servers**. DNS servers contain not only registers or entries for converting hostnames into IPs, but also information about the mail server to use when sending an email, verification details for security, reverse entries, and more. Also, DNS servers can be used to block access to services by returning invalid IPs for some domains, or to speed up internet navigation by using faster DNS servers than the ones provided by the ISP. When a domain name is registered, a new entry is created in the root tables for the domain pointing to the DNS server. This will take care of that domain resolution, and later, those entries will be populated and cached across the internet for faster resolution.

If we want to modify the DNS servers defined for a connection, remember to use `nmcli con mod NAME ipv4.dns IP` (or IPv6 equivalent) and use a `+` symbol beforehand, as in `+ipv4.dns`, to add a new entry to the list of DNS servers. Any manual changes that are to `resolv.conf` might be overwritten.

Now that we have learned about how DNS works and how our system uses it, let's look at how to secure system network access.

Overview of firewall configuration

When a system is connected to a network, many of the services running can be reached from other systems. That is the goal behind having systems connected. However, we also want to keep systems secure and away from unauthorized usage.

A **firewall** is a software layer that sits between the network cards and the services and allows us to fine-tune what is allowed or not.

We cannot completely block all the incoming connections to our system as often, the incoming connection is a response from a request that our system made.

The connections are blocked via a kernel framework named **netfilter**, which is used by the firewall software to modify how the packets are processed. **Nftables** is a new filter and packet classifier subsystem that enhances parts of netfilter code, but retains the architecture and provides faster processing among other features using only one interface (**nft**), thus deprecating old frameworks such as `iptables`, `ip6tables`, `ebtables`, and `arptables`.

> **Important note**
>
> As we explained earlier in regard to network configuration, a bad configuration in a firewall can lock you out of the system, so be extremely careful when you're setting some restrictive rules so that you can log into the system again if you are remotely accessing it.

firewalld is a frontend to the nftables framework that, before its adoption, interfaced with netfilter via iptables. In the vast majority of occasions, firewalld should be able to deal with the requirements for filtering, so it is the recommended frontend for editing rules. It can be installed on your system by you installing the `firewalld` package, which should be included in a base installation. It will provide the `firewall-cmd` command once installed for interacting with the service.

firewalld uses the concept of zones, which allows us to predefine a set of rules for each of those zones. These can also be assigned to network connections. This is more relevant, for example, for laptops that might be roaming across connections, and they might have some default settings for when you're using home or corporate connections. However, they will default to a more secure one when you're using Wi-Fi from a cafeteria.

firewalld also uses predefined services so that the firewall knows what ports and protocols should enable based on the services and zones they have been enabled on.

Let's check out the available zones and some more details about the home zone:

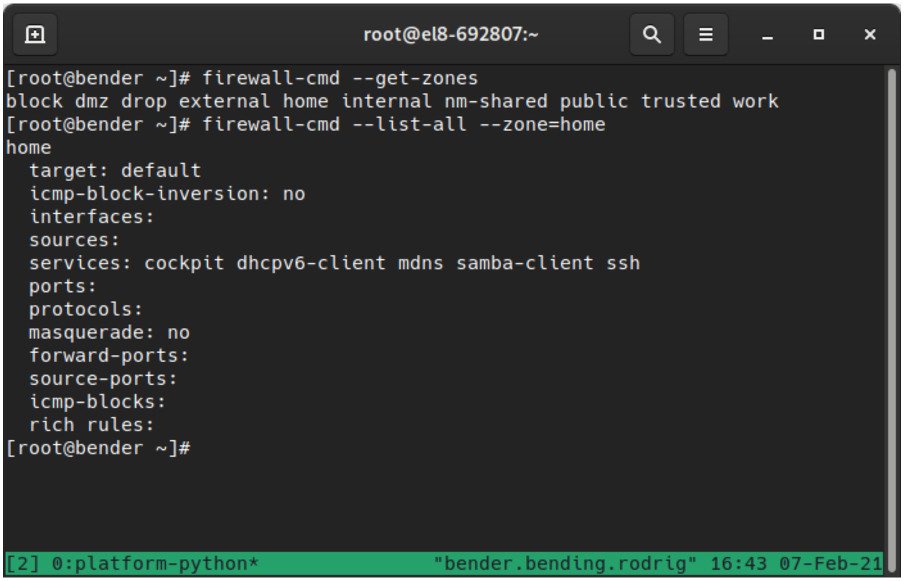


Figure 6.15 – Available zones and configuration for the zone home

As we can see, several zones have been defined:

- `public`: This is the default zone for newly added interfaces. It allows us to cockpit SSH and DHCP clients and rejects all incoming traffic not related to the outgoing traffic.
- `block`: Rejects all incoming traffic unless it's related to outgoing traffic.
- `dmz`: Rejects all incoming traffic unless it's related to outgoing or SSH connections.
- `drop`: Drops all incoming packets that are not related to outgoing ones (not even ping).
- `external`: Blocks all incoming traffic except that related to outgoing traffic. It also allows SSH and it masquerades traffic as originating from this interface.
- `home`: In addition to public, its allows `smb` and `mdns`.
- `internal`: Based on the home zone.
- `trusted`: Allows all incoming traffic.
- `work`: Blocks all incoming traffic except that related to outgoing or SSH/cockpit/DHCP traffic.

Next, we'll learn how to use those zones when we're configuring the firewall.

Configuring the firewall

As shown in the introduction to this section, a firewall can be configured via the `firewall-cmd` command (as well as the cockpit web interface, which was described earlier in this book on *Chapter 4*, *Tools for Regular Operations*). The most common command options that are used are as follows:

- `firewall-cmd --get-zones`: Lists the available zones.
- `firewall-cmd --get-active-zones`: List the active zones and interfaces that have been assigned.
- `firewall-cmd --list-all`: Dumps the current configuration.
- `firewall-cmd --add-service`: Adds a service to the current zone.
- `firewall-cmd --add-port`: Adds a port/protocol to the current zone.

- `firewall-cmd --remove-service`: Removes the service from the current zone.
- `firewall-cmd --remove-port`: Removes the port/protocol from the current zone.

> **Important note**
> Note that you need to mention the port number and service name after the preceding commands to add or remove a service/port.

- `firewall-cmd --reload`: Reloads the configuration from the saved data, thus discarding the runtime configuration.
- `firewall-cmd –get-default-zone`: Gets the default zone.
- `firewall-cmd --set-default-zone`: Defines the default zone to use.

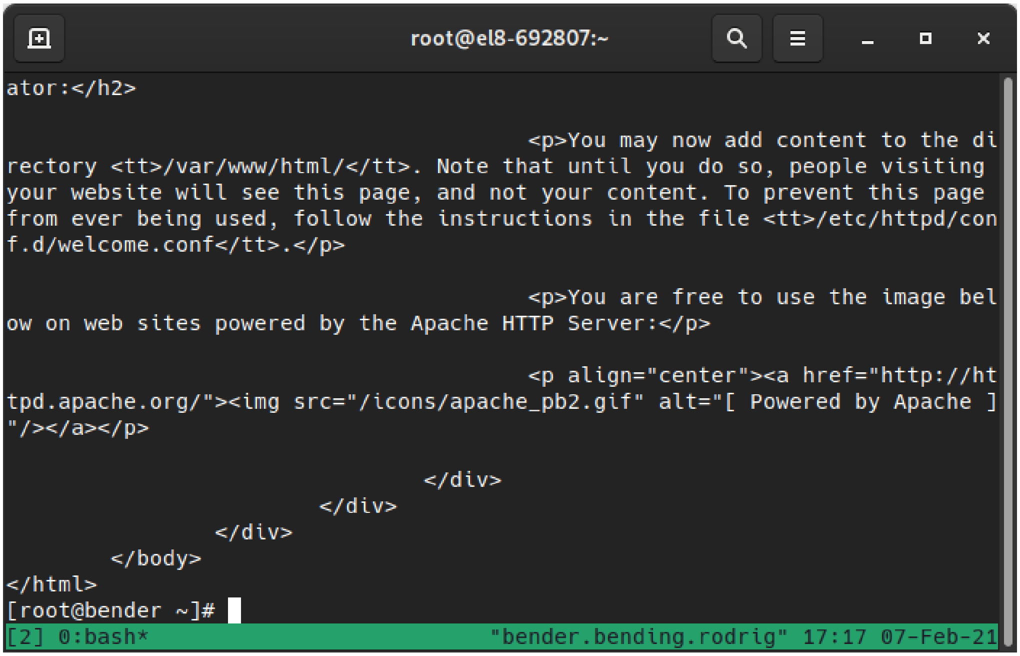
For example, when we install an HTTP server in our system (for serving web pages), port `80` on TCP must be enabled.

Let's try this in our sample system by installing, running, and opening the HTTP port:

```
dnf -y install httpd
systemctl enable httpd
systemctl start httpd
firewall-cmd –add-service=http
curl localhost
```

The last command will make a petition to the local `http` server to grab the results. If you have access to an additional system, you can try to connect to the IP of the server that we have been using to watch the default web page be served by the system.

In the following screenshot, we can see the output of the `curl localhost` command:



```
root@el8-692807:~
ator:</h2>

                                        <p>You may now add content to the di
rectory <tt>/var/www/html/</tt>. Note that until you do so, people visiting
your website will see this page, and not your content. To prevent this page
from ever being used, follow the instructions in the file <tt>/etc/httpd/con
f.d/welcome.conf</tt>.</p>

                                        <p>You are free to use the image bel
ow on web sites powered by the Apache HTTP Server:</p>

                                        <p align="center"><a href="http://ht
tpd.apache.org/"><img src="/icons/apache_pb2.gif" alt="[ Powered by Apache ]
"/></a></p>

                                </div>
                        </div>
                </div>
        </body>
</html>
[root@bender ~]# 
[2] 0:bash*                                 "bender.bending.rodrig" 17:17 07-Feb-21
```

Figure 6.16 – Output of curl when requesting the web page hosted by our system

At this point, we have reviewed how to configure some basic firewall rules, so we are ready to check the network's connectivity.

Testing network connectivity

In the previous sections, we were interacting with network interfaces, addresses, and firewall rules that define, limit, or allow connections to our system. In this section, we will review some of the basic tools that can be used to validate that network connectivity exists.

Note that the following commands assume that the firewall is not set to strict mode and that we can use the **Internet Control Message Protocol** (**ICMP**) to reach the servers hosting the service. In secured networks, the service might be working but not answering to ping – it may only be answering the service queries themselves.

There are several commands that we can use here, so consider these suggestions for diagnosing issues:

- Check the local interface's IP address, netmask, and gateway.
- Use the `ping` command with the IP address of the gateway to validate the proper network configuration.
- Use the `ping` command to ping the DNS servers in `/etc/resolv.conf` to see if those are reachable. Alternatively, use the `host` or `dig` command to query the DNS servers.
- If there's supposedly external network connectivity, try to reach external DNS servers such as `8.8.8.8` or `1.1.1.1` or use `curl` or `wget` to request some of the web pages of known services; for example, `curl nasa.gov`.

This should give you a rough idea of where a problem might be, based on how far you reach into the tests. Remember that there are other tools, such as `tracepath`, that will show the hops a TCP packet does before reaching the destination. The man pages for each command will give you hints and examples on their usage.

In the following screenshot, you can see the output of `tracepath` against one web server:

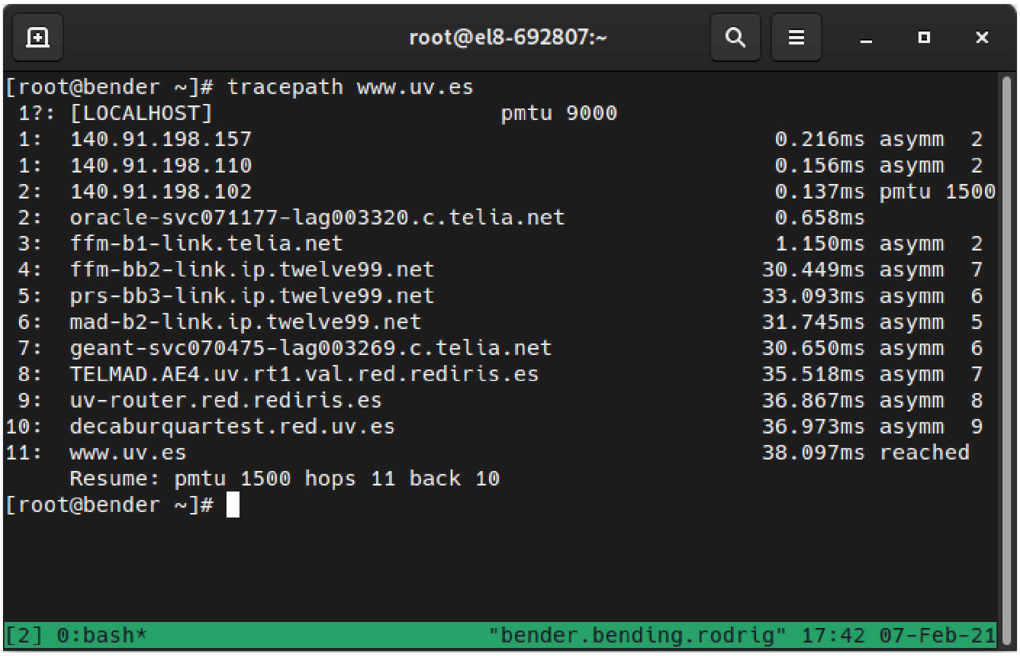


Figure 6.17 – The output of the tracepath command against the University of Valencia, Spain web server

As we can see, there were 11 steps that were performed across different servers until our data package reached the destination host. This allowed us to learn about how a package traverses the internet to reach target systems.

Summary

In this chapter, we learned about configuring network interfaces using different approaches, either via manual interaction or via methods that allow us to script or automate the configuration.

Some troubleshooting for network issues was also introduced to help us find some of the basic errors that might occur.

As we mentioned in this chapter's introduction, networking is the basis for our system to reach other services and to provide services to other systems. We also introduced the idea of more complex network setups that fall outside the scope of this RHCSA level, but it's interesting to at least be familiar with the keywords that we'll be using in our career.

In the next chapter, we will cover some important topics related to security such as adding, patching, and managing the software in our systems.

7
Adding, Patching, and Managing Software

Maintaining a system's software, to close security issues, apply fixes, and keep the system up to date is an essential task in systems management. In this chapter, we will review how the **Red Hat Subscription Management system** works, how to ensure that packages are verified, and other software management tasks to keep the system fresh.

Going a bit more into the details, in this chapter, we will go over how the subscription system works and how to use your developer subscription for self-training or installing a personal server. We will also check how to manage the software origins, also known as repositories, that your system will use. That includes learning the role of signatures in package management, to ensure the software installed is the one Red Hat provides. We will also learn about key tasks such as adding and removing packages and package groups, using different software versions with modularity, and reviewing and rolling back changes.

To simplify expanding your knowledge, enabling you to prepare your own labs, we will see how to have a full local copy of all the **Red Hat Enterprise Linux (RHEL)** repositories in your system.

And last but not the least, we need to understand **Red Hat Package Manager** (**RPM**), now changed to RPM Package Manager, by learning the basics of how the internals of package management work.

In summary, we will cover the following topics in this chapter:

- RHEL subscription registration and management
- Managing repositories and signatures with Yum/DNF
- Doing software installations, updates, and rollbacks with Yum/DNF
- Creating and syncing repositories with createrepo and reposync
- Understanding RPM internals

Now, let's start managing the software in our systems.

RHEL subscription registration and management

RHEL is a fully **open source operating system**, which means that all the source code used to build it is available to access, modify, redistribute and learn from. Pre-built binaries are, on the other hand, delivered as a service, and accessible via a subscription. As seen in *Chapter 1*, *Installing RHEL8*, we can have, for our own personal use, a developer subscription. That subscription provides access to ISO images, but also to the updated, signed packages that are part of RHEL 8. These are the exact same bits that are used in production by so many companies worldwide.

Let's see how to use that subscription with our own system.

First, let's take a look at the **Red Hat Customer Portal** at `https://access.redhat.com` and click **LOG IN**:

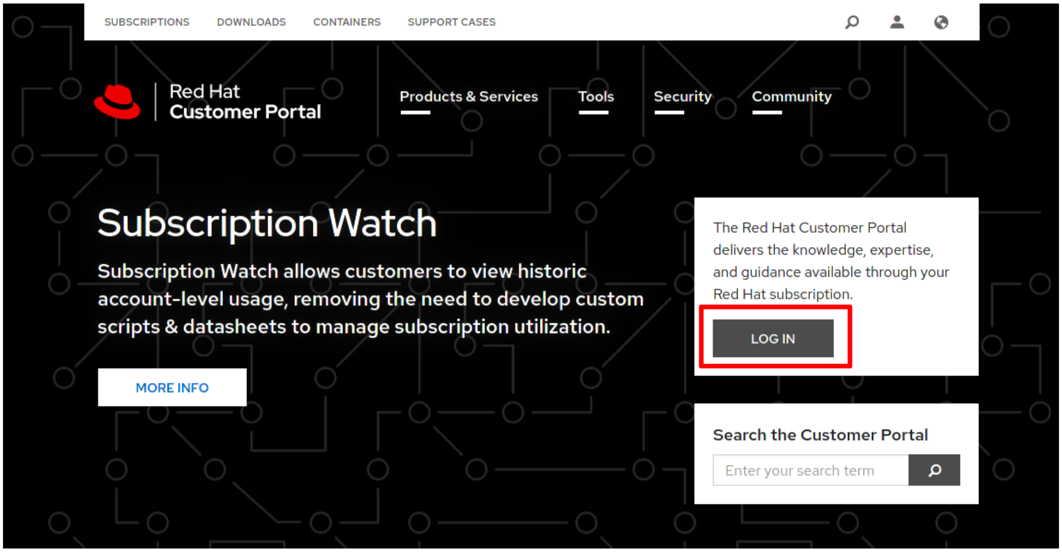


Figure 7.1 – Log into the Red Hat Customer Portal

Once we click on **LOG IN**, we will be redirected to the **Single Sign-On** page for all the Red Hat services. There, we will need to use the username we created in *Chapter 1, Installing RHEL8*. In the following screenshot, we are using `student` as an example:

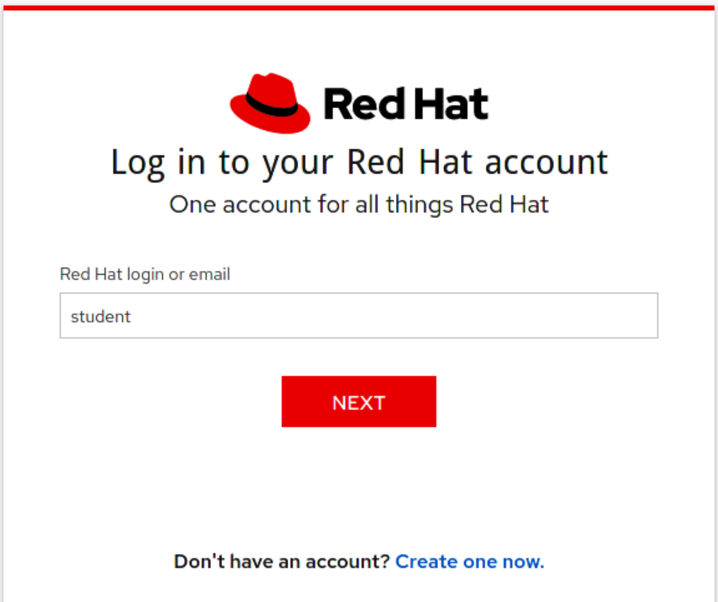


Figure 7.2 – Entering our username in Red Hat Single Sign-On

Now it's time to type our password to validate:

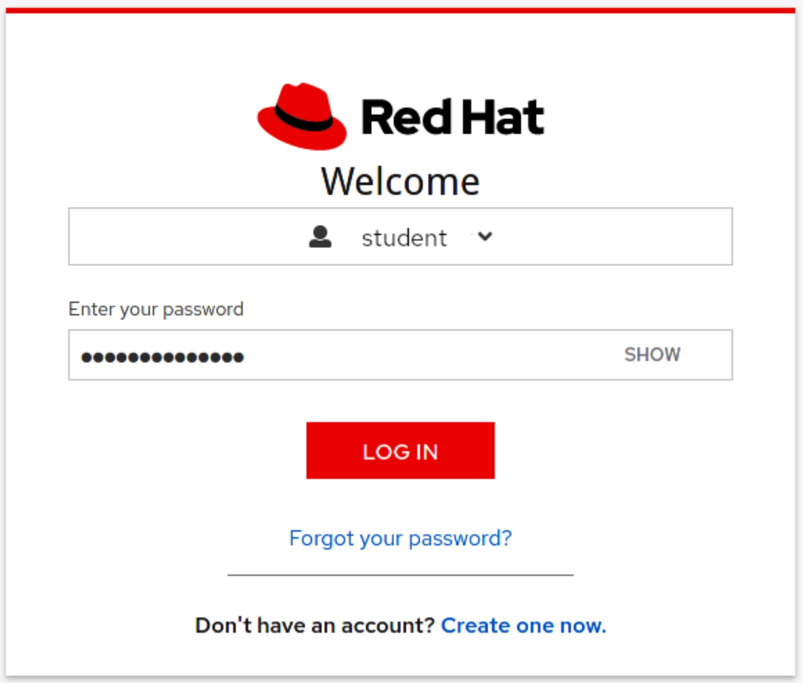


Figure 7.3 – Entering our password in Red Hat Single Sign-On

Once logged in, we shall go to the **Red Hat subscriptions page** by clicking on the **SUBSCRIPTIONS** link in the top bar:

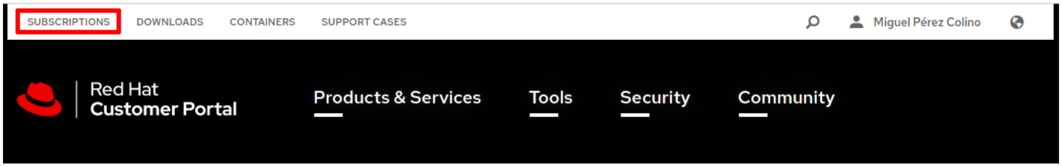


Figure 7.4 – Accessing the subscriptions page in the Red Hat Customer Portal

The subscriptions page will look like this for a user with one physical machine subscribed:

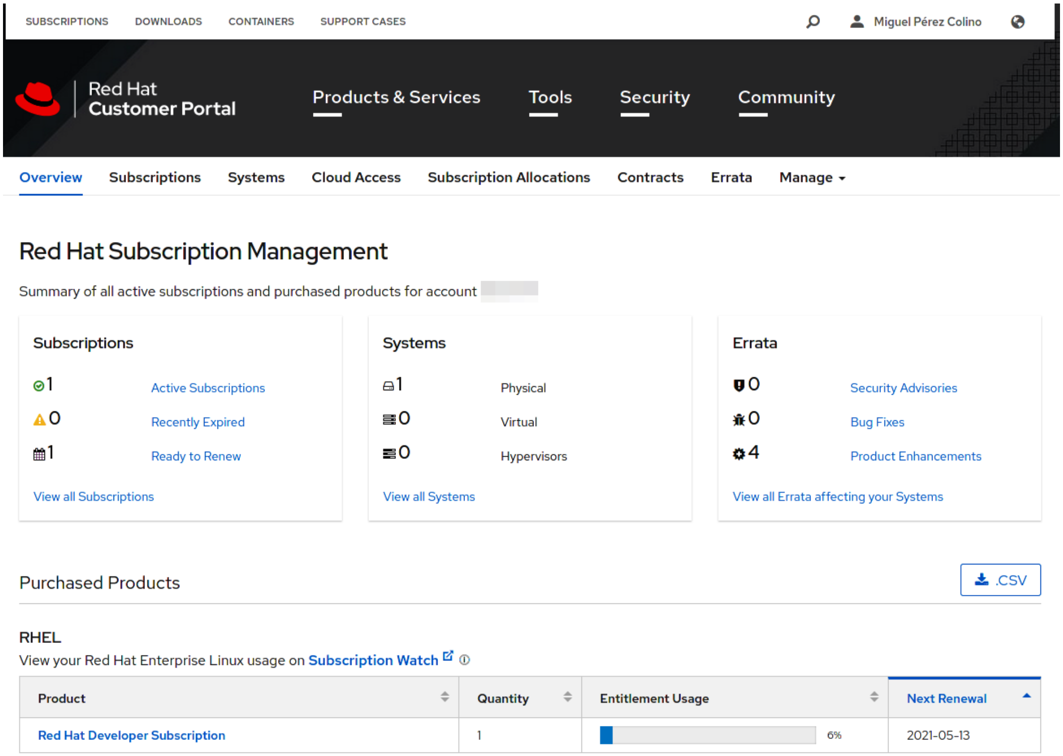


Figure 7.5 – Subscription page example in the Red Hat Customer Portal

> **Tip**
>
> The Developer Subscription was updated in January 2021 to support up to 16 systems. You can use your account for more than one single system to simulate larger production-like deployments.

Now let's register our new system:

```
[root@rhel8 ~]# subscription-manager register
Registering to: subscription.rhsm.redhat.com:443/subscription
Username: student
Password:
The system has been registered with ID: d9673662-754f-49f3-
828c-86fd9f5b4e93
The registered system name is: rhel8.example.com
```

With this, our system will be registered in the Red Hat **Content Delivery Network** (**CDN**) but will still not have a subscription assigned.

Let's go to the subscriptions page and refresh to see the new system there. We will click on **View all Systems** to continue:

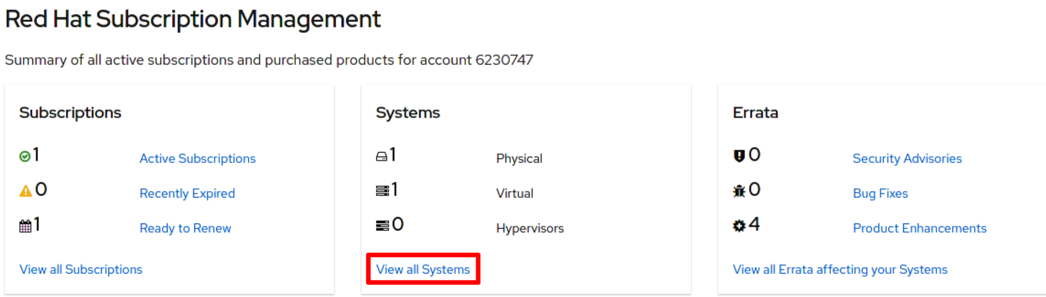


Figure 7.6 – Subscriptions page with the new subscribed system

We can see our new system on the page, `rhel8.example.com`, with a red square next to it to show it does not have a subscription attached to it. Let's click on the name of the system to see the details:

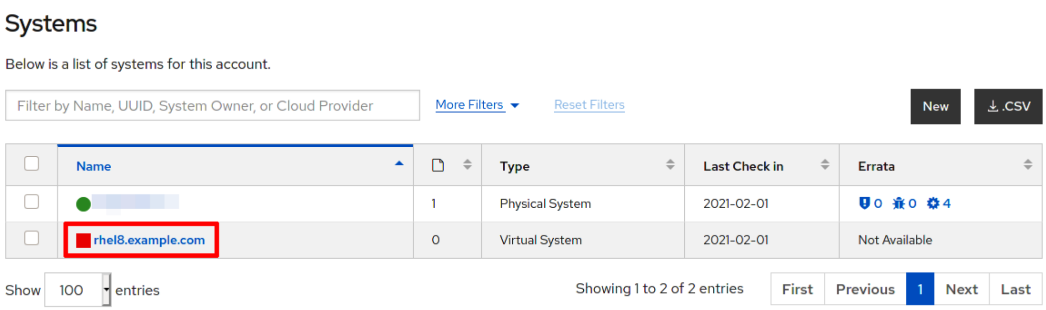


Figure 7.7 – Subscription page with the new subscribed system

Once on the specific system page, we see all the details for the system. We click on **Subscriptions** to see the attached ones:

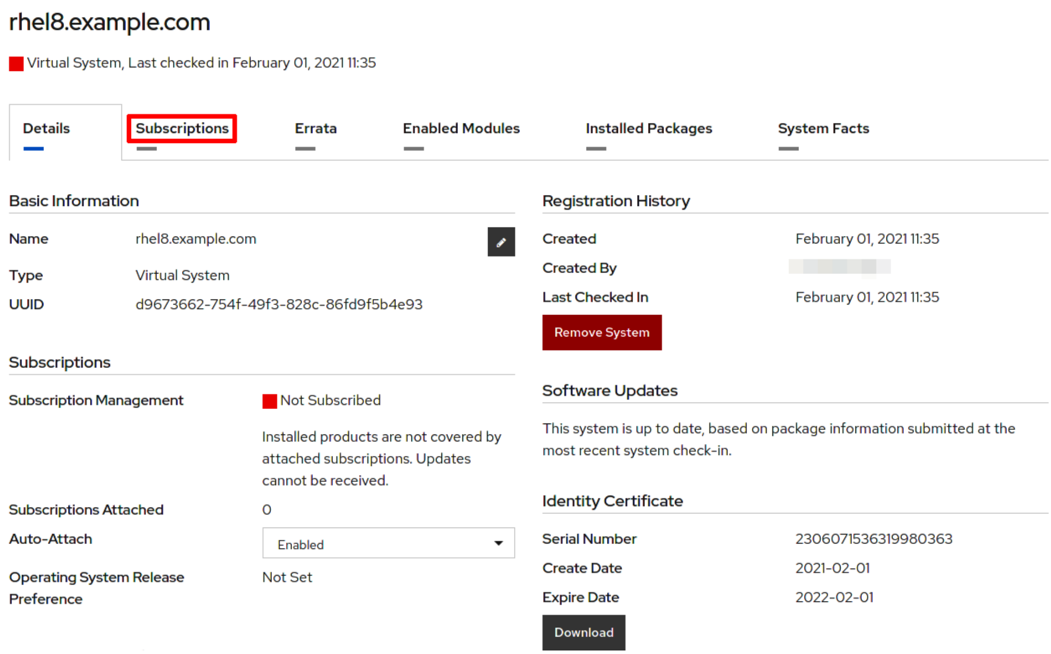


Figure 7.8 – Subscriptions page with the new subscribed system's details

We can see on the page that there are no attached subscriptions for this system:

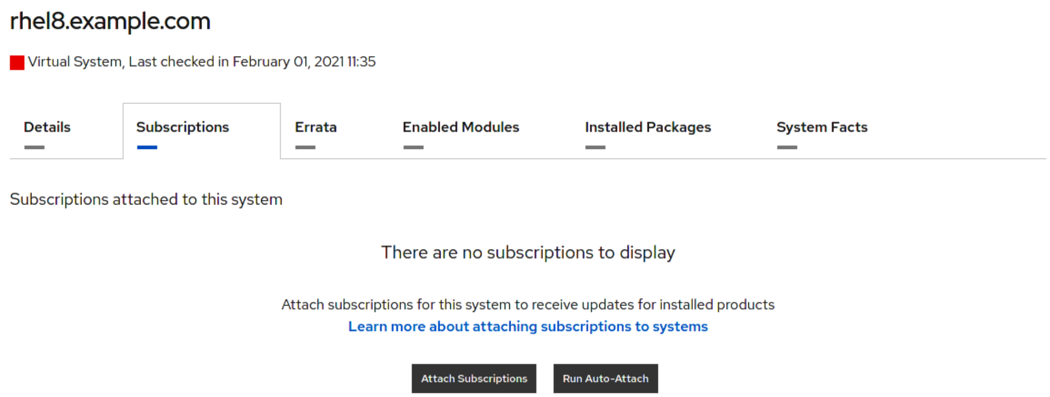


Figure 7.9 – Subscriptions page with the new subscribed system, with no subscription attached

Let's attach a subscription to our system using `subscription-manager attach`:

```
[root@rhel8 ~]# subscription-manager attach --auto
Installed Product Current Status:
Product Name: Red Hat Enterprise Linux for x86_64
Status:         Subscribed
```

The result of the command shows that the system is now registered and has a subscription attached to it for `Red Hat Enterprise Linux for x86_64`. Let's refresh the page for the system to ensure the subscription attachment runs properly:

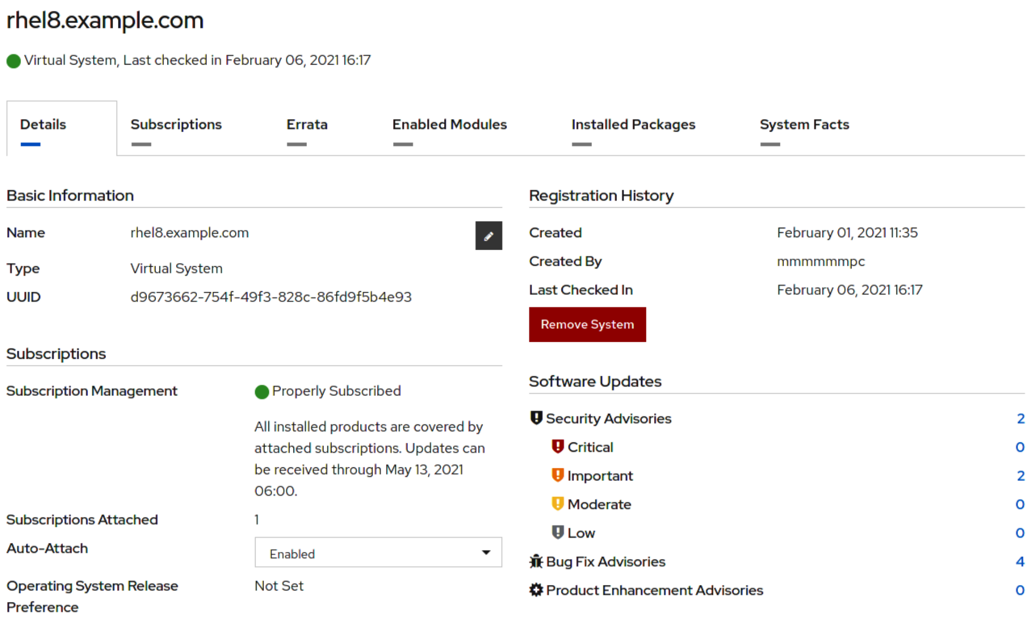


Figure 7.10 – Subscriptions page with the new subscribed system, with one subscription attached

With this, we know for sure that the system is properly registered and subscribed to the Red Hat CDN and it's ready to access all the software, patches, and updates available from it.

Also, in the system, we can see that a new file with the information on the software **repositories**, or **repos** for short, has been created:

```
[root@rhel8 ~]# ls -l /etc/yum.repos.d/redhat.repo
-rw-r--r--. 1 root root 94154 feb  6 15:17 /etc/yum.repos.d/
redhat.repo
```

Now we know how to manage available subscriptions and assign them to a running system so it has access to the software binaries built by Red Hat. Let's learn more about how to use the provided repositories in the next section.

Managing repositories and signatures with YUM/DNF

RHEL like many other Linux distributions, has a mechanism to provide software based on repos. These contain a list of software packages (which could be end user applications such as Firefox, or components for them such as GTK3), a list of dependencies between the packages, and other useful metadata.

Once we finish subscribing the system, we can take a look at the repositories available in the system using `yum` or `dnf`:

```
[root@rhel8 ~]# yum repolist
Updating Subscription Management repositories.
repo id                              repo name
rhel-8-for-x86_64-appstream-rpms     Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
rhel-8-for-x86_64-baseos-rpms        Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
[root@rhel8 ~]# dnf repolist
Updating Subscription Management repositories.
repo id                              repo name
rhel-8-for-x86_64-appstream-rpms     Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
rhel-8-for-x86_64-baseos-rpms        Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
```

As you can see, the output of `yum` and `dnf` is exactly the same. As a matter of fact, `dnf` is the evolution of `yum`, and in RHEL8 the `yum` command is just a symbolic link to `dnf`:

```
[root@rhel8 ~]# which yum
/usr/bin/yum
[root@rhel8 ~]# ll /usr/bin/yum
lrwxrwxrwx. 1 root root 5 jul 29  2020 /usr/bin/yum -> dnf-3
[root@rhel8 ~]# which dnf
```

```
/usr/bin/dnf
[root@rhel8 ~]# ll /usr/bin/dnf
lrwxrwxrwx. 1 root root 5 jul 29  2020 /usr/bin/dnf -> dnf-3
```

They both can be used indistinctively in RHEL8. From now on, we will use `dnf` only, but remember, if you prefer `yum`, feel free to use it.

> **Tip**
>
> **YUM** used to be an acronym for **Yellowdog Updater Modified**, a project that started as a Linux distribution for Macs called Yellowdog. **DNF** stands for **Dandified YUM**.

Now let's take a look at the repository definition created during the subscription attachment `/etc/yum.repos.d/redhat.repo`. We can edit the file and go to the entry of the `BaseOS` repository shown above `rhel-8-for-x86_64-baseos-rpms`:

```
[rhel-8-for-x86_64-baseos-rpms]
name = Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
baseurl = https://cdn.redhat.com/content/dist/
rhel8/$releasever/x86_64/baseos/os
enabled = 1
gpgcheck = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
sslverify = 1
sslcacert = /etc/rhsm/ca/redhat-uep.pem
sslclientkey = /etc/pki/entitlement/7881187918683323950-key.pem
sslclientcert = /etc/pki/entitlement/7881187918683323950.pem
metadata_expire = 86400
enabled_metadata = 1
```

As you can see, every section in the file starts with the name of the section between brackets – in the previous case, `[rhel-8-for-x86_64-baseos-rpms]`. Now let's check all the entries under this section:

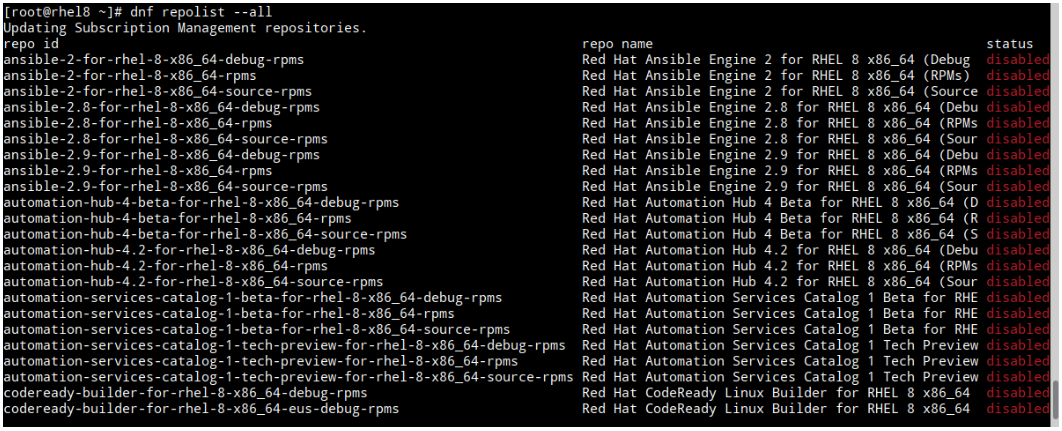
- **name**: A long descriptive name for the repository. It's the one displayed when we listed the repos in the previous example.
- **baseurl**: The main resource the packages will be obtained from. In this case, it is a single HTTPS source. It contains the `$releasever` variable that will be substituted before being accessed. Other methods are NFS, HTTP, and FTP.

- **enabled**: A variable to provide a simple way to have repositories enabled or disabled in the system. When set to `1`, it will be enabled, and when set to `0` it will be disabled.
- **gpgcheck**: Mechanism to verify the packages. When set to `1` it will be enabled and all the packages installed in the system with `dnf` / `yum` will be verified using their `gpg` signatures against a provided key.
- **gpgkey**: A key to verify, using `gpg`, the downloaded packages.
- **sslverify**: A mechanism to verify the machine subscription to the CDN. Enabled when set to `1`, and disabled when set to `0`.
- **sslcacert**: A certificate used as a Certificate Authority to validate client certificates.
- **sslclient key**: A client key used to habilitate the client certificate.
- **sslclientcert**: The client certificate that the machine will use to identify itself against the CDN.
- **metadata_expire**: The time in seconds after which to consider the retrieved metadata as expired. The default, as shown here, is 24 hours.
- **enabled_metadata**: An option to enable other tools (not `dnf`) to use the metadata as downloaded in this repository.

The minimal required options to have a running repository are: `name`, `baseurl`, and `gpgckeck`, having this last one set to `0`.

> **Important note**
> Although it is possible to change the configuration of the repositories by editing the files, the best way to modify the Red Hat provided repos is by using commands that will be shown in this chapter. That's because the `redhat.repo` file will be overwritten by the subscription manager when refreshing data.

By running `dnf repolist`, we obtained the list of repositories `enabled` in the system. What if we want to see all the repositories, the ones enabled and the ones disabled? That can be done by running `dnf repolist --all`:



```
[root@rhel8 ~]# dnf repolist --all
Updating Subscription Management repositories.
repo id                                                            repo name                                         status
ansible-2-for-rhel-8-x86_64-debug-rpms                             Red Hat Ansible Engine 2 for RHEL 8 x86_64 (Debug  disabled
ansible-2-for-rhel-8-x86_64-rpms                                   Red Hat Ansible Engine 2 for RHEL 8 x86_64 (RPMs)  disabled
ansible-2-for-rhel-8-x86_64-source-rpms                            Red Hat Ansible Engine 2 for RHEL 8 x86_64 (Source disabled
ansible-2.8-for-rhel-8-x86_64-debug-rpms                           Red Hat Ansible Engine 2.8 for RHEL 8 x86_64 (Debu disabled
ansible-2.8-for-rhel-8-x86_64-rpms                                 Red Hat Ansible Engine 2.8 for RHEL 8 x86_64 (RPMs disabled
ansible-2.8-for-rhel-8-x86_64-source-rpms                          Red Hat Ansible Engine 2.8 for RHEL 8 x86_64 (Sour disabled
ansible-2.9-for-rhel-8-x86_64-debug-rpms                           Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (Debu disabled
ansible-2.9-for-rhel-8-x86_64-rpms                                 Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (RPMs disabled
ansible-2.9-for-rhel-8-x86_64-source-rpms                          Red Hat Ansible Engine 2.9 for RHEL 8 x86_64 (Sour disabled
automation-hub-4-beta-for-rhel-8-x86_64-debug-rpms                 Red Hat Automation Hub 4 Beta for RHEL 8 x86_64 (D disabled
automation-hub-4-beta-for-rhel-8-x86_64-rpms                       Red Hat Automation Hub 4 Beta for RHEL 8 x86_64 (R disabled
automation-hub-4-beta-for-rhel-8-x86_64-source-rpms                Red Hat Automation Hub 4 Beta for RHEL 8 x86_64 (S disabled
automation-hub-4.2-for-rhel-8-x86_64-debug-rpms                    Red Hat Automation Hub 4.2 for RHEL 8 x86_64 (Debu disabled
automation-hub-4.2-for-rhel-8-x86_64-rpms                          Red Hat Automation Hub 4.2 for RHEL 8 x86_64 (RPMs disabled
automation-hub-4.2-for-rhel-8-x86_64-source-rpms                   Red Hat Automation Hub 4.2 for RHEL 8 x86_64 (Sour disabled
automation-services-catalog-1-beta-for-rhel-8-x86_64-debug-rpms    Red Hat Automation Services Catalog 1 Beta for RHE disabled
automation-services-catalog-1-beta-for-rhel-8-x86_64-rpms          Red Hat Automation Services Catalog 1 Beta for RHE disabled
automation-services-catalog-1-beta-for-rhel-8-x86_64-source-rpms   Red Hat Automation Services Catalog 1 Beta for RHE disabled
automation-services-catalog-1-tech-preview-for-rhel-8-x86_64-debug-rpms Red Hat Automation Services Catalog 1 Tech Preview disabled
automation-services-catalog-1-tech-preview-for-rhel-8-x86_64-rpms  Red Hat Automation Services Catalog 1 Tech Preview disabled
automation-services-catalog-1-tech-preview-for-rhel-8-x86_64-source-rpms Red Hat Automation Services Catalog 1 Tech Preview disabled
codeready-builder-for-rhel-8-x86_64-debug-rpms                     Red Hat CodeReady Linux Builder for RHEL 8 x86_64  disabled
codeready-builder-for-rhel-8-x86_64-eus-debug-rpms                 Red Hat CodeReady Linux Builder for RHEL 8 x86_64  disabled
```

Figure 7.11 – Partial output of dnf repolist –all

The list is very extensive. It includes the repositories with the binaries as used in many production cases, from SAP to managing systems with Satellite. We can filter it with `grep` to search for `supplementary`:

```
[root@rhel8 ~]# dnf repolist --all | grep supplementary
rhel-8-for-x86_64-supplementary-debug-rpms
disabled
rhel-8-for-x86_64-supplementary-eus-debug-rpms
disabled
rhel-8-for-x86_64-supplementary-eus-rpms
disabled
rhel-8-for-x86_64-supplementary-eus-source-rpms
disabled
rhel-8-for-x86_64-supplementary-rpms
disabled
rhel-8-for-x86_64-supplementary-source-rpms
disabled
```

There are four different types of channels here:

- **Regular channels**: For example, `rhel-8-for-x86_64-supplementary-rpms`, which contains the packages, also known as `rpms`, ready to be installed in the system. These apply during the standard maintenance period.
- **Extended update support**: For example, `rhel-8-for-x86_64-supplementary-eus-rpms`, which contains `eus` in the name. These provide packages with backports to be able to keep the same minor version for a longer time. Don't use them unless it is required by a third-party vendor.
- **Source channels**: For example, `rhel-8-for-x86_64-supplementary-source-rpms`, which contains `source` in the name. They provide the source used to build the packages delivered in *regular* and *extended updated support* channels.
- **Debug channels**: For example, `rhel-8-for-x86_64-supplementary-debug-rpms`, which contain `debug` in their name. These include the debugging information as generated when building a package useful for deep troubleshooting of issues.

We can enable `rhel-8-for-x86_64-supplementary-rpms` by using the `config-manager` option of `dnf`, running the following:

```
[root@rhel8 ~]# dnf config-manager --enable rhel-8-for-x86_64-
supplementary-rpms
Updating Subscription Management repositories.
[root@rhel8 ~]# dnf repolist
Updating Subscription Management repositories.
repo id                                                  repo name
rhel-8-for-x86_64-appstream-rpms                         Red Hat
Enterprise Linux 8 for x86_64 - AppStream (RPMs)
rhel-8-for-x86_64-baseos-rpms                            Red Hat
Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
rhel-8-for-x86_64-supplementary-rpms                     Red Hat
Enterprise Linux 8 for x86_64 - Supplementary (RPMs)
```

The repository is now enabled. You may want to try to enable and disable other repositories to practice.

Let's now try to add a repository for which we know only its URL, such as the **EPEL** repo. This repo contains **Extra Packages for Enterprise Linux**, and is specifically built for Linux, but not supported by Red Hat. As it is a well-known repo, it is replicated around the world and there is a local mirror at `http://mirror.uv.es/mirror/fedora-epel/8/Everything/x86_64/` (you can find your local one in the mirror list here: `https://admin.fedoraproject.org/mirrormanager/mirrors/EPEL`). Now we can add this repo using `dnf config-manager`:

```
[root@rhel8 ~]# dnf config-manager --add-repo="http://mirror.uv.es/mirror/fedora-epel/8/Everything/x86_64/"
Updating Subscription Management repositories.
Adding repo from: http://mirror.uv.es/mirror/fedora-epel/8/Everything/x86_64/
[root@rhel8 ~]# dnf repolist
Updating Subscription Management repositories.
repo id                                                     repo name
mirror.uv.es_mirror_fedora-epel_8_Everything_x86_64_ created by dnf config-manager from http://mirror.uv.es/mirror/fedora-epel/8/Everything/x86_64/
rhel-8-for-x86_64-appstream-rpms                            Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
rhel-8-for-x86_64-baseos-rpms                               Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
rhel-8-for-x86_64-supplementary-rpms                        Red Hat Enterprise Linux 8 for x86_64 - Supplementary (RPMs)
```

We can inspect the newly created file – `/etc/yum.repos.d/mirror.uv.es_mirror_fedora-epel_8_Everything_x86_64_.repo`:

```
[mirror.uv.es_mirror_fedora-epel_8_Everything_x86_64_]
name=created by dnf config-manager from http://mirror.uv.es/mirror/fedora-epel/8/Everything/x86_64/
baseurl=http://mirror.uv.es/mirror/fedora-epel/8/Everything/x86_64/
enabled=1
```

You may have realized that there is an option missing in this repo, however, let's move ahead. I can search for a package available in EPEL, such as, for example, `screen`:

```
[root@rhel8 ~]# dnf info screen
Updating Subscription Management repositories.
created by dnf config-manager from http://mirror.uv.es/mirror/
fedor  18 MB/s | 8.9 MB     00:00
Last metadata expiration check: 0:00:02 ago on sáb 13 feb 2021
15:34:56 CET.
Available Packages
Name         : screen
Version      : 4.6.2
Release      : 10.el8
Architecture : x86_64
Size         : 582 k
Source       : screen-4.6.2-10.el8.src.rpm
Repository   : mirror.uv.es_mirror_fedora-epel_8_Everything_
x86_64_
Summary      : A screen manager that supports multiple logins
on one terminal
URL          : http://www.gnu.org/software/screen
License      : GPLv3+
Description  : The screen utility allows you to have multiple
logins on just one
             : terminal. Screen is useful for users who telnet
into a machine or are
             : connected via a dumb terminal, but want to use
more than just one
             : login.
             :
             : Install the screen package if you need a screen
manager that can
             : support multiple logins on one terminal.
```

The package is found, now let's try to install it:

```
[root@rhel8 ~]# dnf install screen
[omitted]
Install  1 Package

Total download size: 582 k
Installed size: 971 k
Is this ok [y/N]: y
Downloading Packages:
screen-4.6.2-10.el8.x86_64.rpm
2.8 MB/s | 582 kB     00:00
---------------------------------------------------------------
-------------------------------------
Total
2.8 MB/s | 582 kB     00:00
warning: /var/cache/dnf/mirror.uv.es_mirror_fedora-epel_8_
Everything_x86_64_-ee39120d2e2a3152/packages/screen-4.6.2-
10.el8.x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID
2f86d6a1: NOKEY
Public key for screen-4.6.2-10.el8.x86_64.rpm is not installed
The downloaded packages were saved in cache until the next
successful transaction.
You can remove cached packages by executing 'yum clean
packages'.
Error: GPG check FAILED
```

As we can see there is an error trying to install from this source, as it required the `gpgcheck` and `gpgkey` entries to be configured to have a properly secured repo (as `gpg` ensures that the content delivered is the same as the content created).

We can obtain `gpgkey` from the same mirror, at the URL `http://mirror.uv.es/mirror/fedora-epel/RPM-GPG-KEY-EPEL-8`, and drop it where `dnf` will search for it, `/etc/pki/rpm-gpg/`:

```
[root@rhel8 ~]# curl -s http://mirror.uv.es/mirror/fedora-epel/
RPM-GPG-KEY-EPEL-8 > /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-8
[root@rhel8 ~]# head -n 1 /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-8
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

Now let's modify the file `/etc/yum.repos.d/mirror.uv.es_mirror_fedora-epel_8_Everything_x86_64_.repo` to look like this:

```
[mirror.uv.es_mirror_fedora-epel_8_Everything_x86_64_]
name=created by dnf config-manager from http://mirror.uv.es/mirror/fedora-epel/8/Everything/x86_64/
baseurl=http://mirror.uv.es/mirror/fedora-epel/8/Everything/x86_64/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-8
```

You can see we added both the `gpgcheck` and `gpgkey` entries in the file. Let's try installing the `screen` package again:

```
[root@rhel8 ~]# dnf install screen
[omitted]
Install  1 Package

Total size: 582 k
Installed size: 971 k
Is this ok [y/N]: y
Downloading Packages:
[SKIPPED] screen-4.6.2-10.el8.x86_64.rpm: Already downloaded
warning: /var/cache/dnf/mirror.uv.es_mirror_fedora-epel_8_Everything_x86_64_-ee39120d2e2a3152/packages/screen-4.6.2-10.el8.x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID 2f86d6a1: NOKEY
created by dnf config-manager from http://mirror.uv.es/mirror/fedor 1.6 MB/s | 1.6 kB     00:00
Importing GPG key 0x2F86D6A1:
Userid     : "Fedora EPEL (8) <epel@fedoraproject.org>"
Fingerprint: 94E2 79EB 8D8F 25B2 1810 ADF1 21EA 45AB 2F86 D6A1
From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-8
Is this ok [y/N]: y
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
```

```
Transaction test succeeded.
Running transaction
  Preparing        :                                1/1
  Running scriptlet: screen 4.6.2-10.el8.x86_64     1/1
  Installing       : screen-4.6.2-10.el8.x86_64     1/1
  Running scriptlet: screen-4.6.2-10.el8.x86_64     1/1
  Verifying        : screen-4.6.2-10.el8.x86_64     1/1
Installed products updated.

Installed:
  screen-4.6.2-10.el8.x86_64

Complete!
```

You'll notice that there is a step in which it asks you to confirm that the `gpg` key fingerprint is correct: `94E2 79EB 8D8F 25B2 1810 ADF1 21EA 45AB 2F86 D6A1`. To do so, you can go to the Fedora security page, as the Fedora project is managing EPEL, and check. The URL for that page is `https://getfedora.org/security/`:

Figure 7.12 – Partial capture of the Fedora security page with an EPEL8 gpg fingerprint

As you can see, it is correct. We have just verified that the signature we are using has the same fingerprint as announced by the project managing it, and now all the packages downloaded from this repo will be verified with it to avoid package tampering (which is someone changing the content before you receive it).

Let's review the commands that we used, which `dnf` provides to manage repos:

Command	Usage
`dnf repolist`	Shows repos configured and enabled
`dnf repolist --all`	Shows repos configured, enabled, or disabled
`dnf config-manager --enable <reponame>`	Enables an existing repo
`dnf config-manager --add-repo=<repo_url>`	Adds a new repo with the provided URL

Now that we know how to securely manage repositories in RHEL, let's start adding more packages to our system, updating them, and undoing installations if we need to.

Doing software installations, updates, and rollbacks with YUM/DNF

In the previous section, we saw how to install a package. During the process, we saw a confirmation request to ensure that we were sure that we wanted to include new software in the system. Let's now install software with `dnf install`, but using the `-y` option to answer yes to all questions that the command will issue:

```
[root@rhel8 ~]# dnf install zip -y
[omitted]
Installed:
unzip-6.0-43.el8.x86_64 zip-3.0-23.el8.x86_64

Complete!
```

As you can see, the `zip` package was installed, and also a dependent package called `unzip`, without asking questions. We also notice that `dnf` finds the dependent packages, resolves the **dependencies**, and installs all that is needed for a package to run. This way, the system is kept in a coherent status, making it more reliable and predictable.

We cam see which packages are ready to be updated using the `dnf check-update` command:

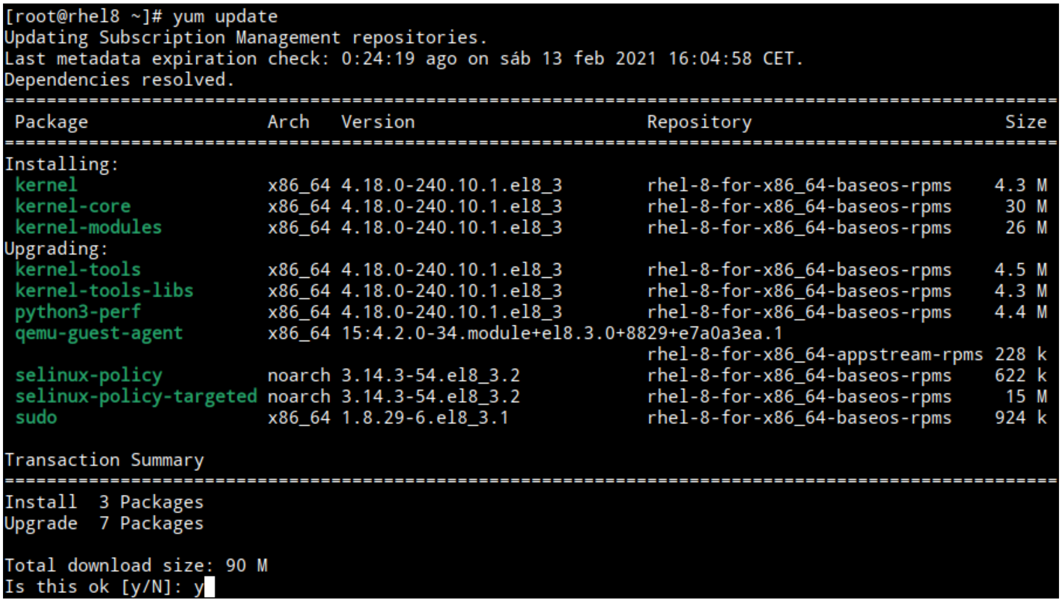
```
[root@rhel8 ~]# dnf check-update
Updating Subscription Management repositories.
Last metadata expiration check: 0:20:00 ago on sáb 13 feb 2021
16:04:58 CET.

kernel.x86_64           4.18.0-240.10.1.el8_3            rhel-
8-for-x86_64-baseos-rpms
kernel-core.x86_64      4.18.0-240.10.1.el8_3           rhel-8-
for-x86_64-baseos-rpms
kernel-modules.x86_64   4.18.0-240.10.1.el8_3       rhel-8-for-
x86_64-baseos-rpms
kernel-tools.x86_64      4.18.0-240.10.1.el8_3     rhel-8-for-
x86_64-baseos-rpms
kernel-tools-libs.x86_64 4.18.0-240.10.1.el8_3     rhel-8-for-
x86_64-baseos-rpms
python3-perf.x86_64      4.18.0-240.10.1.el8_3         rhel-8-
for-x86_64-baseos-rpms
qemu-guest-agent.x86_64  15:4.2.0-34.module+el8.3.0+8829+e7a0a
3ea.1          rhel-8-for-x86_64-appstream-rpms
selinux-policy.noarch    3.14.3-54.el8_3.2         rhel-8-for-
x86_64-baseos-rpms
selinux-policy-targeted.noarch   3.14.3-54.el8_3.2 rhel-8-for-
x86_64-baseos-rpms
sudo.x86_64              1.8.29-6.el8_3.1                 rhel-
8-for-x86_64-baseos-rpms
tzdata.noarch           2021a-1.el8                      rhel-
8-for-x86_64-baseos-rpms
```

The easiest way to update a package and apply fixes and security patches is to use `dnf update`:

```
[root@rhel8 ~]# dnf update tzdata -y
[omitted]
Upgraded:
  tzdata-2021a-1.el8.noarch
Complete!
```

To update everything, just run `dnf update` without specifying the package:



```
[root@rhel8 ~]# yum update
Updating Subscription Management repositories.
Last metadata expiration check: 0:24:19 ago on sáb 13 feb 2021 16:04:58 CET.
Dependencies resolved.
================================================================================================
 Package                  Arch   Version                         Repository                  Size
================================================================================================
Installing:
 kernel                   x86_64 4.18.0-240.10.1.el8_3           rhel-8-for-x86_64-baseos-rpms    4.3 M
 kernel-core              x86_64 4.18.0-240.10.1.el8_3           rhel-8-for-x86_64-baseos-rpms     30 M
 kernel-modules           x86_64 4.18.0-240.10.1.el8_3           rhel-8-for-x86_64-baseos-rpms     26 M
Upgrading:
 kernel-tools             x86_64 4.18.0-240.10.1.el8_3           rhel-8-for-x86_64-baseos-rpms    4.5 M
 kernel-tools-libs        x86_64 4.18.0-240.10.1.el8_3           rhel-8-for-x86_64-baseos-rpms    4.3 M
 python3-perf             x86_64 4.18.0-240.10.1.el8_3           rhel-8-for-x86_64-baseos-rpms    4.4 M
 qemu-guest-agent         x86_64 15:4.2.0-34.module+el8.3.0+8829+e7a0a3ea.1
                                                                 rhel-8-for-x86_64-appstream-rpms 228 k
 selinux-policy           noarch 3.14.3-54.el8_3.2               rhel-8-for-x86_64-baseos-rpms    622 k
 selinux-policy-targeted  noarch 3.14.3-54.el8_3.2               rhel-8-for-x86_64-baseos-rpms     15 M
 sudo                     x86_64 1.8.29-6.el8_3.1                rhel-8-for-x86_64-baseos-rpms    924 k

Transaction Summary
================================================================================================
Install   3 Packages
Upgrade   7 Packages

Total download size: 90 M
Is this ok [y/N]: y
```

Figure 7.13 – Partial capture of RHEL updating with dnf/yum

The result of running `dnf update` in the system is the following:

```
Upgraded:
   kernel-tools-4.18.0-240.10.1.el8_3.x86_64
   kernel-tools-libs-4.18.0-240.10.1.el8_3.x86_64
   python3-perf-4.18.0-240.10.1.el8_3.x86_64
   qemu-guest-agent 15:4.2.0-34.module+el8.3.0+8829+e7a0a3ea.1.
x86_64
   selinux-policy-3.14.3-54.el8_3.2.noarch
   selinux-policy-targeted-3.14.3-54.el8_3.2.noarch
   sudo-1.8.29-6.el8_3.1.x86_64

Installed:
   kernel-4.18.0-240.10.1.el8_3.x86_64
   kernel-core-4.18.0-240.10.1.el8_3.x86_64
   kernel-modules-4.18.0-240.10.1.el8_3.x86_64

Complete!
```

These were examples of packages upgraded in a system. Your system, depending on the time you last upgraded it and the newly released packages, may have a different output.

> **Important note**
>
> The `kernel` is the most important part of the system. It enables hardware access and all the basic functions of the operating system. That's why, instead of upgrading it, a new version is installed. The system keeps the previous two versions just in case the system becomes unbootable, and one of them can be selected to run easily.

We can search the packages available with the `dnf search` command:

```
[root@rhel8 ~]# dnf search wget
Updating Subscription Management repositories.
Last metadata expiration check: 0:05:02 ago on sáb 13 feb 2021
16:34:00 CET.
=================== Name Exactly Matched: wget
===================
wget.x86_64 : A utility for retrieving files using the HTTP or
FTP protocols
```

We can obtain extended information about a package, installed or not, with `dnf info`:

```
[root@rhel8 ~]# dnf info wget
Updating Subscription Management repositories.
Last metadata expiration check: 0:06:45 ago on sáb 13 feb 2021
16:34:00 CET.
Available Packages
Name         : wget
Version      : 1.19.5
Release      : 10.el8
Architecture : x86_64
Size         : 734 k
Source       : wget-1.19.5-10.el8.src.rpm
Repository   : rhel-8-for-x86_64-appstream-rpms
Summary      : A utility for retrieving files using the HTTP or
FTP protocols
URL          : http://www.gnu.org/software/wget/
License      : GPLv3+
Description  : GNU Wget is a file retrieval utility which can
```

```
use either the HTTP or
             : FTP protocols. Wget features include the ability
to work in the
             : background while you are logged out, recursive
retrieval of
             : directories, file name wildcard matching, remote
file timestamp
             : storage and comparison, use of Rest with FTP
servers and Range with
             : HTTP servers to retrieve files over slow or
unstable connections,
             : support for Proxy servers, and configurability.
```

We can also remove an installed package with `dnf remove`:

```
[root@rhel8 ~]# dnf remove screen -y
[omitted]
Removed:  screen-4.6.2-10.el8.x86_64
Complete!
```

Sometimes you want to install some packages that come together to perform a specific task, and that's what **package groups** are for. Let's get a list of the groups first with `dnf grouplist`:

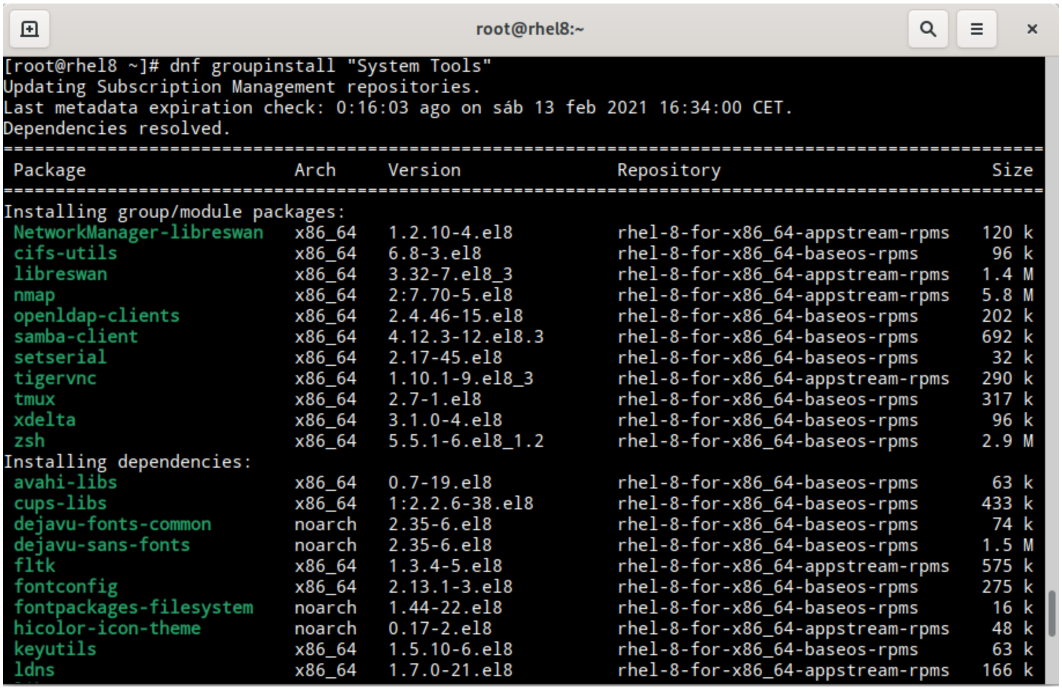
```
[root@rhel8 ~]# dnf grouplist | grep Tools
   Additional Virtualization Tools
   RPM Development Tools
   Security Tools
   Development Tools
   System Tools
   Graphical Administration Tools
```

You may run it without `| grep Tools` to see the full list.

Let's install the `System Tools` group with `dnf groupinstall`:

```
[root@rhel8 ~]# dnf groupinstall "System Tools"
Updating Subscription Management repositories.
Last metadata expiration check: 0:16:03 ago on sáb 13 feb 2021
16:34:00 CET.
Dependencies resolved.
```

The entire output of the preceding command is shown in the following screenshot:



```
root@rhel8:~
[root@rhel8 ~]# dnf groupinstall "System Tools"
Updating Subscription Management repositories.
Last metadata expiration check: 0:16:03 ago on sáb 13 feb 2021 16:34:00 CET.
Dependencies resolved.
================================================================================================
 Package                    Arch     Version              Repository                       Size
================================================================================================
Installing group/module packages:
 NetworkManager-libreswan   x86_64   1.2.10-4.el8         rhel-8-for-x86_64-appstream-rpms   120 k
 cifs-utils                 x86_64   6.8-3.el8            rhel-8-for-x86_64-baseos-rpms       96 k
 libreswan                  x86_64   3.32-7.el8_3         rhel-8-for-x86_64-appstream-rpms   1.4 M
 nmap                       x86_64   2:7.70-5.el8         rhel-8-for-x86_64-appstream-rpms   5.8 M
 openldap-clients           x86_64   2.4.46-15.el8        rhel-8-for-x86_64-baseos-rpms      202 k
 samba-client               x86_64   4.12.3-12.el8.3      rhel-8-for-x86_64-baseos-rpms      692 k
 setserial                  x86_64   2.17-45.el8          rhel-8-for-x86_64-baseos-rpms       32 k
 tigervnc                   x86_64   1.10.1-9.el8_3       rhel-8-for-x86_64-appstream-rpms   290 k
 tmux                       x86_64   2.7-1.el8            rhel-8-for-x86_64-baseos-rpms      317 k
 xdelta                     x86_64   3.1.0-4.el8          rhel-8-for-x86_64-baseos-rpms       96 k
 zsh                        x86_64   5.5.1-6.el8_1.2      rhel-8-for-x86_64-baseos-rpms      2.9 M
Installing dependencies:
 avahi-libs                 x86_64   0.7-19.el8           rhel-8-for-x86_64-baseos-rpms       63 k
 cups-libs                  x86_64   1:2.2.6-38.el8       rhel-8-for-x86_64-baseos-rpms      433 k
 dejavu-fonts-common        noarch   2.35-6.el8           rhel-8-for-x86_64-baseos-rpms       74 k
 dejavu-sans-fonts          noarch   2.35-6.el8           rhel-8-for-x86_64-baseos-rpms      1.5 M
 fltk                       x86_64   1.3.4-5.el8          rhel-8-for-x86_64-appstream-rpms   575 k
 fontconfig                 x86_64   2.13.1-3.el8         rhel-8-for-x86_64-baseos-rpms      275 k
 fontpackages-filesystem    noarch   1.44-22.el8          rhel-8-for-x86_64-baseos-rpms       16 k
 hicolor-icon-theme         noarch   0.17-2.el8           rhel-8-for-x86_64-appstream-rpms    48 k
 keyutils                   x86_64   1.5.10-6.el8         rhel-8-for-x86_64-baseos-rpms       63 k
 ldns                       x86_64   1.7.0-21.el8         rhel-8-for-x86_64-appstream-rpms   166 k
```

Figure 7.14 – Partial capture of RHEL installing a group dnf/yum

Once the preinstallation completes, we can see that we will install 78 packages:

```
Install  78 Packages

Total download size: 44 M
Installed size: 141 M
Is this ok [y/N]:y
```

Replying with `y` will perform the installation (note that the `-y` option works here too, assuming yes to all questions).

We can check the history of all the installation transactions with `dnf history`:

```
root@rhel8:~
[root@rhel8 ~]# dnf history
Updating Subscription Management repositories.
ID     | Command line                                     | Date and time    | Action(s)      | Altered
-------------------------------------------------------------------------------------------------------
    15 | groupinstall System Tools                        | 2021-02-13 16:51 | Install        |   79
    14 | remove screen                                    | 2021-02-13 16:44 | Removed        |    1
    13 | update                                           | 2021-02-13 16:30 | I, U           |   10
    12 | update tzdata                                    | 2021-02-13 16:27 | Upgrade        |    1
    11 | install zip -y                                   | 2021-02-13 16:15 | Install        |    2
    10 | install screen                                   | 2021-02-13 16:05 | Install        |    1
     9 | remove screen                                    | 2021-02-13 16:04 | Removed        |    1
     8 | install screen                                   | 2021-02-13 15:51 | Install        |    1
     7 | remove screen                                    | 2021-02-07 15:59 | Removed        |    1
     6 | install screen                                   | 2021-02-07 15:58 | Install        |    1
     5 | install tar                                      | 2020-12-27 16:07 | Install        |    1
     4 | remove bash-completion                           | 2020-12-26 12:40 | Removed        |    5
     3 | install bash-completion -y                       | 2020-12-26 10:58 | Install        |    5
     2 | update                                           | 2020-12-26 10:50 | I, U           |   44
     1 |                                                  | 2020-12-07 16:41 | Install        |  389 EE
[root@rhel8 ~]#
```

Figure 7.15 – Partial capture of RHEL dnf/yum history

It's easy to obtain specific info from each transaction specifying the number of it to `dnf history`:

```
[root@rhel8 ~]# dnf history info 12
Updating Subscription Management repositories.
Transaction ID : 12
Begin time     : sáb 13 feb 2021 16:27:06 CET
Begin rpmdb    : 393:cec089e1c176497af3eb97582311fcd7cb7adb02
End time       : sáb 13 feb 2021 16:27:06 CET (0 seconds)
End rpmdb      : 393:6cf80ca6746149100bb1a49d76ebbf7407804e56
User           : root <root>
Return-Code    : Success
Releasever     : 8
Command Line   : update tzdata
Comment        :
Packages Altered:
    Upgrade  tzdata-2021a-1.el8.noarch @rhel-8-for-x86_64-baseos-rpms
    Upgraded tzdata-2020d-1.el8.noarch @@System
```

More interestingly, we can roll back to one of the previous points marked with `dnf history rollback`. To make it faster, *install* the `lsof` package and then *roll back* to the previous number:

```
[root@rhel8 ~]# dnf history rollback 15
[omitted]
Removed:  lsof-4.93.2-1.el8.x86_64
Complete!
```

We can also undo a single transaction with `yum history undo`. Let's see it with this transaction:

```
[root@rhel8 ~]# dnf history undo 10 -y
[omitted]
Removed:
   screen-4.6.2-10.el8.x86_64
Complete!
```

Let's review the most important transactions done with `dnf`:

Command	**Usage**
`dnf search`	Search for a package with a given name.
`dnf info`	Shows information of a package whether installed or not.
`dnf install`	Installs the packages mentioned and their dependencies.
`dnf install -y`	Installs packages and dependencies assuming yes to all questions.
`dnf remove`	Removes a specified package and the dependencies that would become orphans.
`dnf check-update`	Shows the packages that have an update available.
`dnf update`	Updates all packages that have a newer version available.
`dnf grouplist`	Shows a list of the groups available, whether installed or not.

Command	Usage
`dnf groupinstall`	Installs a group of packages.
`dnf history`	Shows the list of transactions done with `dnf`.
`dnf history info`	Provides info on a specific transaction.
`dnf history rollback`	Undoes all transactions done up to a specified point.
`dnf history undo`	Undoes a specific transaction.

There is a new feature on RHEL 8 that wasn't available in the previous version, which is **modularity**. It enables the possibility to have different versions for the same package in the system. It is all managed with `dnf` so there is no need to install extra software:

```
[root@rhel8 repos]# dnf module list postgresql
Updating Subscription Management repositories.
Last metadata expiration check: 0:00:30 ago on dom 14 feb 2021
19:25:32 CET.
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
Name                Stream        Profiles                    Summ
ary
postgresql          9.6           client, server [d]
PostgreSQL server and client module
postgresql          10 [d]        client, server [d]
PostgreSQL server and client module
postgresql          12            client, server [d]
PostgreSQL server and client module

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

> **Tip**
>
> The `dnf module list` command, without specifying any package, would show the full list of modules. Try it!

As you can see, we have three different versions of the PostgreSQL database available in RHEL8, which are 9.6, 10, and 12. None of them are enabled and the default one is version 10.

Let's enable version 12 of PostgreSQL using `dnf module`:

```
[root@rhel8 ~]# dnf module enable postgresql:12
[omitted]
Enabling module streams: postgresql                  12
[omitted]
Is this ok [y/N]: y
Complete!
[root@rhel8 ~]# dnf module list postgresql
```

The output of the preceding command can be seen in the following screenshot:

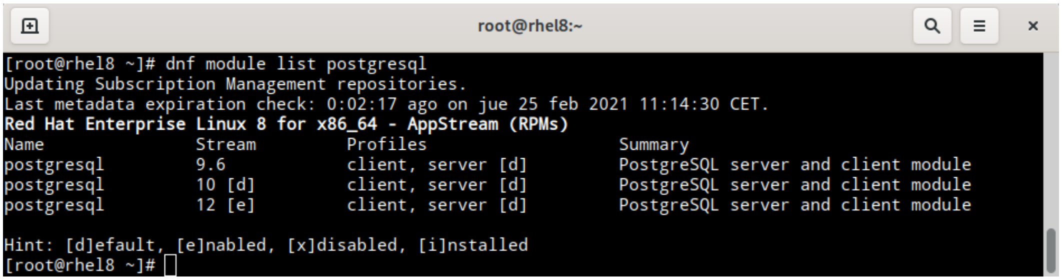


Figure 7.16 – Capture of the PostgreSQL module list

From now on Yum will install, update, and maintain in this system version 12 of PostgreSQL. Let's install it:

```
[root@rhel8 ~]# dnf install postgresql -y
[omitted]
Installed:
  libpq-12.5-1.el8_3.x86_64
  postgresql-12.5-1.module+el8.3.0+9042+664538f4.x86_64

Complete!
```

In the previous example, version 12 was installed.

We can remove the PostgreSQL package and reset the module status to go back to the initial point:

```
[root@rhel8 ~]# dnf remove postgresql -y
[omitted]
Removing:
```

```
postgresql  x86_64  12.5-1.module+el8.3.0+9042+664538f4  @rhel-
8-for-x86_64-appstream-rpms  5.4 M
Removing unused dependencies:
libpq       x86_64  12.5-1.el8_3                         @rhel-
8-for-x86_64-appstream-rpms  719 k
[omitted]
Complete!
[root@rhel8 ~]# dnf module reset postgresql
Updating Subscription Management repositories.
Last metadata expiration check: 1:23:08 ago on dom 14 feb 2021
19:25:32 CET.
Dependencies resolved.
======================================================
===Package                Architecture          Version
Repository               Size
=========================================================
Resetting modules:
postgresql
Transaction Summary
=========================================================Is
this ok [y/N]: y
Complete!
[root@rhel8 ~]# dnf module list postgresql
Updating Subscription Management repositories.
Last metadata expiration check: 1:23:21 ago on dom 14 feb 2021
19:25:32 CET.
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
Name                Stream        Profiles                  Summ
ary
postgresql          9.6           client, server [d]
PostgreSQL server and client module
postgresql          10 [d]        client, server [d]
PostgreSQL server and client module
postgresql          12            client, server [d]
PostgreSQL server and client module

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

Let's review the commands shown in this section for modularity:

Command	Usage
`dnf module list`	Lists all modules available in the system
`dnf module enable`	Enables one specific module to work within the system
`dnf module reset`	Clears the configuration for a module

> **Tip**
> For more info on modularity, go to the system's manual page by running `man dnf.modularity`.

Now that we have learned how to handle software transactions in RHEL, let's go ahead with how to create and handle local repositories.

Creating and syncing repositories with createrepo and reposync

It's common that we receive an RPM file and keep it in a repository that we can use on our machine (and sometimes share it with other machines with a web server of an NFS share). It's also common that when we start building our own RPMs, we distribute them and, to do so, we need to create a repository. To do that, we can use the **createrepo** tool.

First let's create a folder in `/var/tmp` for repos:

```
[root@rhel8 ~]# cd /var/tmp/
[root@rhel8 tmp]# mkdir repos
[root@rhel8 tmp]# cd repos/
```

Then let's create a folder for `slack`, a common tool to communicate with your team, and download the RPM package:

```
[root@rhel8 repos]# mkdir slack
[root@rhel8 repos]# cd slack/
[root@rhel8 repos]# curl -s -O https://downloads.slack-edge.
com/linux_releases/slack-4.12.2-0.1.fc21.x86_64.rpm
[root@rhel8 slack]# ls -l
total 62652
-rw-r--r--. 1 root 64152596 feb 14 18:12 slack-4.12.2-0.1.fc21.
x86_64.rpm
```

Now we have a repository with an RPM file. We could have one with as many RPMs as we want but we will continue with only this single package.

Let's install the `createrepo` tool:

```
[root@rhel8 slack]# dnf install -y createrepo
[omitted]
Installed:
  createrepo_c-0.15.11-2.el8.x86_64 createrepo_c-libs-0.15.11-2.el8.x86_64 drpm-0.4.1-3.el8.x86_64
Complete!
```

And now we could simply run it to create a repository in the current folder with the following command:

```
[root@rhel8 slack]# createrepo .
Directory walk started
Directory walk done - 1 packages
Temporary output repo path: ./.repodata/
Preparing sqlite DBs
Pool started (with 5 workers)
Pool finished
[root@rhel8 slack]# ls -l
total 62656
drwxr-xr-x. 2 root     4096 feb 14 18:19 repodata
-rw-r--r--. 1 root 64152596 feb 14 18:12 slack-4.12.2-0.1.fc21.x86_64.rpm
```

We see that the `repodata` folder has been created. In it, we can find the `repomd.xml` file that defines the repository content and also the recently created index files:

```
[root@rhel8 slack]# ls repodata/
13b6b81deb95354164189de7fe5148b4dbdb247fb910973cc94c120d36c0fd27-filelists.xml.gz
18fb83942e8cb5633fd0653a4c8ac3db0f93ea73581f91d90be93256061043f0-other.sqlite.bz2
aa72116fa9b47caaee313ece2c16676dce26ffcc78c69dc74ebe4fc59aea2c78-filelists.sqlite.bz2
d5e2ff4b465544a423bfa28a4bc3d054f316302feab8604d64f73538809b1cf0-primary.xml.gz
```

```
e92cd0e07c758c1028054cfeb964c4e159004be61ae5217927c27d27e
a2c7966-primary.sqlite.bz2
f68973de8a710a9a078faf49e90747baaf496c5a43865cd5dc5757512a06
64a8-other.xml.gz
repomd.xml
```

Now we can add the repository to the system. We could do it without `gpg` signatures, setting the `gpgcheck` variable to `0` but, to have better security, let's do it with the `gpg` signature. By searching in the `slack` page, we find the signature and download it to the `/etc/pki/rpm-gpg` directory:

```
[root@rhel8 slack]# curl https://slack.com/gpg/slack_
pubkey_2019.gpg -o /etc/pki/rpm-gpg/RPM-GPG-KEY-SLACK
```

Then we add the repository to the system by creating the file `/etc/yum.repos.d/local-slack.repo` with the following content:

```
[local-slack-repo]
name=Local Slack Repository
baseurl=file:///var/tmp/repos/slack
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-SLACK
```

And now we can try installing `slack`. To run completely, it would require that the package group *Server with a GUI* is installed, however, for the purpose of this exercise, we can continue with its installation. We can do so by running `dnf -y install slack` – please note how the `gpg` key gets imported automatically and the package is verified and installed:

```
root@rhel8 slack]# dnf -y install slack
[omitted]
warning: /var/tmp/repos/slack/slack-4.12.2-0.1.fc21.x86_64.rpm:
Header V4 RSA/SHA1 Signature, key ID 8e6c9578: NOKEY
Local Slack Repository
1.6 MB/s | 1.6 kB     00:00
Importing GPG key 0x8E6C9578:
Userid     : "Slack Packages (Signing Key) <packages@slack-
corp.com>"
Fingerprint: 93D5 D2A6 2895 1B43 83D8 A4CE F184 6207 8E6C 9578
From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-SLACK
```

```
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
[omitted]
  slack-4.12.2-0.1.fc21.x86_64
Complete!
```

Once a new version of Slack appears, we can download it to the same folder, and regenerate the repository index by running `createrepo` again. This way all the systems using this repository will update `slack` when they run a `yum update`. It's a good way to keep all systems standardized and in the same version. For advanced features when managing RPM repositories, please check Red Hat Satellite.

Sometimes we want to have a local replica of the repositories in our system. To do that, we can use the **reposync** tool.

First, we install `reposync`, which comes in the `yum-utils` package:

```
[root@rhel8 ~]# dnf install yum-utils -y
[omitted]
Installed:
  yum-utils-4.0.17-5.el8.noarch

Complete!
```

> **Tip**
>
> If you try to install the `dnf-utils` package, this same package will be installed.

Now it's time to disable all repos provided by Red Hat except `rhel-8-for-x86_64-baseos-rpms`, which can be done with the following command:

```
[root@rhel8 ~]# subscription-manager repos --disable="*"
--enable="rhel-8-for-x86_64-baseos-rpms"
```

Time to check the change:

```
[root@rhel8 ~]# dnf repolist
Updating Subscription Management repositories.
repo id                                                    repo name
local-slack-repo                                           Local
Slack Repository
mirror.uv.es_mirror_fedora-epel_8_Everything_x86_64_ created
by dnf config-manager from http://mirror.uv.es/mirror/fedora-
epel/8/Everything/x86_64/
rhel-8-for-x86_64-baseos-rpms                              Red Hat
Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
```

We can also disable the other repos, but this time we will do it in a different way, renaming them to something that doesn't end with `.repo`:

```
[root@rhel8 ~]# mv /etc/yum.repos.d/local-slack.repo  /etc/yum.
repos.d/local-slack.repo_disabled
[root@rhel8 ~]# mv /etc/yum.repos.d/mirror.uv.es_mirror_fedora-
epel_8_Everything_x86_64_.repo  /etc/yum.repos.d/mirror.uv.es_
mirror_fedora-epel_8_Everything_x86_64_.repo_disabled
[root@rhel8 ~]# yum repolist
Updating Subscription Management repositories.
repo id                              repo name
rhel-8-for-x86_64-baseos-rpms        Red Hat Enterprise Linux
8 for x86_64 - BaseOS (RPMs)
```

Now we can run `reposync` with some options:

```
[root@rhel8 ~]# cd /var/tmp/repos
[root@rhel8 repos]# reposync --newest-only --download-metadata
--destdir /var/tmp/repos
Updating Subscription Management repositories.
[omitted]
(1725/1726): selinux-policy-3.14.3-54.el8_3.2.noarch.rpm
2.3 MB/s | 622 kB     00:00
(1726/1726): selinux-policy-devel-3.14.3-54.el8_3.2.noarch.rpm
4.1 MB/s | 1.5 MB     00:00
[root@rhel8 repos]# ls
rhel-8-for-x86_64-baseos-rpms  slack
[root@rhel8 repos]# ls rhel-8-for-x86_64-baseos-rpms/
```

```
Packages  repodata
[root@rhel8 repos]# ls rhel-8-for-x86_64-baseos-rpms/repodata/
14d4e7f9bbf5901efa7c54db513a2ac68cb0b6650ae23a2e0bff15dc035
65f25-other.sqlite.bz2
26727acbd819c59d4da7c8aeaddb027adbfb7ddb4861d31922465b4c09
22f969-updateinfo.xml.gz
46f0b974d2456ad4f66dec3afff1490648f567ee9aa4fe695494ec2cfc9a
88f6-primary.sqlite.bz2
580de0089dbaa82ca8963963da9cb74abf7a5c997842492210e2c10e1d
eac832-primary.xml.gz
5954c1ef-00bc-457b-9586-e51789358b97
a7504888345e2440fa62e21a85f690c64a5f5b9ffd84d8e525a077c955644
abe-filelists.xml.gz
acad9f7dfbc7681c2532f2fd1ff56e0f4e58eb0e2be72cc1d4a
4ec8613008699-comps.xml
d2e90d6a0f138e6d8ea190cf995902c821309a03606c7acc28857e1864899
74a-filelists.sqlite.bz2
e0a7c4b677c633b859dba5eac132de68e138223e4ad696c72a97c454f2fe7
0bd-other.xml.gz
repomd.xml
```

This will download the latest packages for the enabled channels. Let's review the options:

- `--newest-only`: Red Hat repositories keep all the versions of the packages since the first release. This will download only the latest version.
- `--download-metadata`: To be sure that we download a fully functional repo, and we do not need to run `createrepo` on it, we can use this option, which will retrieve all metadata in a source repository.
- `--destdir /var/tmp/repos`: Sets the destination directory for the downloaded files. It will also create a directory for each repo configured so the specified directory will be the parent of them all.

With this replicated repository, we can also work in isolated environments. It could be very convenient to prepare test environments. For advanced repo management features, please remember to try Red Hat Satellite.

After learning the basics of repositories and how to use them to manage software, let's dive into the technology behind it, the **Red Hat Package Manager**, or **RPM**.

Understanding RPM internals

Linux distributions tend to have their own package manager, from Debian with `.deb` to Pacman in Arch Linux and other more exotic mechanisms. The intention of the package managers is to keep software installed on the system, update it, patch it, keep dependencies, and maintain an internal database of what is installed on the system. RPM is used by distributions such as Fedora, openSUSE, CentOS, Oracle Linux, and, of course, RHEL.

To handle RPMs, the `rpm` command is available in the system, however, since the introduction of `yum/dnf`, it is hardly ever used in system administration, and is not included in RHCSA.

RPMs contain the following:

- The files to be installed on the system, stored in CPIO format and compressed
- Information on permissions and the assigned owner and group for each file
- The dependencies required and provided by each package, along with, conflicts with other packages
- Install, uninstall, and upgrade scripts to be applied in any of those phases
- A signature to ensure the package was not modified

To learn a bit about it, we will show some simple useful commands.

Commands to check packages include the following:

- `rpm -qa`: Lists all the installed packages in the system
- `rpm -qf <filename>`: Shows which package installed the mentioned filename
- `rpm -ql <packagefile>`: Lists the files included in a downloaded package (interesting to check the previously downloaded packages)

Commands to install, upgrade, and remove include the following:

- `rpm -i <packagefile>`: Installs the list of provided packages, not fetching dependencies.
- `rpm -U <packagefile>`: Upgrades a package with the downloaded one. Checks dependencies but doesn't manage them.
- `rpm -e <packagename>`: Removes the packages specified, although it won't remove dependencies.

If you want to understand how the dependency management system works in `yum/dnf`, try installing packages with `rpm -i`.

It is important to know that all the databases of installed packages are located in `/var/lib/rpm` and can be managed with the `rpmdb` command.

In modern times, having to work with the `rpm` command usually means having a low-level issue, so it's better to try to break a test system before having to use it in real life.

With this, we've completed software management in RHEL systems.

Summary

In this chapter, we have gone through the admin parts of software management in a RHEL 8 system, from subscriptions to installation, to modularity, and other miscellaneous tips.

All the system patching, updating, and management in RHEL relies on `yum/dnf` and simplifies managing dependencies, installing the right versions of software, and distributing it in isolated environments. This is one of the tasks more common for system administrators and should be understood completely.

For the Red Hat Certified Engineer level, a more in-depth look will be required, including creating RPM packages, which are very useful to manage, maintain, and distribute internally produced software in your own environments leveraging the experience and tools that Red Hat provides.

Now that our systems are up to date, let's move on to learn how to manage them remotely in the upcoming chapter.

Section 2: Security with SSH, SELinux, a Firewall, and System Permissions

Security in production systems is a direct responsibility of systems administrators. To handle that, RHEL includes capabilities such as SELinux, an integrated firewall, and of course, the standard system permissions. In this section, a good overview and understanding of the security mechanisms in RHEL are provided so that you can perform everyday maintenance tasks.

The following chapters are included in this section:

- *Chapter 8, Administering Systems Remotely*
- *Chapter 9, Securing Network Connectivity with firewalld*
- *Chapter 10, Keeping Your System Hardened with SELinux*
- *Chapter 11, System Security Profiles with OpenSCAP*

8
Administering Systems Remotely

When working with systems, once the server has been installed, and many times, even during the installation itself, administration can be performed remotely. Once a machine has been installed, the tasks that need to be performed during its life cycle are not that different to the ones that have already been performed.

In this chapter, we will cover, from a connection point of view, how to connect to remote systems, transfer files, and how to automate the connection so that it can be scripted and make it resilient if issues arise with the network link. Administration tasks that can be performed on the system are the same as the ones we described in previous chapters, such as installing software, configuring additional networking settings, and even managing users.

Since administering a system requires privileged credentials, we will focus on the available tools that are considered to be secure to perform such connections, as well as how to use them to encapsulate other traffic.

We will cover the following topics:

- SSH and OpenSSH overview and base configuration
- Accessing remote systems with SSH
- Key-based authentication with SSH

- Remote file management with SCP/rsync
- Advanced remote management – SSH tunnels and SSH redirections
- Remote terminals with tmux

By covering these topics, we will be able to master remote system access and bring our administration skills to the next level.

Let's start by talking about the SSH protocol and the OpenSSH client and server in the next section.

Technical requirements

You can continue using the virtual machine that we created at the beginning of this book in *Chapter 1*, *Installing RHEL8*. Any additional packages that are required will be indicated in the text. Any additional files that are required for this chapter can be downloaded from `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration`.

SSH and OpenSSH overview and base configuration

SSH is an acronym for **Secure Shell Host**. It started to spread by replacing traditional telnet usage, which was a remote login protocol that used no encryption for connecting to hosts, so the credentials that were used for logging in were transmitted in plain text. This means that anyone who had a system between the user terminal and the remote server could intercept the username and password and use that information to connect to remote systems. This is similar to what happens when credentials are transmitted to a web server via HTTP and not HTTPS.

With SSH, a secure channel is created between the client and the target host, even if the connection is performed over untrusted or insecure networks. Here, the SSH channel that's created is secure and no information is leaked.

OpenSSH provides both a server and a client (the `openssh-server` and `openssh-clients` packages in **Red Hat Enterprise Linux** (**RHEL**) that can be used to connect to and allow connections from remote hosts.

> **Tip**
>
> Knowing everything is not possible, so it is really important for **Red Hat Certified System Administrator** (**RHCSA**) certified individuals (and even later certifications, if you followed that path) to be resourceful. We already know how to install packages and how to check the manual pages that are installed by them, but we can also use those packages to find the necessary configuration files. This skill can be used to find the possible configuration files we need to edit to configure a service or a client. Remember to use `rpm -ql package` to review the list of files provided by a package if you cannot remember which one to use.

The default configuration for both the client and server allows connections, but there are many options that can be tuned.

OpenSSH server

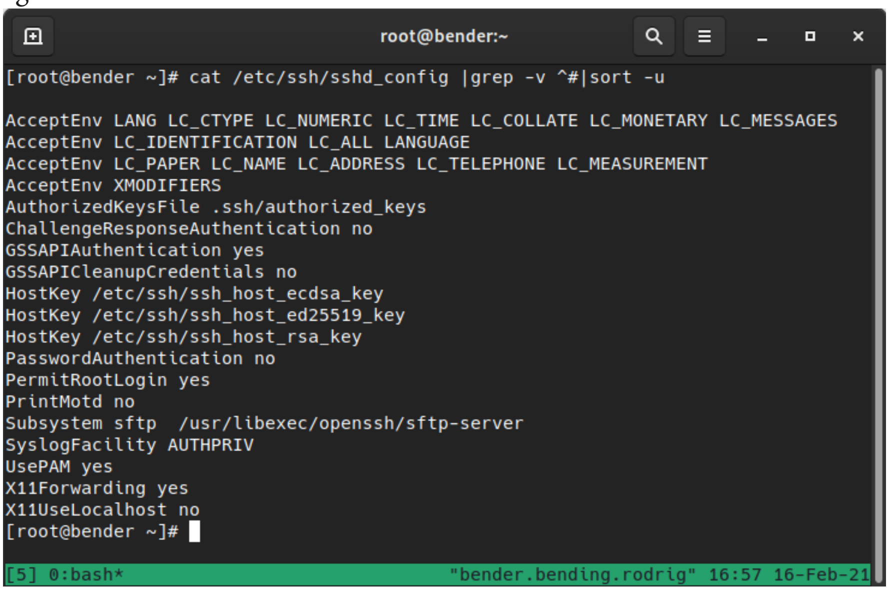
OpenSSH is a free implementation based on the last free SSH version that was created by OpenBSD members and updated with all the relevant security and features. It has become a standard in many operating systems, both as a server and as a client, to make secure connections between them.

The main configuration file for the OpenSSH server is located at `/etc/ssh/sshd_config` (and you can use `man sshd_config` to get detailed information about the different options). Some of the most used options are as follows:

- `AcceptEnv`: Defines which environment variables that have been set by the client will be used on the remote host (for example, locale, terminal type, and so on).
- `AllowGroups`: A list of groups a user should be a member of to get access to the system.
- `AllowTcpForwarding`: Allows us to forward ports using the SSH connection (we will discuss this later in this chapter, in the *SSH tunnels and SSH redirections* section).
- `DisableForwarding`: This takes precedence over other forwarding options, making it easier to restrict the service.
- `AuthenticationMethods`: Defines which authentication methods can be used, such as disabling password-based access.
- `Banner`: Files to send to the connecting user before authentication is allowed. This defaults to no banner, which might also reveal who is running the service that might be providing too much data to possible attackers.

- `Ciphers`: A list of valid ciphers to use when you're interacting with the server. You can use + or – to enable or disable them.
- `ListenAddress`: The hostname or address and port where the `sshd` daemon should be listening for incoming connections.
- `PasswordAuthentication`: This defaults to yes and can be disabled to block users from interactively connecting to the system unless a public/private keypair is used.
- `PermitEmptyPasswords`: Allows accounts with no password to access the system (the default is no).
- `PermitRootLogin`: Defines how login works for the root user, for example, to avoid the root user from connecting remotely with a password.
- `Port`: Related to `ListenAddress`, this defaults to `22`. It's the port number where the `sshd` daemon listens for incoming connections.
- `Subsystem`: Configures the command for the external subsystem. For example, it is used with `sftp` for file transfers.
- `X11Forwarding`: This defines whether `X11` forwarding is permitted so that remote users can open graphical programs on their local display by tunneling the connection.

The following screenshot shows the options that are installed by our system while we're removing comments:



```
root@bender:~
[root@bender ~]# cat /etc/ssh/sshd_config |grep -v ^#|sort -u

AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv XMODIFIERS
AuthorizedKeysFile .ssh/authorized_keys
ChallengeResponseAuthentication no
GSSAPIAuthentication yes
GSSAPICleanupCredentials no
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key
HostKey /etc/ssh/ssh_host_rsa_key
PasswordAuthentication no
PermitRootLogin yes
PrintMotd no
Subsystem sftp  /usr/libexec/openssh/sftp-server
SyslogFacility AUTHPRIV
UsePAM yes
X11Forwarding yes
X11UseLocalhost no
[root@bender ~]#

[5] 0:bash*                                  "bender.bending.rodrig" 16:57 16-Feb-21
```

Figure 8.1 – Default values at installation time defined in /etc/ssh/sshd_config

We'll check the client part of the configuration in the next section.

OpenSSH client

The client part of OpenSSH is configured system-wide via the `/etc/ssh/ssh_config` file and the files in the `/etc/ssh/ssh_config.d/` folder. They are also configured via each user `~/.ssh/config` file.

Usually, the system-wide file just contains some comments, not actual settings, so we will be focusing on the per-user configuration file and command-line parameters.

One example entry in our `~/.ssh/config` file could be as follows:

```
Host jump
    Hostname jump.example.com
    User root
    Compression yes
    StrictHostKeyChecking no
    GSSAPIAuthentication yes
    GSSAPIDelegateCredentials yes
    GSSAPIKeyExchange yes
    ProxyCommand connect-proxy -H squid.example.com:3128 %h %p
    ControlPath ~/.ssh/master-%r@%h:%p
    ControlMaster auto
```

In the previous example, we defined an entry named `jump` (that we can use with `ssh jump`) that will connect the `root` username to the `jump.example.com` host.

This is a basic setting, but we're also defining that we'll be using a helper program in `ProxyCommand` that will make use of a proxy server on `squid.example.com` on port `3128` to connect to the `%h` host and `%p` port to reach our target system. Additionally, we're making use of `Compression` and using `ControlMaster` with additional `GSSAPI` authentication.

One feature that has security implications is `StrictHostKeyChecking`. When we connect to a host for the first time, keys are exchanged between the client and the host, and the server identifies itself with the keys that are used. If they're accepted, they will be stored in the `.ssh/known_hosts` file at the user's home.

If the remote host key is changed, a warning will be printed on the `ssh` client's terminal and the connection will be refused, but when we set `StrictHostKeyChecking` to `no`, we will accept any key that's sent by the server, which might be useful if we're using a test system that gets redeployed frequently (and thus, generating a new host key). It is not recommended to be used in general, since it protects us from a server being replaced and also someone impersonating the server we want to connect to with a server that, for example, logs usernames and passwords to access our system later.

In the next section, we will learn about accessing remote systems with `ssh`.

Accessing remote systems with SSH

SSH, as we mentioned earlier in this chapter, is a protocol that's used to connect to remote systems. In general, the syntax, in its most basic form, is just executing `ssh host` within a terminal.

The `ssh` client will then initiate a connection to the `ssh` server on the target host, using the username of the currently logged-in user by default, and will try to reach the remote server on port `22/tcp`, which is the default for the SSH service.

In the following screenshot, we can see the closest server to our `localhost` system, which means we will be connecting to our own server:

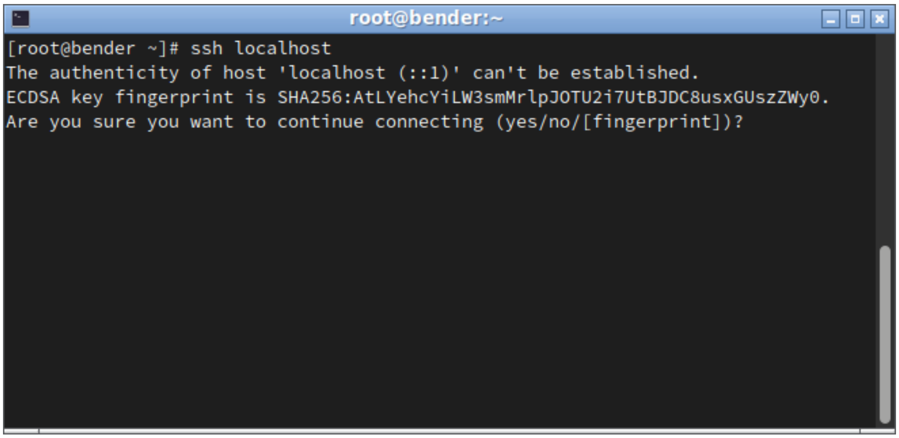


Figure 8.2 – Initiating a SSH connection to localhost

In the preceding screenshot, we can see how the first interaction with the server prints the fingerprint of the server to authenticate it. This is what was discussed in the previous section; that is, `StrictHostKeyChecking`. Once accepted, if the host key changes, the connection will be denied until we manually remove the older key to confirm that we're aware of the server change.

Let's add the key and try again, as shown in the following screenshot:

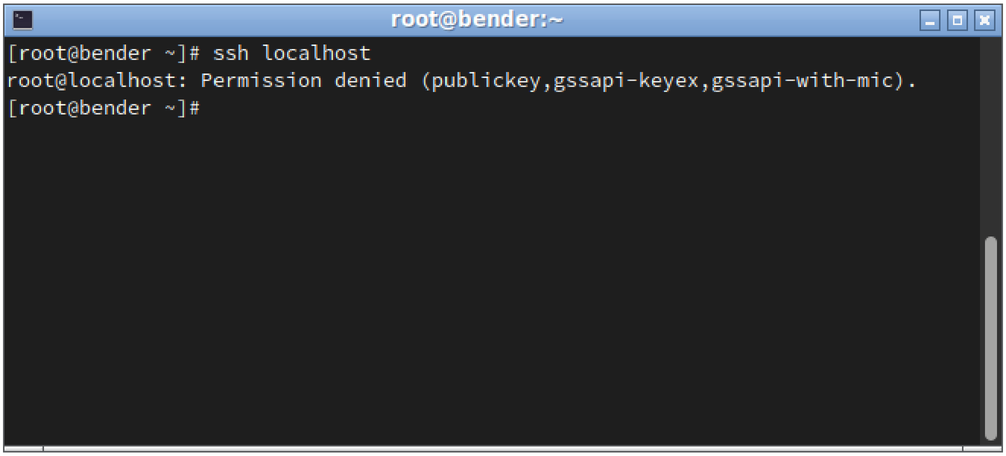


Figure 8.3 – Initiating an SSH connection to localhost denied

On our second attempt, the connection failed, but let's examine the output; that is, `Permission denied (publickey,gssapi-keyex,gssapi-with-mic)`. What does it mean? If we pay attention, `password` isn't listed, which means that we cannot connect to this host via a password prompt (which comes from setting `PasswordAuthentication` to `no`, which we defined in the `/etc/ssh/sshd_config` file).

In the following screenshot, we can see that once we set `PasswordAuthentication` to `yes`, the system asks for the password, which is not echoed on screen. Once validated, we get a shell prompt so that we can start typing in commands:

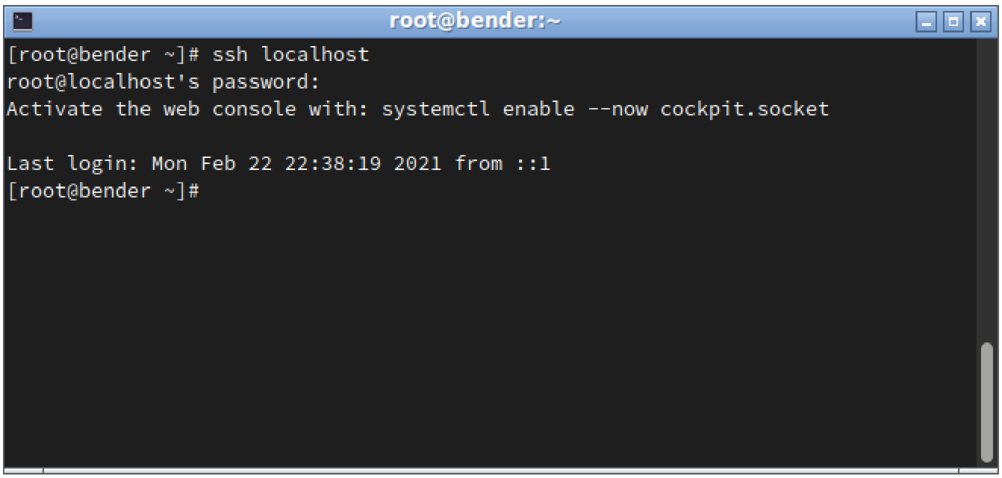


Figure 8.4 – SSH connection completed

In general, password authentication can be a security risk as the keyboard might be intercepted, someone might be looking over your shoulder, a brute-force attack might be used against the accounts, and so on. Due to this, it's common practice to at least disable it for the `root` user, meaning that someone trying to log into the system should know the username and password for a user, and from there, use the system tools to become `root`.

Let's learn how to log into remote systems that have passwords disabled by using authentication keys.

Key-based authentication with SSH

One big advantage of SSH connections is that commands can be given to be executed on remote hosts, for example, to grab updated data that can be used for monitoring without requiring a specific agent on the host.

Having to provide login details on each connection is not something that we could consider an improvement to the user experience, but SSH also allows us to create a keypair that can be used for authentication to remote systems so that no password or credential input is required.

The keys contain two parts: one that is public and must be configured in each host we want to connect to, and one that is private and must be secured as it will be used to identify us while we're trying to connect to remote hosts.

It is not necessary to say that this entire process happens over the encrypted connection created by SSH. So, using SSH and compression will also make our connections faster versus other legacy methods such as telnet, which is unencrypted.

First of all, let's create one keypair for authentication.

> **Tip**
>
> It is recommended to have at least one keypair per user so that each user can have keys based on the roles when they're connecting to servers. Even if the keys can be shared for users in a role, it's better to have each user have its own set so that keys can be revoked individually. For example, we can keep several `ssh` keypairs to be used in different roles, such as personal systems, production systems, lab systems, and so on. Having to specify the keypair for connecting is also an extra security measure: we cannot connect to production systems unless we use the production keypair.

To create a keypair, we can use the `ssh-keygen` tool, which has several options for the key we are creating, as shown in the following screenshot:

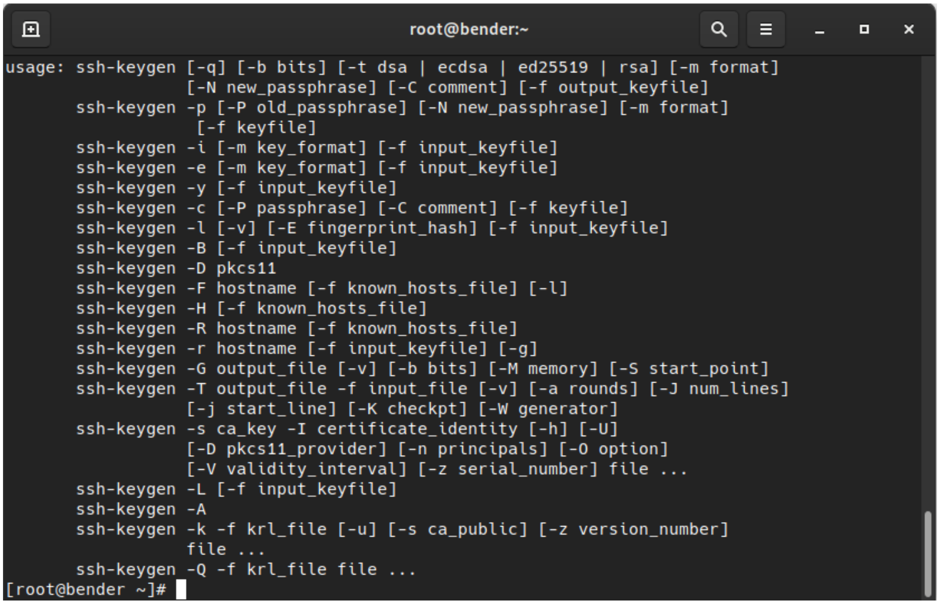


Figure 8.5 – ssh-keygen options

When no arguments are provided, by default, it will create a key for the current user and ask for a password for the key. When we use the defaults and provide no values, we get an output similar to the one shown in the following screenshot:

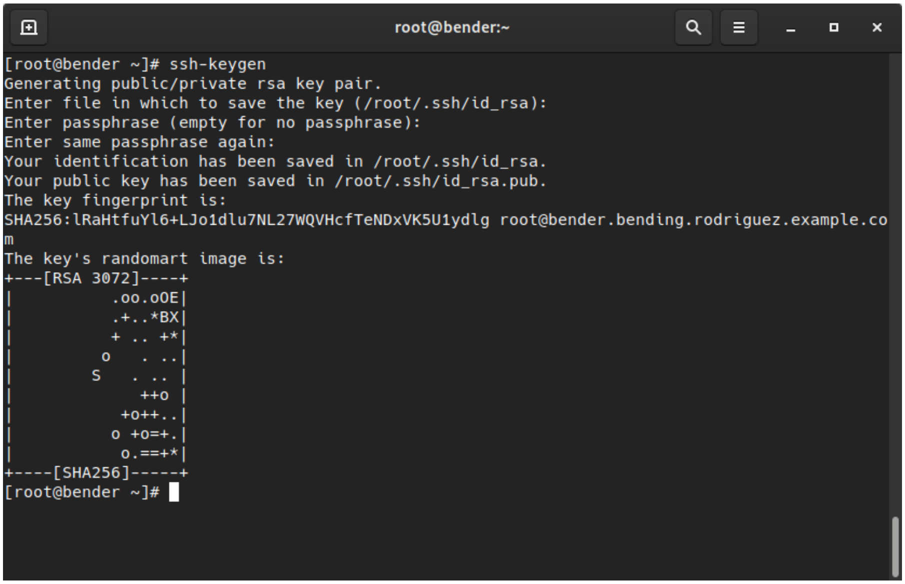


Figure 8.6 – ssh-keygen execution creating an RSA keypair under ~/.ssh/{id_rsa,id_rsa.pub}

From this point on, this system has created a keypair for the root user, and it has stored the two pieces of it in the same folder, which is `.ssh` by default. The public one contains the `.pub` suffix, while the other contains the private key.

How do we use them? If we look inside the `.ssh` folder in our home directory, we can see several files: we have an `authorized_keys` file and a `known_hosts` file, in addition to the keypair we have just created. The `authorized_keys` file will contain one entry per line. This contains the public keys that can be used to log into this system for this user.

> **Tip**
>
> The vast range of options that can be used with `authorized_keys` goes further than adding just regular keys – you can also define commands to execute, expiry times for keys, remote hosts that can be used to connect so that only those hosts will be able to use that key successfully, and many more. Again, `man sshd` is your friend, so check out the `AUTHORIZED_KEYS FILE FORMAT` section there to learn about more complex setups.

To simplify how keys are set up on remote systems, we have the `ssh-copy-id` utility, which connects via `ssh` to the remote host. This will ask for the `ssh` password and install the available public keys on our system. However, this requires the system to have password authentication enabled.

The alternate method consists of manually appending our public key to that file (`.ssh/autorized_keys`), as shown in the following screenshot:

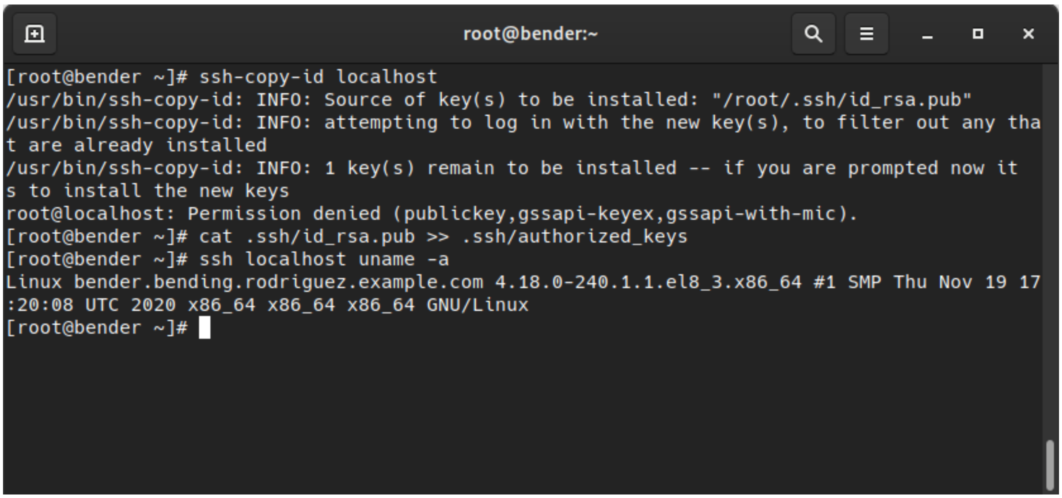


Figure 8.7 – ssh-copy-id failure and manual authorization of the private key

The first line has attempted to use `ssh-copy-id`, but since we had password authentication enabled, it tried to copy our public key and failed. Then, we appended the public key using `>>` to the `authorized_keys` file. Finally, we demonstrated how to connect to `localhost` with `ssh` and execute a command without a password.

> **Important Note**
>
> The permissions for the `.ssh` folder and the `authorized_keys` file must not be too wide open (for example, 777). If they are, the `ssh` daemon will reject them as someone could have appended new keys and tried to gain access without really being a legit user of the system.

What has just happened opens a new world of automation. Using the keys being exchanged between our system and the remote hosts, we can now connect remotely to them to run commands interactively or to script commands to be executed on remote hosts. We can check the results in our terminal. Let's consider this simple script, for a system load average check, which is available at `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/blob/main/chapter-08-remote-systems-administration/loadaverage-check.sh`:

```
#!/usr/bin/bash
for system in host1 host2 host3 host4;
do
    echo "${system}: $(ssh ${system} cat /proc/loadavg)"
done
```

In this example, we're running a loop to connect to four systems and then outputting the name and the load average of that system, as shown in the following screenshot:

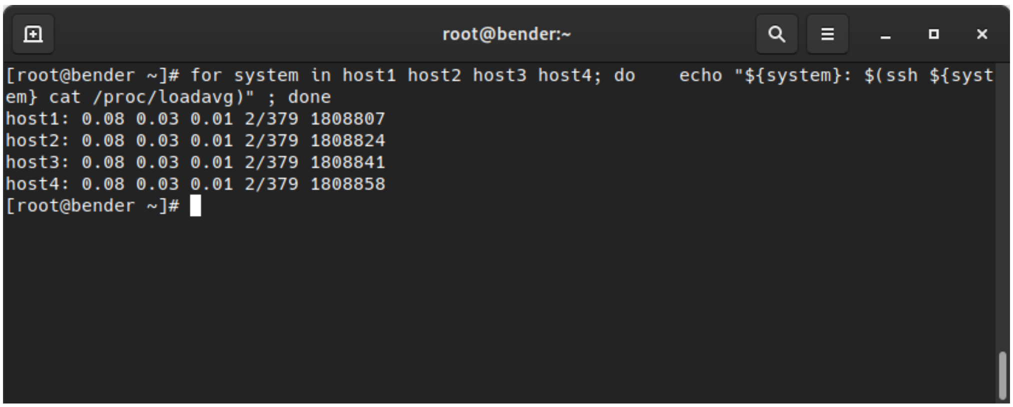


Figure 8.8 – Password-less login to four hosts to check their load average

As we can see, we quickly grabbed the information from four hosts over `ssh`. If you want to test this in your environment, you might want to put into practice what we learned about creating entries in the `/etc/hosts` file, which points to `127.0.0.1` for the host names we want to try, so that the connection goes to your own practice system, as we explained in *Chapter 6*, *Enabling Network Connectivity*.

Now, think about the different options we have for administering our systems remotely:

- Check IPs for a range of hosts.
- Install updates or add/remove one package.
- Check the local time in case the system has drifted.
- Restart one service after adding a new user to the system.

Many more options exist, but these are the main ones.

Of course, there are more suitable tools for remotely administering systems and ensuring that errors are detected and handled properly, such as using Ansible, but in this case, for simple tasks, we are good to go.

Previously, we created a key and replied with `<ENTER>` when we were asked for a password. What if we had typed one in? We'll look at this in the next section.

SSH agent

If we have decided to create an SSH key with a password to protect it (good choice), we will need to input the passphrase each time we want to use the key, so in the end, it might be as insecure as having to type in the password as someone might be checking over our shoulder. To overcome this, we can use a program called `ssh-agent` that temporarily keeps the passphrase in memory. This is convenient and reduces the chances of someone watching while you type in your key.

When you're using a graphical desktop, such as **GNOME**, as provided by RHEL, the agent might be already set up to start at session login. In the event that you're using a console (local or remote), the agent must be started manually by executing `ssh-agent`.

When `ssh-agent` is executed, it will output some variables that must be set in our environment so that we can make use of it, as shown in the following screenshot:

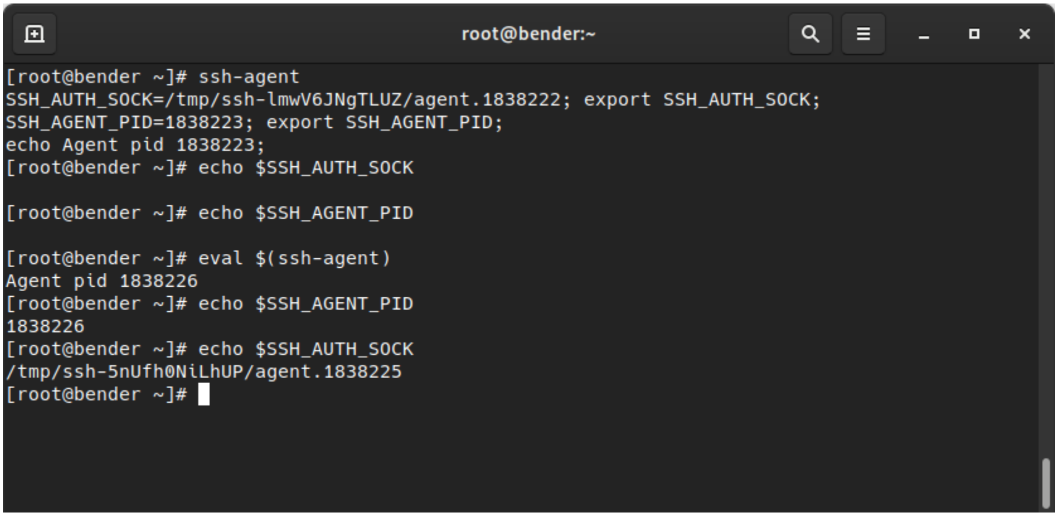


Figure 8.9 – ssh-agent being used to set the required variables

As shown in the preceding screenshot, before being executed, or just while we're executing the agent, the variables are undefined. However, if we were to execute `eval $(ssh-agent)`, we would accomplish the goal of having the variables defined and ready to use.

The next step is to add the keys to the agent. This can be accomplished with the `ssh-add` command, which can be used without parameters or by specifying the key to be added. If the key requires a password, it will prompt you for it. Once we're done, we might be able to use that key to log into the systems with the passphrase that's being cached until we exit the session that executed the agent, thus clearing the passphrase from memory.

The following screenshot shows the command that was used to generate a new keypair with a password. Here, we can see that the only difference is that we're storing it in a file named `withpass` versus what we did earlier in this chapter:

```
root@bender:~
[root@bender ~]# ssh-keygen -f withpass
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in withpass.
Your public key has been saved in withpass.pub.
The key fingerprint is:
SHA256:ONGQ/GyYRQ1JYPCUKxCBycH/BSF44FNLEXWWahOWzFc root@bender.bending.rodriguez.example.co
m
The key's randomart image is:
+---[RSA 3072]----+
|o=*OB=BO*E       |
|+++.o@*=o .      |
| ooo. BB.        |
|  ...++++        |
|    o.=.S        |
|     . .         |
|                 |
|                 |
|                 |
+----[SHA256]-----+
[root@bender ~]#
```

Figure 8.10 – Creating an additional ssh keypair with a password

We can see how to connect to our localhost (which we added the key for with the password public part of our `.ssh/authorized_keys` while removing the one without a password) and how the connection behaves in the following screenshot:

```
root@bender:~
[root@bender ~]# ssh localhost
root@localhost: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
[root@bender ~]# ssh -i withpass localhost
Enter passphrase for key 'withpass':
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Wed Feb 24 17:47:00 2021 from 37.15.250.35
[root@bender ~]# logout
Connection to localhost closed.
[root@bender ~]# ssh-add withpass
Could not open a connection to your authentication agent.
[root@bender ~]# eval $(ssh-agent)
Agent pid 1838385
[root@bender ~]# ssh-add withpass
Enter passphrase for withpass:
Identity added: withpass (root@bender.bending.rodriguez.example.com)
[root@bender ~]# ssh localhost
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Wed Feb 24 17:47:14 2021 from ::1
[root@bender ~]#
```

Figure 8.11 – Using ssh-agent to remember our passphrase

To make this clearer, let's analyze what's happening:

1. First, we ssh to the host. Permission is denied as the default key we used was removed from authorized_keys.
2. We ssh again but while defining the identity file (the keypair) to connect to, as we can see, we're asked for the passphrase for the key, not for logging into the system.
3. Then, we log out and the connection is closed.
4. Next, we try to add the key, and we get an error because we have not set the environment variables for the agent.
5. As instructed when we introduced the agent, we execute the command for loading the environment variables for the agent in the current shell.
6. When we retry adding the key with ssh-add withpass, the agent asks for our passphrase.
7. When we finally ssh to the host, we can connect without a password as the key is in memory for our keypair.

Here, we have achieved two things: we now have an automated/disattended method to connect to systems and have ensured that only authorized users will know the passphrase to unlock them.

We'll learn how to do remote file management in the next section!

SCP/rsync – remote file management

Similar to telnet, which was replaced with ssh on many devices and systems, using insecure solutions for file transfer is being reduced. By default, the **File Transfer Protocol** (**FTP**) uses TCP port 21, but since communication happened in plain text, it was a perfect target for intercepting credentials. FTP is still used today, mostly for serving file on servers that only allow anonymous access and wish to move to more secure options.

SSH usually enables two interfaces for copying files: scp and sftp. The first one is used in a similar way to the regular cp command, but here, we're accepting remote hosts as our target or source, while sftp uses a client approach similar to the traditional ftp command that interacts with FTP servers. Just remember that in both cases, the connection is encrypted and happens over port 22/tcp on the target host.

We'll dig into SCP in the next section.

Transferring files with an OpenSSH secure file copy

The `scp` command, which is part of the `openssh-clients` package, allows us to copy files between systems using the `ssh` layer for the whole process. This allows us to securely transfer a file's contents, plus all the automation capabilities that were introduced by keypair login, to various systems.

To set up this example, we will create a new user in our sample system that will be used to copy over files using the tools described in this section, as shown in the following screenshot:

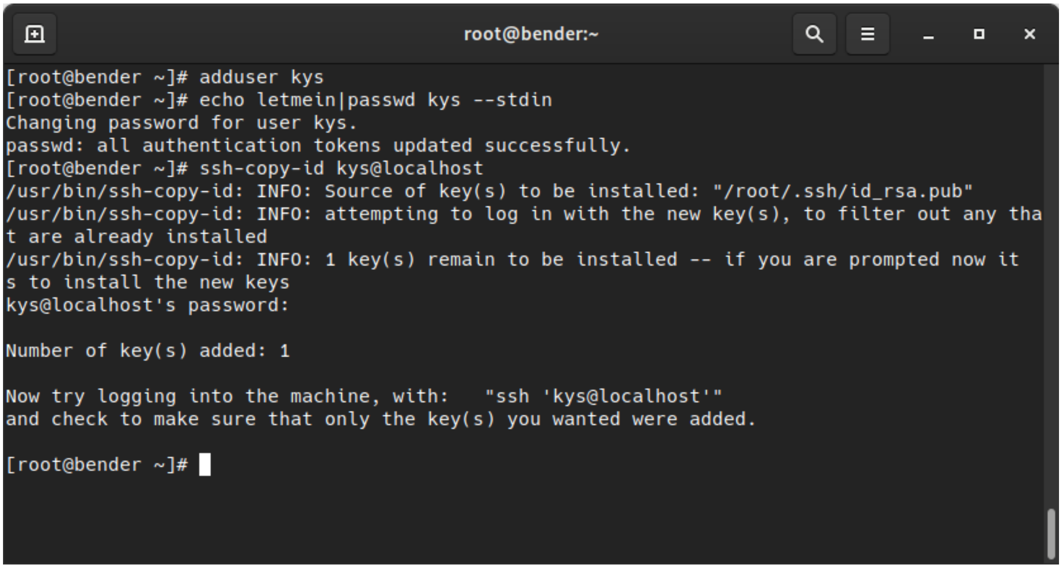


Figure 8.12 – Preparing our system with an additional user to practice file transfers

You can find the preceding commands in a script available at `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/blob/main/chapter-08-remote-systems-administration/create-kys-user.sh`.

Once the user has been created and the key has been copied, we can start testing!

Earlier in this chapter, we created a key named `withpass` with a public counterpart at `withpass.pub`. To provide the key to the newly created user, we can copy both files to the `kys` user via the following command:

```
scp withpass* kys@localhost:
```

Let's analyze each part of the command using this template:

```
scp origin target
```

In our case, `origin` is indicated with `withpass.*`, which means that it will select all the files starting with the `withpass` string.

Our `target` value is a remote host. Here, the username is `kys`, the host is `localhost`, and the folder you should store the files in is the default one, usually the home folder of the user indicated (the one with an empty path after the `:` symbol).

In the following screenshot, we can see the output of the command and the validation we can perform later via remote execution:

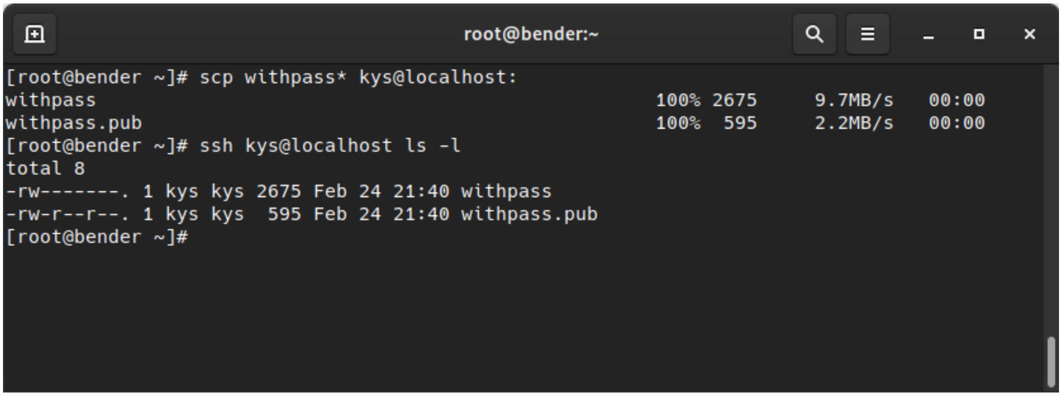


Figure 8.13 – Copying SCP files to a remote path and validating the files that have been copied

In the preceding screenshot, you can also check that the files that were owned by the root user are copied. The copied ones are owned by the `kys` user, so the file's contents are the same, but since the creator on the target is the `kys` user, files have their ownership.

We can also make more complex copies by indicating remote files first and local paths as targets so that we download files to our system, or even copy files across remote locations for both the origin and target (unless we specify the `-3` option, they will go directly from `origin` to `target`).

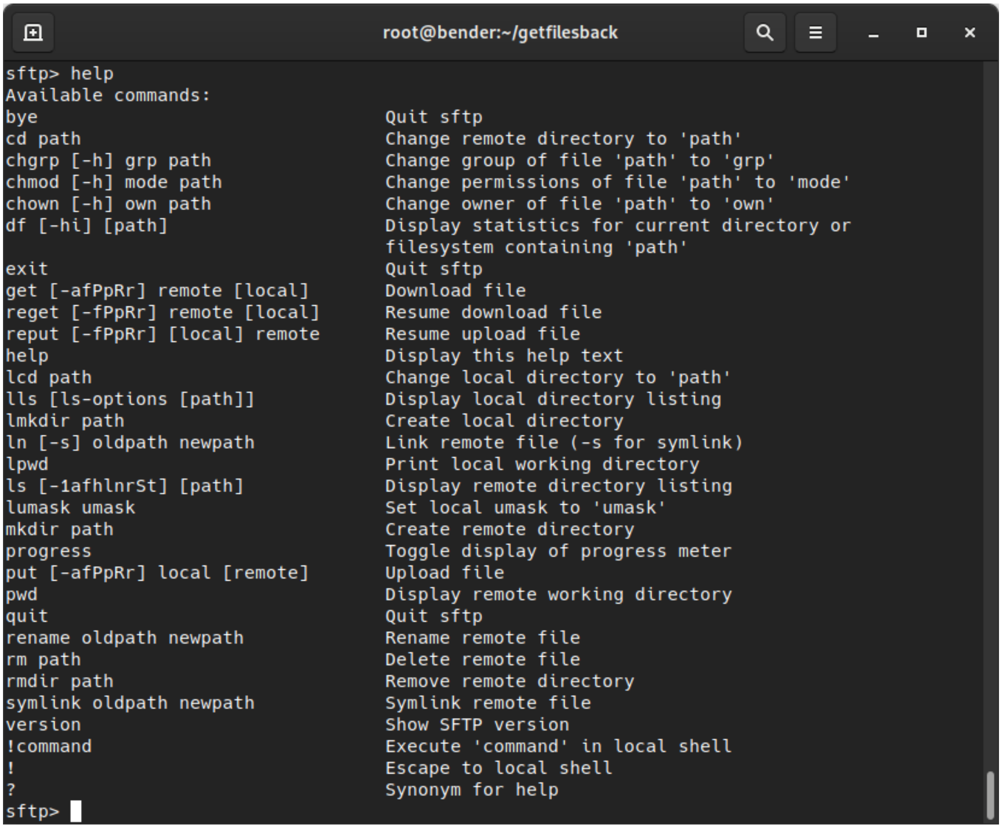
> **Tip**
>
> Time for a reminder! `man scp` will show you all the available options for the `scp` command, but since it is based on `ssh`, most of the options we use with `ssh` are available, as well as the host definitions we made in the `.ssh/config` file.

We'll explore the `sftp` client in the next section.

Transferring files with sftp

Compared to `scp`, which can be scripted in the same way we can script with the regular `cp` command, `sftp` has an interactive client for navigating a remote system. However, it can also automatically retrieve files when a path containing files is specified.

To learn about the different commands that are available, you can invoke the `help` command, which will list the available options, as shown in the following screenshot:



```
root@bender:~/getfilesback
sftp> help
Available commands:
bye                                Quit sftp
cd path                            Change remote directory to 'path'
chgrp [-h] grp path                Change group of file 'path' to 'grp'
chmod [-h] mode path               Change permissions of file 'path' to 'mode'
chown [-h] own path                Change owner of file 'path' to 'own'
df [-hi] [path]                    Display statistics for current directory or
                                   filesystem containing 'path'
exit                               Quit sftp
get [-afPpRr] remote [local]       Download file
reget [-fPpRr] remote [local]      Resume download file
reput [-fPpRr] [local] remote      Resume upload file
help                               Display this help text
lcd path                           Change local directory to 'path'
lls [ls-options [path]]            Display local directory listing
lmkdir path                        Create local directory
ln [-s] oldpath newpath            Link remote file (-s for symlink)
lpwd                               Print local working directory
ls [-1afhlnrSt] [path]             Display remote directory listing
lumask umask                       Set local umask to 'umask'
mkdir path                         Create remote directory
progress                           Toggle display of progress meter
put [-afPpRr] local [remote]       Upload file
pwd                                Display remote working directory
quit                               Quit sftp
rename oldpath newpath             Rename remote file
rm path                            Delete remote file
rmdir path                         Remove remote directory
symlink oldpath newpath            Symlink remote file
version                            Show SFTP version
!command                           Execute 'command' in local shell
!                                  Escape to local shell
?                                  Synonym for help
sftp>
```

Figure 8.14 – Available sftp interactive mode commands

Let's look at an example of this with the help of the following screenshot:

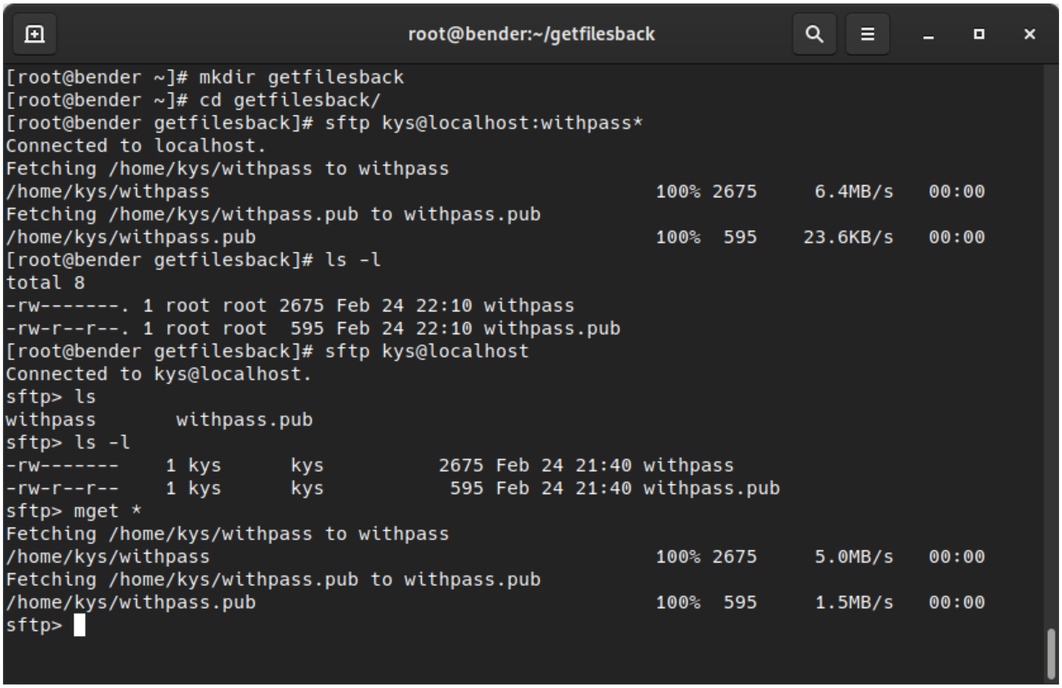


Figure 8.15 – Both modes of operation with sftp – automated transfer or interactive transfer

In this example, we've created a local folder to be our work folder, called `getfilesback`. First, we have invoked `sftp` with a remote path with the files we've identified. Here, `sftp` has automatically transferred the files and has stopped executing. The files we have received are now the property of our user.

In the second command, when we invoke `sftp` with the user and host and we enter interactive mode, we can execute several commands, similar to what we can do on a remote shell session. Finally, using the `mget` command with the `*` wildcard character, we transferred the files to our local system.

In both cases, the files have been transferred from the remote system to our local system, so our goal has been accomplished. However, using `scp` requires knowing the exact path of the files you want to transfer. On the other hand, it might be a bit more user-friendly to navigate the system using the `ls` and `cd` commands within the `sftp` interactive client until we reach the files we want to transfer if we can't remember it.

Now, let's learn how to quickly transfers files and trees with `rsync`.

Transferring files with rsync

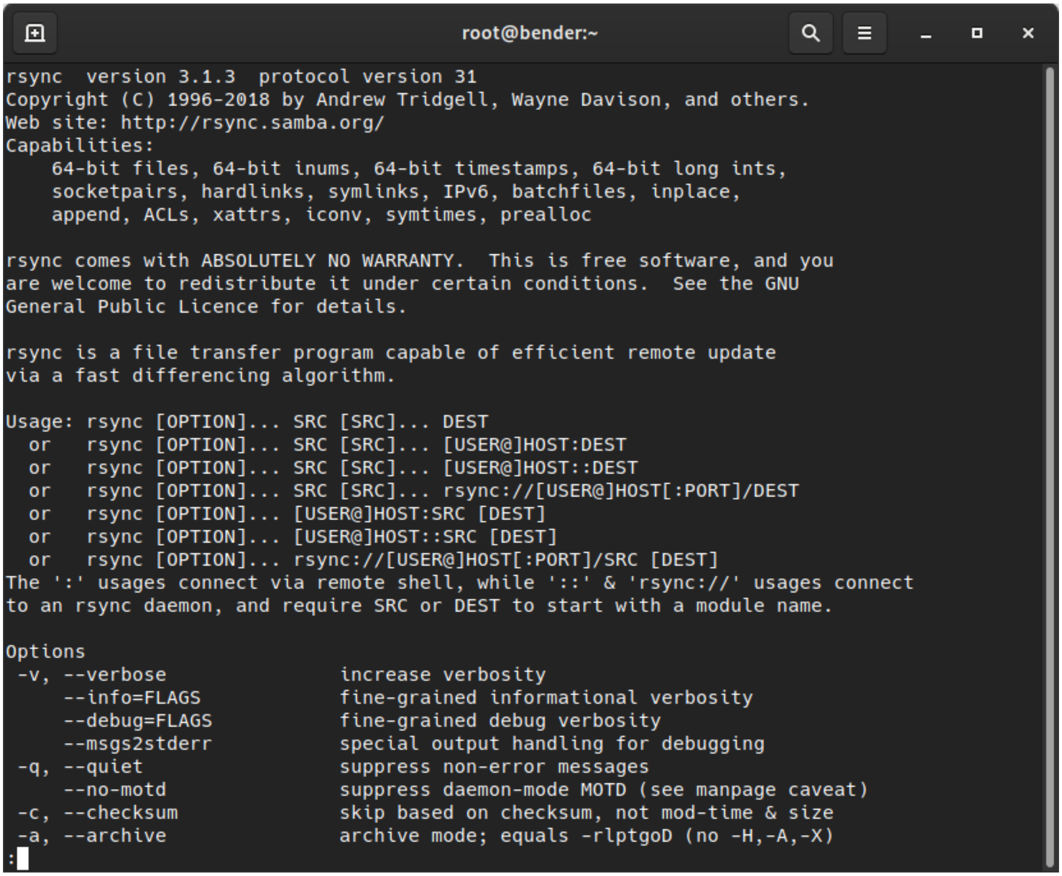
Although we can use the `-r` option of `scp` to transfer files recursively, `scp` only handles the full copy of the file, which is not ideal if we are just keeping some folders in sync across systems.

In 1996, `rsync` was launched, and many systems implemented it by using a dedicated server that was listening to client connections. This was to allow trees to be synchronized with files. This was done by copying over the differences between the files. Here, parts of the source and destination were compared to see whether there were differences that should be copied over.

With `ssh`, and with the `rsync` package installed on both the client and the server, we can take advantage of the secure channel that's created by `ssh` and the faster synchronization provided by `rsync`.

The difference between using the `rsync` daemon and using `ssh` is the syntax for the source or destination, which either uses the `rsync://` protocol or `::` after the hostname. In other cases, it will use `ssh` or even the local filesystem.

The following screenshot shows us mentioning the schema for URLs via the `rsync –help` command:



```
root@bender:~
rsync  version 3.1.3  protocol version 31
Copyright (C) 1996-2018 by Andrew Tridgell, Wayne Davison, and others.
Web site: http://rsync.samba.org/
Capabilities:
    64-bit files, 64-bit inums, 64-bit timestamps, 64-bit long ints,
    socketpairs, hardlinks, symlinks, IPv6, batchfiles, inplace,
    append, ACLs, xattrs, iconv, symtimes, prealloc

rsync comes with ABSOLUTELY NO WARRANTY.  This is free software, and you
are welcome to redistribute it under certain conditions.  See the GNU
General Public Licence for details.

rsync is a file transfer program capable of efficient remote update
via a fast differencing algorithm.

Usage: rsync [OPTION]... SRC [SRC]... DEST
  or   rsync [OPTION]... SRC [SRC]... [USER@]HOST:DEST
  or   rsync [OPTION]... SRC [SRC]... [USER@]HOST::DEST
  or   rsync [OPTION]... SRC [SRC]... rsync://[USER@]HOST[:PORT]/DEST
  or   rsync [OPTION]... [USER@]HOST:SRC [DEST]
  or   rsync [OPTION]... [USER@]HOST::SRC [DEST]
  or   rsync [OPTION]... rsync://[USER@]HOST[:PORT]/SRC [DEST]
The ':' usages connect via remote shell, while '::' & 'rsync://' usages connect
to an rsync daemon, and require SRC or DEST to start with a module name.

Options
 -v, --verbose               increase verbosity
     --info=FLAGS            fine-grained informational verbosity
     --debug=FLAGS           fine-grained debug verbosity
     --msgs2stderr           special output handling for debugging
 -q, --quiet                 suppress non-error messages
     --no-motd               suppress daemon-mode MOTD (see manpage caveat)
 -c, --checksum              skip based on checksum, not mod-time & size
 -a, --archive               archive mode; equals -rlptgoD (no -H,-A,-X)
:
```

Figure 8.16 – The rsync command's help output

Now, let's review some of the useful options we can use with `rsync`:

- `-v` : Provides more verbose output during the transfer.
- `-r`: Recurses into directories.
- `-u`: Update – only copies files that are newer than the ones at the target.
- `-a`: Archive (this includes several options, such as `-rlptgoD`).
- `-X`: Preserves extended attributes.
- `-A`: Preserves ACLs.
- `-S`: Sparse – sequences of nulls will be converted into sparse blocks.

- `--preallocate`: Claims the space that's required for files before transferring them.
- `--delete-during`: Deletes files on the target that are not hosted during the copy.
- `--delete-before`: Deletes files on the target that are not hosted before the copy.
- `--progress`: Shows progress information on the copy (copied files versus total files).

The `rsync` algorithm breaks the file into chunks and calculates checksums for each chunk that's transmitted to source. They are then compared to the ones for local files. We are only allowed to share the differences between the source and the target. `rsync` doesn't check the modification file date and size by default, so if the file has changed without changes being left in both, the change might not be detected unless a checksum check is forced for each file candidate to be transferred.

Let's look at some basic examples:

- `rsync -avr  getfilesback/ newfolder/` will copy the files in the local `getfilesback/` folder to `newfolder/` by showing a progress update, but only for the updated files, as shown in the following screenshot:

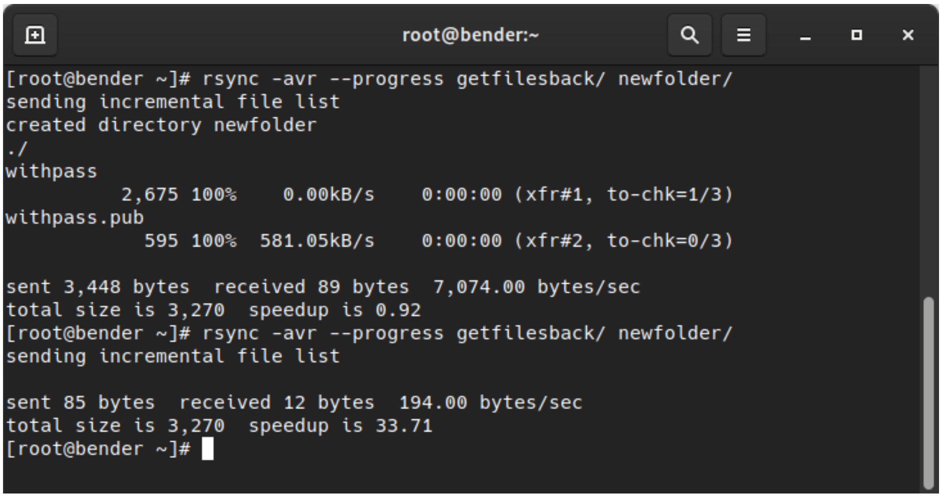


Figure 8.17 – The rsync operation being used on the same source/destination, repeated to illustrate transfer optimization

As we can see, the second operation just sent 85 bytes and received 12 bytes. This is because there was a little checksum operation happening internally to validate across the folders because the files hadn't been changed. The same output can be obtained if we use the remote target approach with `rsync -avr --progress getfilesback/ root@localhost:newfolder/`, but in this case, `ssh` transport will be used.

Let's get some bigger sample files and compare them by checking out a Git repository at some point in time, transferring the files, then updating to the latest version to simulate work on the repository. Then, we will synchronize again.

First, let's install `git` if it's not installed and check out a sample repository by executing the following code:

```
dnf -y install git   # install git in our system
git clone https://github.com/citellusorg/citellus.git  # clone a repository over https
cd citellus # to enter into the repository folder
git reset HEAD~400  # to get back 400 commits in history
```

At this point, we have a folder with files ready for transfer. Once we've done this, we'll execute `git pull` to sync with the latest changes and use `rsync` again to copy the differences. Later, we'll use `--delete` to remove any files that no longer exist on the source.

Let's check out the sequence shown in the following screenshot:

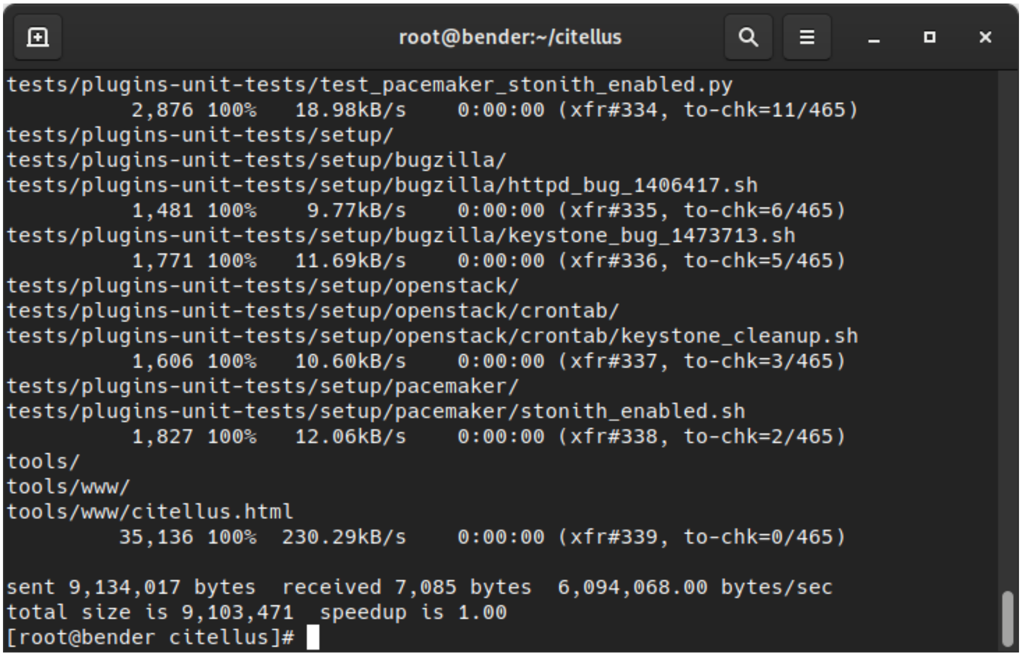


Figure 8.18 – Synchronizing the git folder to a new folder with rsync

In the preceding screenshot, pay attention to the speedup that's reported in the latest line of the command.

Now, let's execute `git pull` to get the 400 changes we were missing and repeat `rsync` again. We will get an output similar to the following:

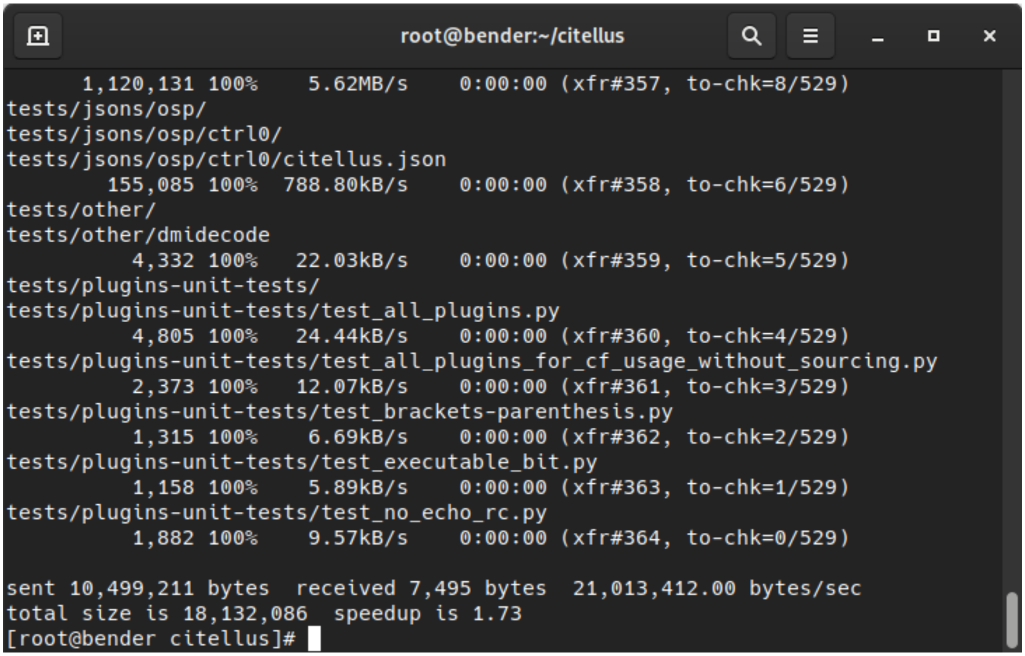


Figure 8.19 – Using rsync again to copy over the differences

In the preceding screenshot, pay attention to the speedup reported in the last line so that you can compare it with the previous one.

From this sequence of screenshots, we can check the last numbers for the total bytes that were sent to see the improvement in transfer, along with some of the files that were received (because we added the `–v` modifier to get verbose output and `--progress`).

The biggest advantage comes when a copy is performed over slower network links and it's performed periodically, for example, as a way to copy to an offsite copy for backup purposes. This is because `rsync` will only copy the changes, update the newer files that have been modified on the source, and allow us to use compression over the `ssh` channel. For example, the Linux kernel at `https://www.kernel.org/` can be mirrored using `rsync`.

In the next section, we will dig into a very interesting feature of SSH to make connecting to servers with no direct access easy.

Advanced remote management – SSH tunnels and SSH redirections

SSH has two really powerful features; that is, SSH tunnels and SSH redirections. When an SSH connection is established, it can not only be used to send commands to the remote host and let us work on them as if they were our local system, but we can also create tunnels that interconnect our systems.

Let's try to imagine a scenario that is common in many companies, where a VPN is used to reach the internal network with all the services and servers, but with SSH instead of a regular VPN.

So, let's put some context into this imaginary scenario.

We can use a host that gets external traffic for `ssh` redirected from our internet router to the `ssh` service in that system. So, in brief, our router gets connections on port `22` via TCP and the connection is forwarded to our server. We will be naming this server bastion in this exercise.

With this in place, our common sense tells us that we will be able to reach that bastion host via SSH, even if we can use other tools or even `ssh` it to connect to other systems later.

Can we connect directly to other hosts in the internal network? The answer is YES, because, by default, SSH allows us to use TCP forwarding (`sshd_config` setting `AllowTcpForwarding`), which empowers us, as remote login users, to create port redirections and even a **SOCKS** proxy to be used for our connections.

For example, we can create a tunnel using that bastion host to reach our internal mail server via the **Internet Message Access Protocol** (**IMAP**) and **Simple Mail Transfer Protocol** (**SMTP**) protocols by just executing the following code:

```
ssh -L 10993:imap.example.com:993 -L 10025:smtp.example.com:25
user@bastionhost
```

This command will listen on local ports `10993` and `10025`. All the connections that are performed there will be tunneled until `bastionhost` connects those to `imap.example.com` at port `993` and `smtp.example.com` at port `25`. This allows our local system to configure our email account using those custom ports and use `localhost` as the server, and still be able to reach those services.

> **Tip**
>
> Ports under `1024` are considered privileged ports and usually, only the root user can bind services to those ports. That's why we use them for our redirection ports, `10025` and `10093`, so that those can be used by a regular user instead of requiring the root user to perform the `ssh` connection. Pay attention to `ssh` messages when you're trying to bind to local ports in case those are in use, as the connections might fail.
>
> Additionally, from the target server's point of view, the connections will appear as if they originated in the bastion server as it's the one effectively performing the connections.

When the list of open ports starts to grow, it is better to go back to what we explained at the beginning of this chapter: the `~/.ssh/config` file can hold the host definition, along with the redirections we want to create, as shown in this example:

```
Host bastion
    ProxyCommand none
    Compression yes
    User myuser
    HostName mybastion.example.com
    Port 330
    LocalForward 2224 mail.example.com:993
    LocalForward 2025 smtp.example.com:25
    LocalForward 2227 ldap.example.com:389
    DynamicForward 9999
```

In this example, when we are connecting to our bastion host (via `ssh bastion`), we are automatically enabling **Compression**, setting the host to connect to `mybastion.example.com` at port `330`, and defining port forwarding for our `imap`, `smtp`, and `ldap` servers and one dynamic forward (SOCKS proxy) at port `9999`. If we have different identities (keypairs), we can also define the one we wish to use via the `IdentityFile` configuration directive for each host, or even use wildcards such as `Host *.example.com` to automatically apply those options to hosts ending in that domain that have no specific configuration stanza.

> **Note**
>
> Sometimes, while using `ssh`, `scp`, or `sftp`, the goal is to reach a system that is accessible from a bastion host. Other port forwarding is not needed here – only reaching those systems is required. In this case, you can use the handy `-J` command-line option (equivalent to defining a `ProxyJump` directive) to use that host as a jump host to the final target you want to reach. For example, `ssh -J bastion mywebsiteserver.example.com` will transparently connect to `bastion` and jump from there to `mywebsiteserver.example.com`.

In the next section, we will learn how to protect ourselves from network issues with our remote connections and get the most out of our remote terminal connection.

Remote terminals with tmux

`tmux` is a terminal multiplexer, which means that it allows us to open and access several terminals within a single screen. A good similitude would be a window manager in a graphical desktop, which allows us to open several windows so that we can switch context while using only one monitor.

`tmux` also allows us to detach and reattach to the sessions, so it's the perfect tool in case our connection drops. Think, for example, about performing a software upgrade on a server. If, for whatever reason, the connection drops, it will be equivalent to abruptly stopping the upgrade process in whatever status it was at that moment, which can lead to bad consequences. However, if the upgrade was launched inside `tmux`, the command will continue executing, and once the connection is restored, the session can be reattached and the output will be available to be examined.

First of all, let's install it on our system via `dnf -y install tmux`. This line will download the package and make the `tmux` command available. Bear in mind that the goal of `tmux` is not to install it on our system (even if this is useful), but for it to be available on the servers we connect to, to get that extra layer of protection in case a disconnection happens. So, it's a good habit to get used to installing it on all the servers we connect to.

> **Tip**
>
> In versions prior to `RHEL8`, the tool that was used for creating virtual multiplexed terminals was `screen`, which has been marked as deprecated and is only available via `EPEL` repositories. If you were used to its key bindings (`CTRL-A + <key>`), most of them are equivalent in `tmux` via (`CTRL-B + <key>`).

In the following screenshot, we can see what `tmux` looks like with the default configuration after executing `tmux` on a command line:

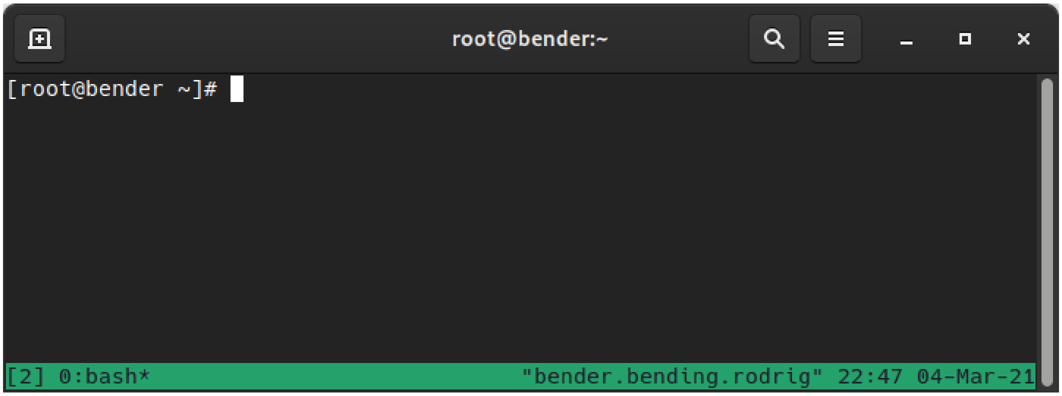


Figure 8.20 – tmux default layout after execution

As shown in the preceding screenshot, it hasn't changed a lot of the view of our terminal except for the status bar in the lower part of the window. This shows some information about the host, such as its name, time, date, and the list of open windows, with `0:bash` being the active one, as denoted by the asterisk (`*`) symbol.

There are lots of combinations for using `tmux`, so let's get familiar with some of them that will cover the initial use case:

- Run `tmux` to create a new session.
- Run `tmux at` to attach to a previous session (for example, after reconnecting to a host).
- Run `tmux at -d` to attach to a previous session and detach other connections from it.

Once we're inside `tmux`, there is a whole world of commands we can use that are preceded by the `CTRL+B` keys. Let's view some important ones (remember that *Ctrl + B* must be pressed before you use the next item in the list):

- `?`: Displays inline help about the shortcuts to use.
- `c`: Creates a new window.
- `n/p`: Go to the next/previous window.
- `d`: Detaches the `tmux` session.
- `0-9`: Go to the window numbered with the pressed number.

- `,`: Renames windows.
- `"`: Splits the pane horizontally.
- `%`: Splits the pane vertically.
- `space`: Switches to the next layout.
- `&`: Kills the window.
- `Pg down/pg up`: Go higher or lower in the window history.
- Arrow keys: Select the pane in the direction of the pressed key.

Let's look at an example in action in the following screenshot:

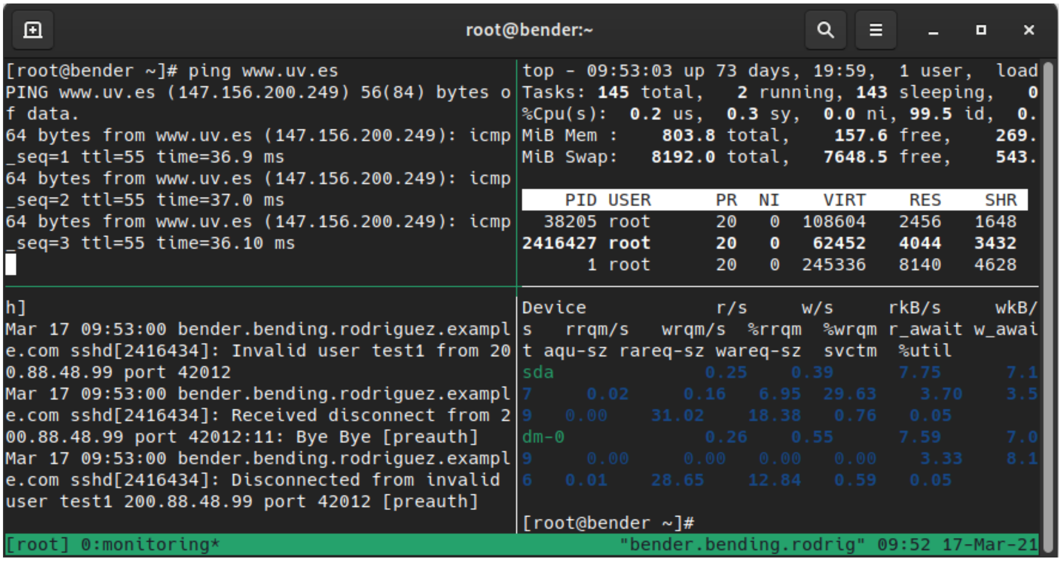


Figure 8.21 – tmux with four panes running different commands inside the same window

As we can see, there are several commands running at the same time – `top`, `journalctl -f`, `iostat -x`, and `ping` – so this is a good way to monitor a system while operations are being performed on it.

Additionally, one of the advantages is that `tmux` can be scripted, so if we are using one layout while administering systems, we can copy that script and execute it as soon as we connect to them so that we can enjoy the same layout and even the commands being executed.

You can find the following code with extra comments and descriptions at `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/blob/main/chapter-08-remote-systems-administration/term.sh` if you want to try it on your system:

```
#!/bin/bash
SESSION=$USER
tmux -2 new-session -d -s $SESSION # create new session
tmux select-window -t $SESSION:0  # select first window
tmux rename-window -t $SESSION "monitoring" #rename to
monitoring
tmux split-window –h #split horizontally
tmux split-window –v #split vertically
tmux split-window –h # split again horizontally
tmux select-layout tiled #tile panes
tmux selectp –t1 # select pane 1
tmux send-keys "top" C-m #run top by sending the letters +
RETURN
tmux selectp –t2 # select pane 2
tmux send-keys "journalctl -f" C-m # run journalctl
tmux selectp –t3 # select pane 3
tmux send-keys "iostat -x" C-m # run iostat
tmux selectp –t0 #select the pane without commands executed
```

Once the session with `tmux` has been set, we can attach the session we've just created and configured by executing `tmux`, which will show a layout similar to the one shown in the preceding screenshot.

Summary

In this chapter, we covered SSH and how to use it to connect to remote systems, how to use keys to authenticate with or without a password, and how to take advantage of it for automation, transferring files, and even making services accessible or reachable via port redirection. With `tmux`, we learned how to make our administration sessions survive network interruptions and at the same time, show important information at a glance by automating the layouts for it.

In the next chapter, we'll be digging into securing our system network via firewalld to only expose the services that are required for operation.

9
Securing Network Connectivity with firewalld

A great mentor and technologist working with military restricted environments once told me that *"The only secure system is the one that is switched off, disconnected from any network, and buried in the middle of the desert."* He is right, of course, but we must provide a service to make the system useful. This means having it running and connected to a network.

One of the techniques that's used in security to reduce incidents, such as avoiding unexpected exposure to a vulnerability and enabling unauthorized remote access, is reducing the attack surface and applying defense in depth principles. When you do that in a network, step one is filtering connections using a **firewall**. The firewall management tool that's included in **Red Hat Enterprise Linux** (**RHEL**) is **firewalld** and it helps us manage zones, profiles, services, and ports. It also includes a command-line tool called `firewall-cmd` and a `systemd` service unit to simplify its management.

In this chapter, we will cover the following topics to get a good understanding of how to manage the default firewall in RHEL:

- Introduction to the RHEL firewall – firewalld
- Enabling firewalld on the system and reviewing the default zones

- Reviewing the different configuration items under firewalld
- Enabling and managing services and ports
- Creating and using service definitions for firewalld
- Configuring firewalld with the web interface

Introduction to the RHEL firewall – firewalld

RHEL comes with two low-level network traffic filtering mechanisms: **nftables**, for filtering IP-related traffic, and **ebtables**, for transparent filtering in bridges. These mechanisms are static and use a set of rules to accept or reject traffic, though they do provide a myriad of other capabilities. In RHEL, they are both handled and managed dynamically by **firewalld**. Unless there is a specific need to have a very low-level usage of these low-level filtering mechanisms, please use firewalld (or its main command; that is, `firewall-cmd`) instead. In this section, we will take a look at the firewall defaults in RHEL.

firewalld is installed by default in the system, which we can check by using the `rpm` command, so there is no need to install it:

```
[root@rhel8 ~]# rpm -qa | grep firewalld
firewalld-filesystem-0.8.2-2.el8.noarch
firewalld-0.8.2-2.el8.noarch
```

If we have an installation that doesn't include firewalld for some reason, we can install it by running `dnf install firewalld`.

firewalld includes a service called `firewalld`, which is configured to run by default at startup. We can check this by using the `systemctl status firewalld` command:

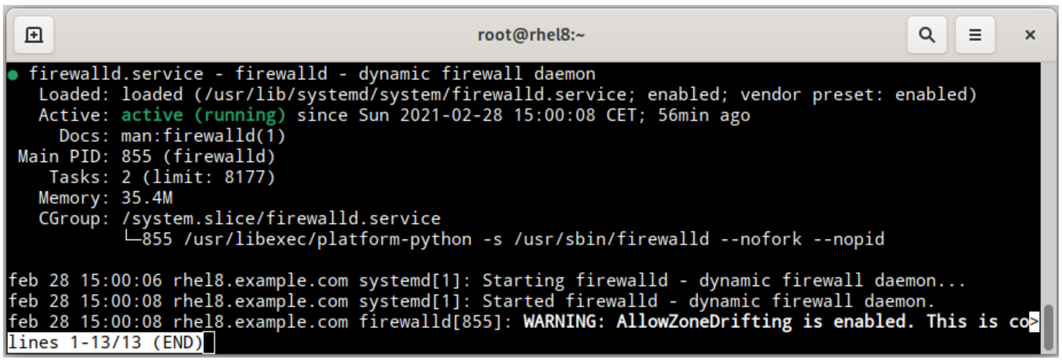


Figure 9.1 – Output of "systemctl status firewalld"

As we can see, the `firewalld` service is enabled and running. This is the default status in a RHEL system.

The main way for a sysadmin to configure firewalld is by using the `firewall-cmd` command. However, you can also do the following:

- Add new files with service definitions in `/etc/firewalld/` (as explained in the *Creating and using service definitions for firewalld* section of this chapter)
- Use the web interface, called **cockpit**, to configure firewalls (as explained in the *Configuring firewalld with the web interface* section of this chapter)
- Use the `firewall-config` graphical interface in your desktop environment

In this chapter, we will review the main mechanism and the web interface.

Now that we know the defaults of the RHEL main firewall, let's learn how to enable it.

Enabling firewalld in the system and reviewing the default zones

We have already seen that **firewalld** is enabled by default in the system. However, we may need to disable (that is, check if the firewall is interfering with a service), re-enable (that is, after restoring configuration files), and start and stop it (that is, to reload configuration or to do a quick check). This tasks are managed like any other service in the system; that is, using `systemctl`. Let's stop the `firewalld` service:

```
[root@rhel8 ~]# systemctl stop firewalld
[root@rhel8 ~]# systemctl status firewalld
  firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Sun 2021-02-28 17:36:45 CET; 4s ago
     Docs: man:firewalld(1)
  Process: 860 ExecStart=/usr/sbin/firewalld --nofork --nopid $FIREWALLD_ARGS (code=exited, status=>
Main PID: 860 (code=exited, status=0/SUCCESS)

feb 28 17:36:19 rhel8.example.com systemd[1]: Starting firewalld - dynamic firewall daemon...
```

```
feb 28 17:36:20 rhel8.example.com systemd[1]: Started firewalld - dynamic firewall daemon.
feb 28 17:36:20 rhel8.example.com firewalld[860]: WARNING: AllowZoneDrifting is enabled. This is co>
feb 28 17:36:45 rhel8.example.com systemd[1]: Stopping firewalld - dynamic firewall daemon...
feb 28 17:36:45 rhel8.example.com systemd[1]: firewalld.service: Succeeded.
feb 28 17:36:45 rhel8.example.com systemd[1]: Stopped firewalld - dynamic firewall daemon.
```

In the previous output, as shown in bold the service is inactive. We can check this by using the `firewall-cmd --state` command:

```
[root@rhel8 ~]# firewall-cmd --state
not running
```

At the moment, the firewall service has stopped and all the rules have been dropped. The configuration for the service, however, has not changed, so if we reboot the system, firewalld will be running again.

> **Tip**
> We can always see the underlying `netfilter` rules by running the `nft list table filter` command. You may want to run it before and after stopping the service to see the difference.

Now, let's try and start the service again:

```
[root@rhel8 ~]# systemctl start firewalld
[root@rhel8 ~]# systemctl status firewalld
  firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-02-28 17:43:31 CET; 7s ago
     Docs: man:firewalld(1)
Main PID: 1518 (firewalld)
    Tasks: 2 (limit: 8177)
   Memory: 23.3M
   CGroup: /system.slice/firewalld.service
```

```
           └─1518 /usr/libexec/platform-python -s /usr/sbin/
firewalld --nofork -nopid
```

Let's check that firewalld is running:

```
[root@rhel8 ~]# firewall-cmd --state
running
```

To fully disable the service, we will need to run the following command:

```
[root@rhel8 ~]# systemctl disable firewalld
Removed /etc/systemd/system/multi-user.target.wants/firewalld.
service.
Removed /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.
service.
```

Let's see how the service is disabled but still running:

```
[root@rhel8 ~]# systemctl status firewalld -n0
  firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service;
disabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-02-28 17:43:31 CET;
8min ago
     Docs: man:firewalld(1)
Main PID: 1518 (firewalld)
    Tasks: 2 (limit: 8177)
   Memory: 24.1M
   CGroup: /system.slice/firewalld.service
           └─1518 /usr/libexec/platform-python -s /usr/sbin/
firewalld --nofork -nopid
```

When you're managing services with **systemd** using `systemctl`, you need to understand that enabling and disabling the service only affects how it behaves during the startup sequence, while starting and stopping only affects the current status of the service.

> **Tip**
>
> To disable and stop in one command, we can use the `--now` option; for example, `systemctl disable firewalld --now`. This option can also be used to enable and start; for example, `systemctl enable firewalld --now`.

Let's reenable the service again and ensure it's running:

```
[root@rhel8 ~]# systemctl enable firewalld --now
Created symlink /etc/systemd/system/dbus-org.fedoraproject.
FirewallD1.service → /usr/lib/systemd/system/firewalld.service.
Created symlink /etc/systemd/system/multi-user.target.wants/
firewalld.service → /usr/lib/systemd/system/firewalld.service.
[root@rhel8 ~]# firewall-cmd --state
running
```

Now that we know how to start and stop, as well as enable and disable, the `firewalld` service, let's understand the configuration structure and learn how to interact with it by reviewing the default configuration.

Reviewing the different configuration items under firewalld

firewalld manages three concepts in its configuration:

- **Zones**: A firewalld zone is a group of rules that can be activated all together and assigned to a network interface. It includes different services and rules but also settings that alter the behavior of network traffic filtering.
- **Services**: A firewalld service is a port or group of ports that must be configured together for a specific system service (hence the name) to work properly.
- **Ports**: A firewalld port includes a port number (that is, `80`) and a type of traffic (that is, TCP) and can be used to manually enable network traffic to a custom system service.

firewalld manages two types of configurations:

- **Running**: The rules that have currently been applied to the system.
- **Permanent**: The rules that have been saved and will be loaded when the service starts.

> **Important Note**
> The concept behind running versus permanent is to try network filtering rules in a running system and, once ensured that they work well, save them as permanent ones. Remember to check that the rules you want in the system have been saved properly.

Now, let's check out our system and see which zones are available:

```
[root@rhel8 ~]# firewall-cmd --get-zones
block dmz drop external home internal nm-shared public trusted
work
```

Let's also check which zone is applied by default:

```
[root@rhel8 ~]# firewall-cmd --get-default-zone
public
```

Let's review the zones that are available in firewalld by looking at the following table:

Zone	Enabled services	Other characteristics
`public`	`cockpit`, `ssh`, and `dhcpv6-client`	Default zone in firewalld. Assigned by default to new network interfaces. Accepts incoming traffic related to outgoing connections.
`work`	`cockpit`, `ssh`, `ipp-client`, and `dhcpv6-client`	Accepts incoming traffic related to outgoing connections.
`internal`	`cockpit`, `ssh`, `mdns`, `ipp-client`, `samba-client`, and `dhcpv6-client`	Accepts incoming traffic related to outgoing connections.
`home`	`cockpit`, `ssh`, `mdns`, `ipp-client`, `samba-client`, and `dhcpv6-client`	Accepts incoming traffic related to outgoing connections. (Same as internal!)
`trusted`	N/A	Accepts all incoming traffic.
`external`	`ssh`	Accepts incoming traffic related to outgoing connections. Any IPv4 traffic that's forwarded through the interface with this zone assigned will be masqueraded, and look as though it originated from this machine.
`dmz`	`ssh`	Accepts incoming traffic related to outgoing connections.
`block`	N/A	Rejects all incoming traffic unless it is related to outgoing connections.
`drop`	N/A	Ignores all incoming traffic, not even notifying us that the connections haven't been accepted.

Important Note

You can always access the information about these zones, and more, by accessing the `firewalld.zones` manual pages available in the system by running `man firewalld.zones`. A good exercise is to review the aforementioned manual page.

The aforementioned services will be reviewed in more detail in the next section. For now, let's learn how to manage zones.

Let's change the default zone to `home`:

```
[root@rhel8 ~]# firewall-cmd --set-default-zone=home
success
[root@rhel8 ~]# firewall-cmd --get-default-zone
home
```

We can establish a `public` zone as the default and assign a `home` zone to our local network:

```
[root@rhel8 ~]# firewall-cmd --set-default-zone=public
success
[root@rhel8 ~]# firewall-cmd --permanent --zone=internal \
--add-source=192.168.122.0/24
success
[root@rhel8 ~]# firewall-cmd --reload
success
[root@rhel8 ~]# firewall-cmd --get-active-zones
internal
  sources: 192.168.122.0/24
public
  interfaces: enp1s0
```

This configuration allows us to publish services to the local network only, which is defined as `192.168.122.0/24` and assigned to the `internal` zone. Any service or port that's assigned to the `internal` zone from now on will be only accessible if it's accessed from an IP address in the internal network. We avoid allowing access to these services from other networks.

Also, to enable services to be accessed from any other network, we only need to assign them to the `public` zone.

Let's review the main options that are used and a couple more that could be useful:

- `--get-zones`: Lists the zones that have been configured in the system.
- `--get-default-zone`: Shows the zone that's configured by default.
- `--set-default-zone=<zone>`: Sets the default zone. This is applied to the running and permanent configuration
- `--get-active-zones`: Shows the zones being used what networks/interfaces they apply to.
- `--zone=<zone>`: Used to specify a zone for another option.
- `--permanent`: Used to apply the changes to the saved configuration. When you use this option, the changes will not be applied to the running configuration.
- `--reload`: Loads the saved configuration as running.
- `--add-source=<network>`: Adds a source network, in CIDR format, to a specified zone. The default zone is used if one hasn't been specified. Changes are applied to the running config; use `--permanent` to save them.
- `--remove-source=<network>`: Removes a source network, in CIDR format, to a specified zone. The default zone is used if one hasn't been specified. Changes are applied to the running config; use `--permanent` to save them.
- `--add-interface=<interface>`: Routes traffic from an interface to a zone. The default zone is used if one hasn't been specified.
- `--change-interface=<interface>`: Changes the traffic that's being routed to an interface to a zone. The default zone used if one hasn't been specified.

Although this list of options may be very helpful, the full list of options is available on the manual page for `firewall-cmd`. You should review this page as you will be using it often when you're reconfiguring your firewall options.

> **Tip**
>
> To view the `firewall-cmd` manual page, simply run `man firewall-cmd`.

Now that we are aware on what zones are and how they are selected, let's learn how to manage services and ports.

Enabling and managing services and ports

As we mentioned in the previous section, a **firewalld service** is a port or group of ports that are configured together for a specific system service (hence the name) to work properly. There are a set of services that are enabled by default in one or many of the available **firewalld zones**. Let's start by reviewing them:

- **ssh**: Provides access to the **Secure Shell** (**SSH**) service in the system, which also enables remote management. The traffic that's accepted goes to port `22` and is of the `TCP` type.
- **mdns**: Provides access to the **Multicast DNS** (**MDNS**) service that's used to announce services in the local network. Traffic is accepted to multicast address `224.0.0.251` (IPv4) or `ff02::fb` (IPv6), on port `5353`, and is of the `UDP` type.
- **ipp-client**: Provides access to the **Internet Printing Protocol** (**IPP**) client, which goes to port `631` and uses the `UDP` protocol.
- **samba-client**: This is a file and print sharing client that's compatible with Microsoft Windows. It uses ports `137` and `138` and is of the `UDP` type.
- **dhcpv6-client**: A **Dynamic Host Configuration Protocol** (**DHCP**) for IPv6. It's destination is the special network `fe80::/64`, its port is `546`, and it's of the `UDP` type.
- **cockpit**: The web management interface for RHEL. Its destination is port `9090` and it's of the `TCP` type.

As you can see, a firewalld service can specify more than one port, a target address, and even a target network.

Now, let's take a look at the services that have been configured in our firewall:

```
[root@rhel8 ~]# firewall-cmd --list-services
cockpit dhcpv6-client ssh
[root@rhel8 ~]# firewall-cmd --list-services --zone=internal
cockpit dhcpv6-client mdns samba-client ssh
```

Please note that when you're not establishing a zone, the services that are displayed are the related to the default zone – in this case, `public`. However, consider that we have configured more than one zone.

Now, let's install a web server – in this case, the Apache `httpd` server:

```
[root@rhel8 ~]# dnf install httpd -y
Updating Subscription Management repositories.
Last metadata expiration check: 0:25:05 ago on lun 01 mar 2021
17:02:09 CET.
Dependencies resolved.
================================================================
=====================================
Package        Arch   Version
Repository                       Size
================================================================
=====================================
Installing:
httpd          x86_64 2.4.37-30.module+el8.3.0+7001+0766b9e7
rhel-8-for-x86_64-appstream-rpms 1.4 M
Installing dependencies:
apr            x86_64 1.6.3-11.el8
rhel-8-for-x86_64-appstream-rpms 125 k
[omitted]
Installed:
  apr-1.6.3-11.el8.x86_64
  apr-util-1.6.1-6.el8.x86_64
  apr-util-bdb-1.6.1-6.el8.x86_64
  apr-util-openssl-1.6.1-6.el8.x86_64
  httpd-2.4.37-30.module+el8.3.0+7001+0766b9e7.x86_64
  httpd-filesystem-2.4.37-30.module+el8.3.0+7001+0766b9e7.
noarch
  httpd-tools-2.4.37-30.module+el8.3.0+7001+0766b9e7.x86_64
  mailcap-2.1.48-3.el8.noarch
  mod_http2-1.15.7-2.module+el8.3.0+7670+8bf57d29.x86_64
  redhat-logos-httpd-81.1-1.el8.noarch

Complete!
```

Let's enable and start the `httpd` service:

```
[root@rhel8 ~]# systemctl enable httpd --now
Created symlink /etc/systemd/system/multi-user.target.wants/
httpd.service → /usr/lib/systemd/system/httpd.service.
[root@rhel8 ~]# systemctl status httpd -n0
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service;
enabled; vendor preset: disabled)
   Active: active (running) since Mon 2021-03-01 17:31:57 CET;
8s ago
     Docs: man:httpd.service(8)
Main PID: 2413 (httpd)
   Status: "Started, listening on: port 80"
    Tasks: 213 (limit: 8177)
   Memory: 25.0M
   CGroup: /system.slice/httpd.service
           ├─2413 /usr/sbin/httpd -DFOREGROUND
           ├─2414 /usr/sbin/httpd -DFOREGROUND
           ├─2415 /usr/sbin/httpd -DFOREGROUND
           ├─2416 /usr/sbin/httpd -DFOREGROUND
           └─2417 /usr/sbin/httpd -DFOREGROUND
```

Now, let's check that the service is listening on all the interfaces:

```
[root@rhel8 ~]# ss -a -A "tcp" | grep http
LISTEN    0         128                 *:http
*:*
```

Optionally, we can check if the port is open by using an external machine (if we have one):

```
[root@external:~]# nmap 192.168.122.8
Starting Nmap 7.80 ( https://nmap.org ) at 2021-03-01 17:45 CET
Nmap scan report for rhel.redhat.lan (192.168.122.8)
Host is up (0.00032s latency).
Not shown: 998 filtered ports
PORT     STATE  SERVICE
22/tcp   open   ssh
9090/tcp closed zeus-admin
```

```
MAC Address: 52:54:00:E6:B4:A4 (QEMU virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 5.15 seconds
```

Now, we can enable the `http` service on the firewall:

```
[root@rhel8 ~]# firewall-cmd --add-service http \
--zone=public --permanent
success
[root@rhel8 ~]# firewall-cmd --add-service http \
--zone=internal --permanent
success
[root@rhel8 ~]# firewall-cmd --reload
success
[root@rhel8 ~]# firewall-cmd --list-services
cockpit dhcpv6-client http ssh
[root@rhel8 ~]# firewall-cmd --list-services --zone=internal
cockpit dhcpv6-client http mdns samba-client ssh
```

With that, the service has been enabled and the port is open. We can verify this from an external machine, like so (this is optional):

```
[root@external:~]# nmap 192.168.122.8
Starting Nmap 7.80 ( https://nmap.org ) at 2021-03-01 17:50 CET
Nmap scan report for rhel.redhat.lan (192.168.122.8)
Host is up (0.00032s latency).
Not shown: 997 filtered ports
PORT     STATE  SERVICE
22/tcp   open   ssh
80/tcp   open   http
9090/tcp closed zeus-admin
MAC Address: 52:54:00:E6:B4:A4 (QEMU virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 5.18 seconds
```

We can see the port `80` open now. We can also retrieve the main page from the web server and show the first line:

```
[root@external:~]# curl -s http://192.168.122.8 | head -n 1
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.
w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

> **Important Note**
> The definitions of the services in firewalld are kept in independent files in the `/usr/lib/firewalld/services` directory. If you need to out the details of a service, you can go there and inspect the file and its definition.

Now, let's try to remove the service from the public network, since this will be an internal service:

```
[root@rhel8 ~]# firewall-cmd --list-services --zone=public
cockpit dhcpv6-client http ssh
[root@rhel8 ~]# firewall-cmd --remove-service http \
--zone=public --permanent
success
[root@rhel8 ~]# firewall-cmd --reload
success
[root@rhel8 ~]# firewall-cmd --list-services --zone=public
cockpit dhcpv6-client ssh
```

Let's assume we didn't have the service definition and we still wanted to open port `80` on `TCP` in the `public` interface:

```
[root@rhel8 ~]# firewall-cmd --list-ports --zone=public

[root@rhel8 ~]# firewall-cmd --add-port 80/tcp --zone=public
--permanent
success
[root@rhel8 ~]# firewall-cmd --reload
success
[root@rhel8 ~]# firewall-cmd --list-ports --zone=public
80/tcp
```

We can review the ports and services in one go, like so:

```
[root@rhel8 ~]# firewall-cmd --list-all --zone=public
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp1s0
  sources:
  services: cockpit dhcpv6-client ssh
  ports: 80/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Now, we can remove the port:

```
[root@rhel8 ~]# firewall-cmd --list-ports --zone=public
80/tcp
[root@rhel8 ~]# firewall-cmd --remove-port 80/tcp --zone=public
--permanent
success
[root@rhel8 ~]# firewall-cmd --reload
success
[root@rhel8 ~]# firewall-cmd --list-ports --zone=public

[root@rhel8 ~]#
```

With this, we know how to add and remove services and ports to/from a firewall and check their statuses. Let's review the options that we can use for `firewall-cmd` to do so:

- `--zone=<zone>`: Used to specify a zone. When no zone is specified, the default one is used.
- `--list-services`: Displays a list of services for the specified zone.
- `--add-service`: Adds a service to the specified zone.

- `--remove-service`: Removes a service from the specified zone.
- `--list-ports`: Lists the open ports in the specified zone.
- `--add-port`: Adds a port to the specified zone.
- `--remove-port`: Removes a port from the specified zone.
- `--list-all`: Lists the ports, services and all configuration items associated with the specified zone.
- `--permanent`: Rules will be applied to the saved configuration instead of the running one.
- `--reload`: Reloads the rules from the saved configuration.

Now that we know how to assign services and ports to different zones in the firewall, let's take a look at how they are defined.

Creating and using service definitions for firewalld

Service definitions for firewalld are stored in the `/usr/lib/firewalld/services` directory. Let's take a look at a simple service, such as the `ssh` service stored in the `ssh.xml` file, which has the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>SSH</short>
  <description>Secure Shell (SSH) is a protocol for logging
into and executing commands on remote machines. It provides
secure encrypted communications. If you plan on accessing your
machine remotely via SSH over a firewalled interface, enable
this option. You need the openssh-server package installed for
this option to be useful.</description>
  <port protocol="tcp" port="22"/>
</service>
```

Here, we can see that we only need an XML file with three sections to describe a basic service:

- `short`: The short name for the service
- `description`: A long description of what the service does
- `port`: The port to be opened for this service

Let's say we want to install an Oracle database in our server. We must have the `1521` port open, and it must be of the `TCP` type. Let's create the `/etc/firewalld/services/oracledb.xml` file with the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>OracleDB</short>

  <description>Oracle Database firewalld service. It allows connections to the Oracle Database service. You will need to deploy Oracle Database in this machine and enable it for this option to be useful.</description>
  <port protocol="tcp" port="1521"/>
</service>
```

We can enable it by using the following code:

```
[root@rhel8 ~]# firewall-cmd --reload
success
[root@rhel8 ~]# firewall-cmd --add-service oracledb
success
[root@rhel8 ~]# firewall-cmd --list-services
cockpit dhcpv6-client oracledb ssh
```

Now, it's ready to be used in the running configuration. We can add it to the permanent configuration like so:

```
[root@rhel8 ~]# firewall-cmd --add-service oracledb --permanent
success
```

> **Tip**
>
> It would be infrequent to have to open more complex services. In any case, the manual page that describes how to create firewalld services is `firewalld.service` and can be opened by running `man firewalld.service`.

With this, we have an easy way to standardize the services to be opened in the firewalls of our systems. We can includes these files in our configuration repositories so that they can be shared with the whole team.

Now that we can create a service, let's take a look at an easier way to configure the firewall in RHEL; that is, using the web interface.

Configuring firewalld with the web interface

To use the RHEL web administrative interface of RHEL8, we must install it. The package and service running it are both called `cockpit`. We can install it by running the following code:

```
[root@rhel8 ~]# dnf install cockpit -y
Updating Subscription Management repositories.
[omitted]
Installing:
cockpit                       x86_64 224.2-1.el8
rhel-8-for-x86_64-baseos-rpms      74 k
[omitted]
  cockpit-224.2-1.el8.x86_64
  cockpit-bridge-224.2-1.el8.x86_64
  cockpit-packagekit-224.2-1.el8.noarch
  cockpit-system-224.2-1.el8.noarch
  cockpit-ws-224.2-1.el8.x86_64

Complete!
```

Now, let's enable it:

```
[root@rhel8 ~]# systemctl enable --now cockpit.socket
Created symlink /etc/systemd/system/sockets.target.wants/
cockpit.socket → /usr/lib/systemd/system/cockpit.socket.
```

> **Tip**
>
> Cockpit uses a clever trick to save resources. The interface is stopped but a socket is enabled to listen on port `9090`. When it receives a connection, cockpit is started. This way, it will only consume resources in your machine when it is in use.

Now, let's learn how to add the `DNS` service to the `public` zone.

Let's access cockpit by pointing a browser to the IP of the machine and port `9090` – in this case, `https://192.168.122.8:9090`. Let's log in as `root` with the password that was provided during installation:

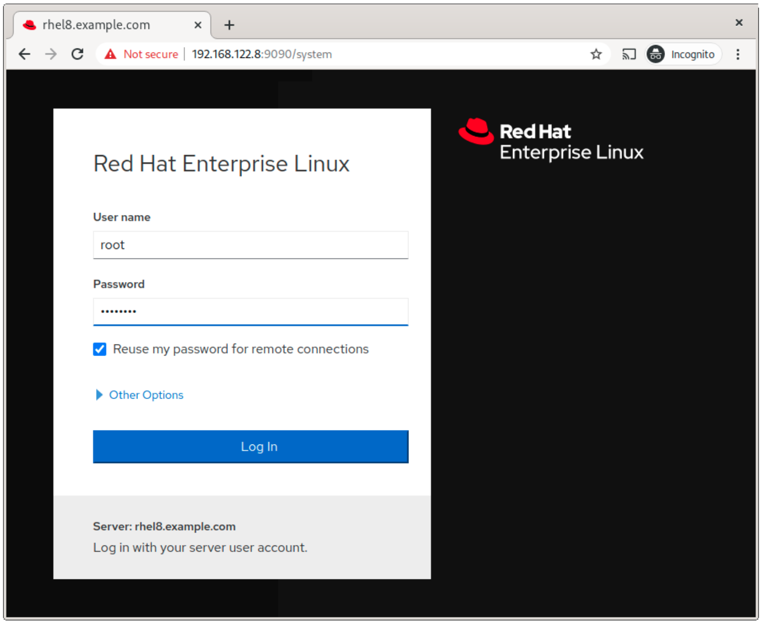


Figure 9.2 – Cockpit login screen

Now, we can access the cockpit dashboard, which contains information about the system:

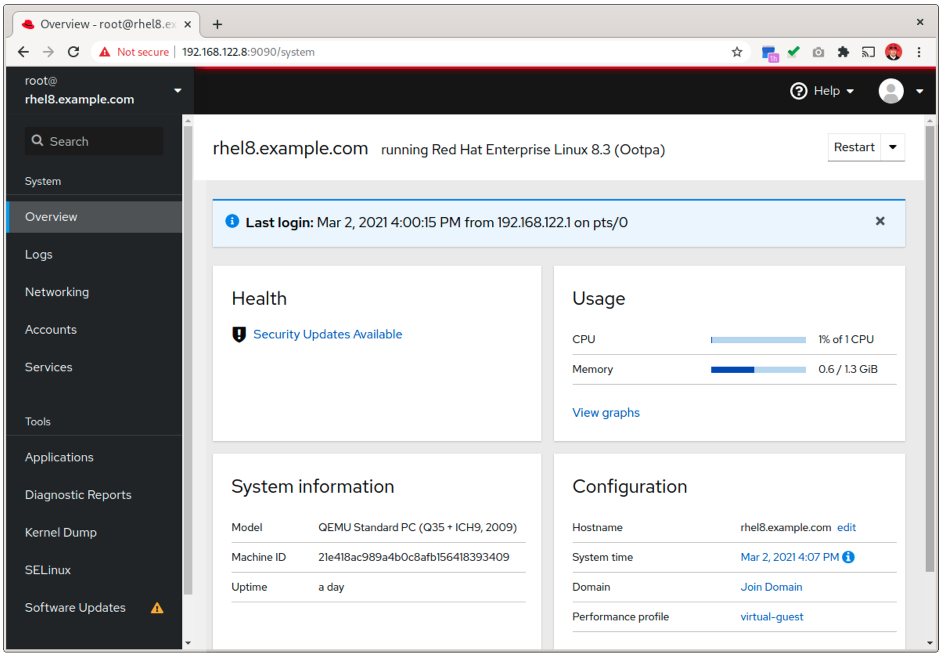


Figure 9.3 – Cockpit initial screen and dashboard

Now, let's go to **Networking** and then click on **Firewall**, as shown in the following screenshot:

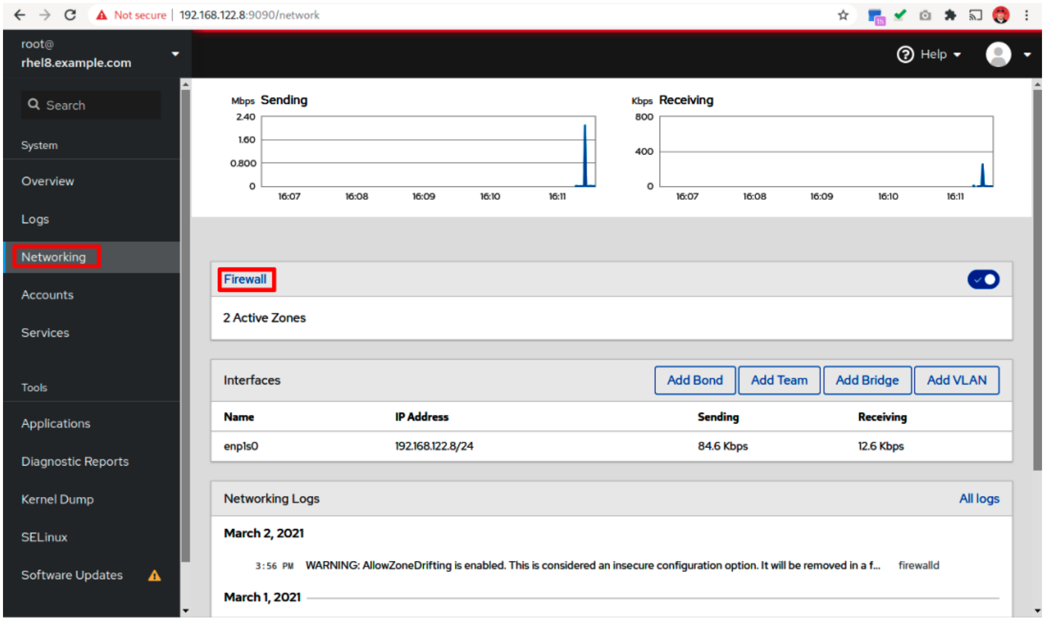


Figure 9.4 – Cockpit accessing the firewall configuration

At this point, we can click on **Add Services** in the **public zone** section to modify it and add one more service:

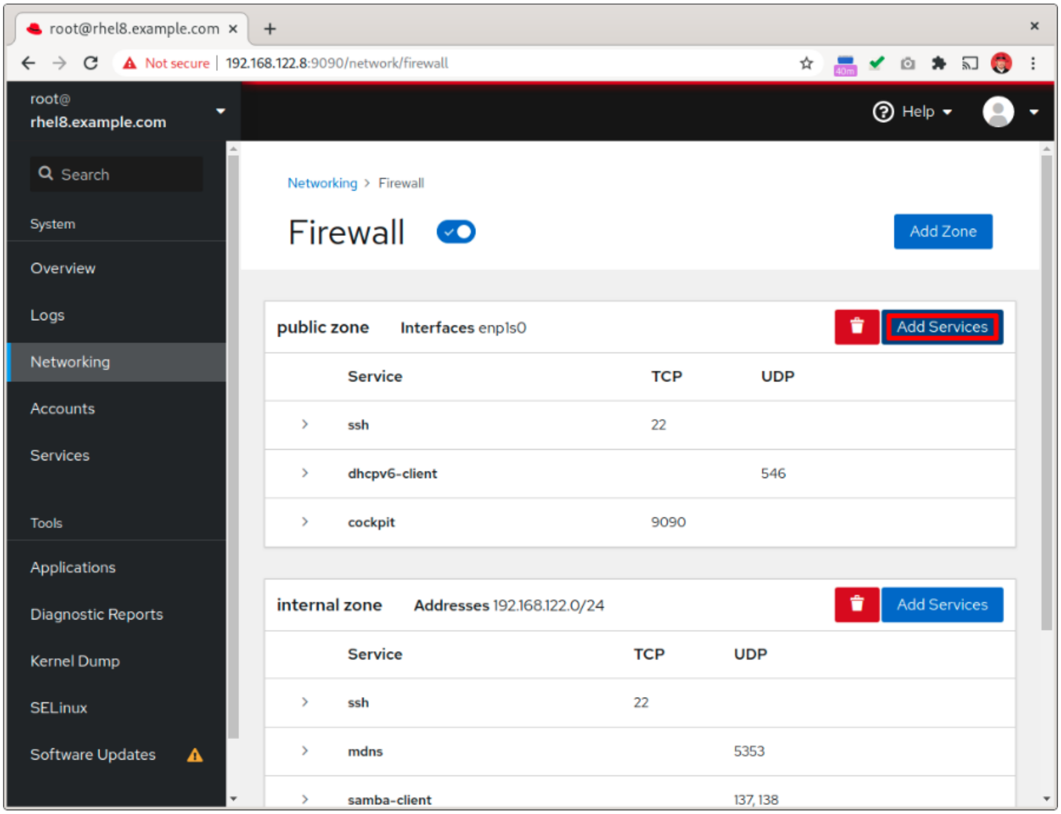


Figure 9.5 – Cockpit firewall configuration interface

The steps to add the **dns** service to the **public zone** section of the firewall are simple:

1. Click on **Services**.
2. Filter the services by typing `dns` into it.
3. Select the **dns** service with **TCP:53** and **UDP:53**.

4. Click on **Add Services**:

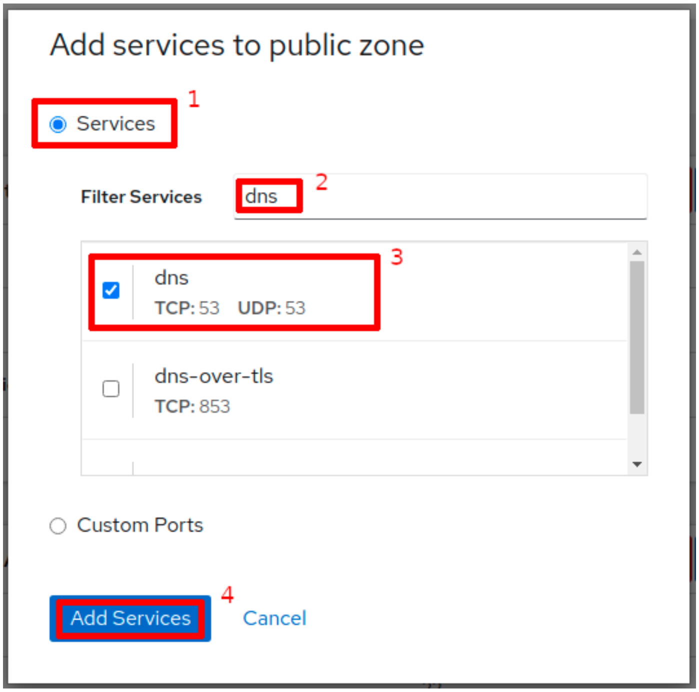


Figure 9.6 – Cockpit firewall – adding a service to a public zone

Once you've done this, the service will be added to the running and permanent configurations. It will be displayed on the **public zone** section of cockpit:

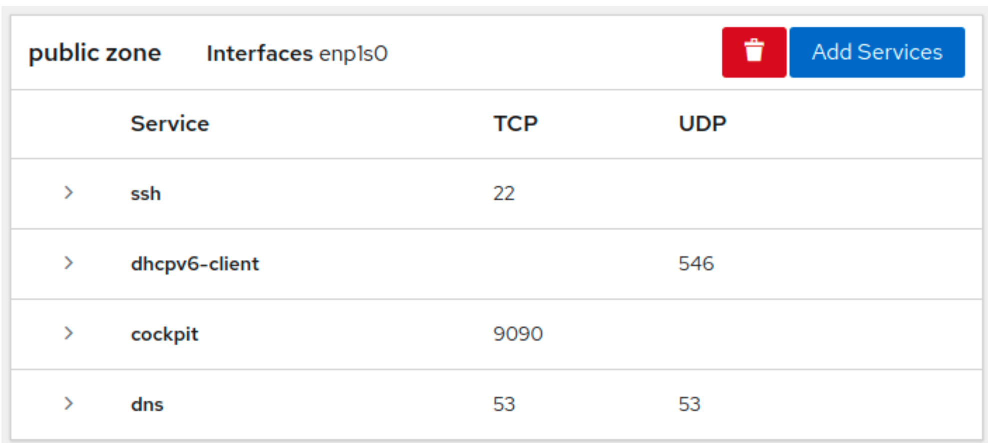


Figure 9.7 – Cockpit firewall – the result of a service DNS being added to a public zone

With this, we know how to make modifications to the firewall in RHEL8 using a web interface. We'll leave it as an exercise for you to remove and redo the configuration we did with the command line at the beginning of this chapter, but with the web interface instead.

Summary

Security is a very important part of system administration. Disabling the security measures on a system just because it's in an isolated network goes against the defense in depth principle, so this is heavily discouraged.

In this chapter, we saw how simple and easy it is to configure a firewall using firewalld, in RHEL8, thus providing us with another tool to manage, filter, and secure the network connections in our system. We also worked with cockpit, a web administration tool that makes this task more visual and easier to perform.

We can now take control of the network connectivity of our systems, provide access to the services we want to provide, and add a layer of security to them. We also know how to manage zones and how to use them, depending on our system's use case. We are now ready to define our own custom services so that we can always filter network connectivity for them. We can now also deploy more secure systems by using the firewall included in RHEL.

Now, we are ready to learn more about security in RHEL, which is what we will do in the next chapter. Remember, security is a team sport, and the system administrators are key.

10
Keeping Your System Hardened with SELinux

In this chapter, we are going to familiarize ourselves with SELinux. SELinux has been around for a while, but a lack of understanding regarding how it works leads many people to suggest disabling it.

This is not something we want, as it would be similar to telling a user to forego a password because it is hard to remember.

We will introduce the origins of SELinux, and what the default modes and policies are. Then, we will understand how SELinux applies to our files, folders, and processes, and how to restore them to the system defaults.

Additionally, we will explore how to fine-tune the policies using Booleans and troubleshoot common issues with the help of the following sections:

- SELinux usage in enforcing and permissive modes
- Reviewing the SELinux context for files and processes
- Tweaking the policy with semanage

- Restoring changed file contexts to the default policy
- Using SELinux Boolean settings to enable services
- SELinux troubleshooting and common fixes

By the end, we will better understand how to use SELinux properly and how to benefit from the additional protection that it provides to our system.

During the chapter, there will be detailed explanations of how SELinux works to aid our understanding of the way it operates, even if using it, in reality, is a lot simpler. We will also use these examples to illustrate cases where SELinux prevents attacks or misconfigurations.

Let's get hands-on with SELinux!

Technical requirements

It is possible to continue the practice of using the virtual machine created at the beginning of this book in *Chapter 1*, *Installing RHEL8*. Any additional packages required for this chapter will be indicated alongside the text and can be downloaded from `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration`.

SELinux usage in enforcing and permissive modes

Security-Enhanced Linux (**SELinux**) was introduced in December of 2000 via the Linux-Kernel mailing list as a product started by the **National Security Agency** (**NSA**) to improve the security of the operating system by means of mandatory access controls and role-based access control, as opposed to the traditional discretionary access controls that were available in the system.

Before SELinux was introduced in the Linux kernel, discussions took place regarding the proper way to do it, and finally, a kernel framework named **Linux Security Modules (LSM)** was introduced and SELinux was implemented using it so that other approaches could use LSM, too, and not just SELinux.

SELinux provides security improvements to Linux as access to files made by users, processes, or even other resources can be controlled in a very granular way.

Let's take one example to make it clearer when SELinux comes into play: when a web server is serving pages from users, it reads files from the user's home directory inside the `public_html` or `www` folders (the most standard ones). Being able to read files from the user's home directory can reveal the contents in the event that the web server process is hijacked by an attacker, and this precise moment is when SELinux comes into play, as it will automatically block access to files that should not be accessible for a web server.

SELinux then confines the processes and services to only perform what they are supposed to, and only using the resources that are authorized. This is a really important feature that keeps things under control, even in the event of software bugs that may lead to unexpected files or resources being accessed. SELinux will block it if it has not been authorized by the active policy.

> **Important tip**
>
> SELinux permissions always arise following regular **Discretionary Access Controls** (**DAC**) if a user has no access to a file because of improper file permissions. SELinux has nothing to do there.

By default, the system installation should deploy it in `enforcing` mode and using the `targeted` policy. It is possible to check your current system status via execution of `sestatus`, as shown in the following screenshot:

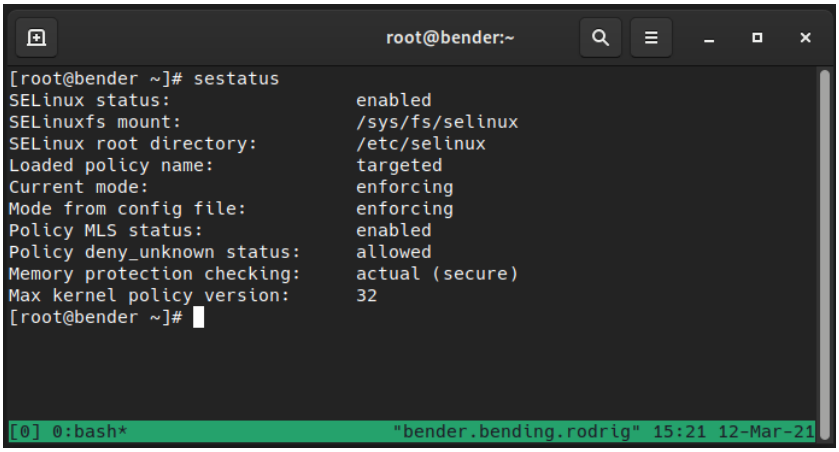


Figure 10.1 – Output of sestatus for our system

As we can see, our system has SELinux `enabled`, using the `targeted` policy, and is currently `enforcing`. Let's learn about what this means.

SELinux works by defining a **policy**, that is, a set of predefined rules for granting or denying access to resources. The ones available can be listed via `dnf list selinux-policy-*` in your system, with `targeted` and `mls` being the most common ones.

We will focus on the targeted policy, but to make an analogy regarding `mls`, the **Multi-Level Security** (**MLS**) policy, it is about allowing users to interact based on their security clearance, similar to what we can see in movies where someone has clearance to know some information, but not other people. How does this apply to a system? Well, the root user might have access to perform certain actions but not others, and if the user became root via `su` or `sudo`, they would still have the original label attached so permissions could be reduced if the root login happened over a local terminal or a remote connection and `sudo` execution.

The mode, listed as `enforcing`, means that the policy is currently being enforced, which is the opposite of `permissive`. We can consider this as being active and offering protection, while permissive entails being active but only providing a warning, and not offering protection.

Why do we have `permissive` instead of just disabling it? This question is a bit tricky, so let's explain a bit more about how it works to provide a better answer.

SELinux uses extended attributes in the filesystem to store the labels. Each time a file is created, a label is assigned based on the policy, but this only happens while SELinux is active, so this makes SELinux `disabled` different from SELinux `permissive`, because the first one will not create those labels for the new files created.

Additionally, SELinux in `permissive` mode allows us to see the errors that will be raised if a program has not received a good policy for it or if a file has no proper labels.

It is really easy to switch from `enforcing` to `permissive` and vice versa and always via the `setenforce` command, while we can use `getenforce` to retrieve the current status, as we can see in the following screenshot:

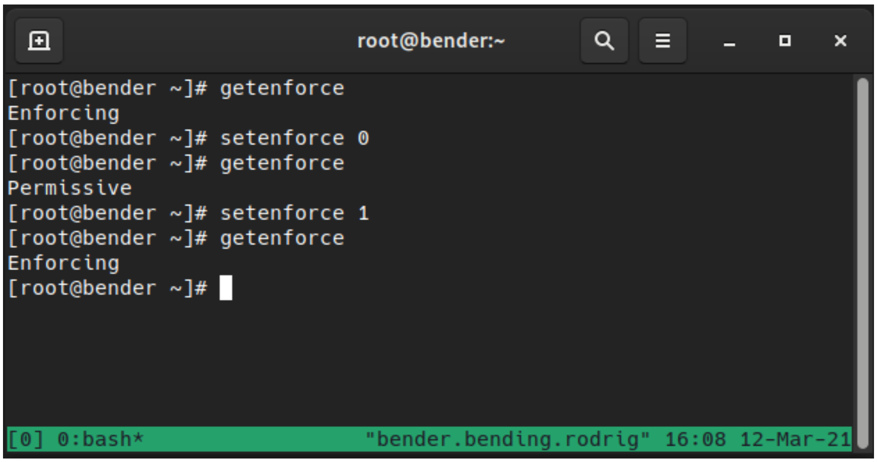


Figure 10.2 – Changing SELinux enforcing status

It might look basic, but it really is as easy as that, a matter of running a command. However, if the status was disabled, it would be a completely different story.

SELinux status is configured by editing the `/etc/selinux/config` file, but changes only take effect after a system reboot; that is, we can switch from `enforcing` to `permissive` in real time or from permissive to enforcing, but when changing the policy from `disabling` to `enabling`, or vice versa, SELinux will require us to reboot the system.

The general advice is to leave SELinux in enforcing mode, but if, for whatever reason, it was disabled, the recommendation is to switch SELinux to `permissive` as the first step when moving from `disabled`. This will allow us to check that the system actually works without being locked out of it because of a kernel blocking access to files and resources.

> **Note**
>
> During the reboot after switching from `disabled` to `permissive` or `enforcing`, the system will force a relabeling of the filesystem based on the policy. This is accomplished by the creation of a file in the root folder of our filesystem named `/.autorelabel`, which will trigger the process and reboot again afterward.

But why opt for disabling instead of `permissive`? For example, some software might require to set it in disabled mode even if later, it can be re-enabled for operations or for other reasons, but bear in mind that SELinux is a security feature that protects your system and should be kept.

Keep in mind that SELinux uses **Access Vector Cache** (**AVC**) messages that are logged to the `/var/log/audit/audit.log` file as well as system journals, and yes, it's a cache, so rules are not checked as frequently so as to speed up operations.

Let's go back to the idea of the filesystem storing labels and let's jump into the next section to see how they relate to processes, files, and the RBAC provided by SELinux.

Reviewing the SELinux context for files and processes

SELinux uses labels, also referred to as the security context attached to each file, and defines several aspects. Let's check one example in our home folder with the `ls -l` command, but with a special modifier, `Z`, that will show SELinux attributes as well, as we can see in the following screenshot:

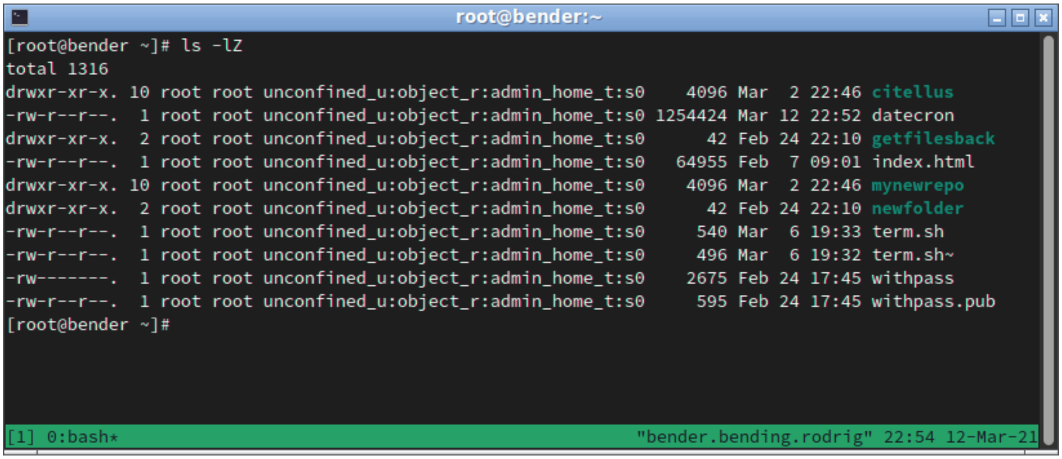


Figure 10.3 – File listing showing SELinux attributes

Let's focus on the output for one of the files:

```
-rw-r--r--.  1 root unconfined_u:object_r:admin_home_t:s0
540 Mar  6 19:33 term.sh
```

The SELinux attributes are the ones listed as `unconfined_u:object_r:admin_home_t:s0`:

- **The first part is the user mapping**: `unconfined_u`
- **The second part is the role**: `object_r`
- **The third part is the type**: `admin_home_t`
- **The fourth part is used for the level**: `s0` in multi-level security and multi-category security

Something similar happens with processes, and similarly, we can append `Z` to many of the common commands to get the contexts, for example, with `ps Z`, as we can see in the following screenshot:

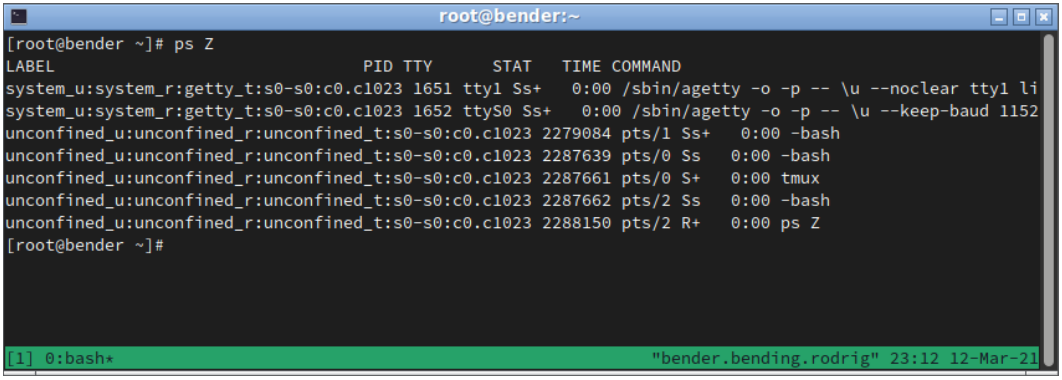


Figure 10.4 – ps output with SELinux contexts

Again, let's examine one of the lines:

```
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2287661
pts/0 S+   0:00 tmux
```

Again, we can see the same approach: user, role, type, and level for multi-level security and multi-category security.

Now that we've introduced what it looks like, let's focus on how it works in the targeted policy.

The targeted policy allows everything to run as if SELinux was not enabled in the system, except for the services targeted by it. This makes a good compromise between security and usability.

During development of the policy, new services are added, while others are refined, and many of the most common services have policies written for protecting them.

SELinux also features something named **transitions**. A transition allows one process started by a user, with a binary with some specific role, to transition via the execution into some other role, which is used later to define what the permissions are for it.

As you might imagine, our user also has a SELinux context, and similarly, we can use the `id -Z` command for checking it:

```
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

So, going back to the first example, Apache Web Server is provided by the `httpd` package, which can be installed via `dnf -y install httpd`. Once installed, let's start it with `systemctl start httpd` and enable it with `systemctl enable httpd`, and then open the firewall with `firewall-cmd --add-service=http` and `firewall-cmd --add-service=https`, as we've done with other services in previous chapters.

Previous commands can be found in the following script: `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/blob/main/chapter-10-selinux/apache.sh`.

Let's see how all that comes into play in the following screenshot:

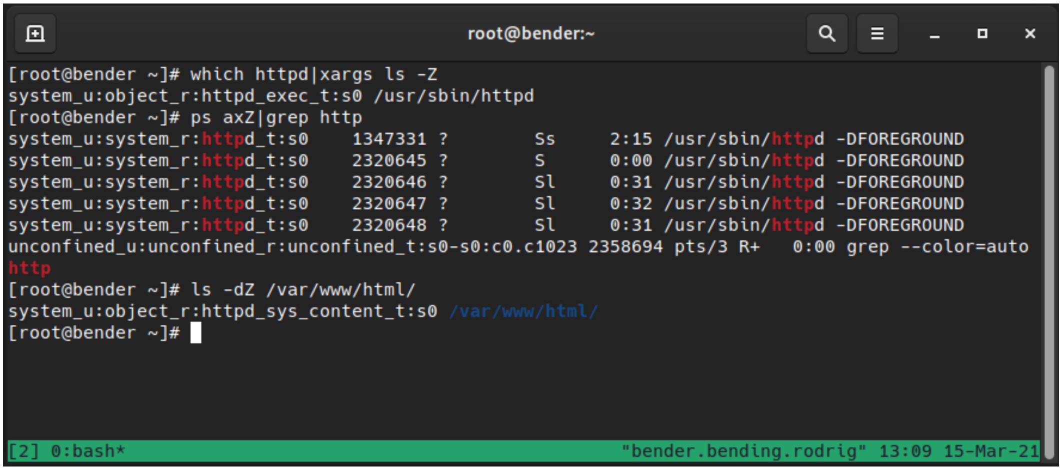


Figure 10.5 – Web server SELinux contexts

Here, we can see how the executable on disk has the context `httpd_exec_t`, the process is `httpd_t`, and the files/folder served by it is `httpd_sys_content_t`, and it works!

Let's now create an `index.htm` file in our `home` folder and move it to the `Apache Web Root` folder as follows:

```
# echo '<html><head><title>Our test</title></head><body>This is our test html</body></html>' > index.htm
```

```
# cp index.htm /var/www/html/index2.htm
```

```
# mv index.htm /var/www/html/index1.htm
```

Let's see what happens when we try to access the files as shown in the following screenshot:

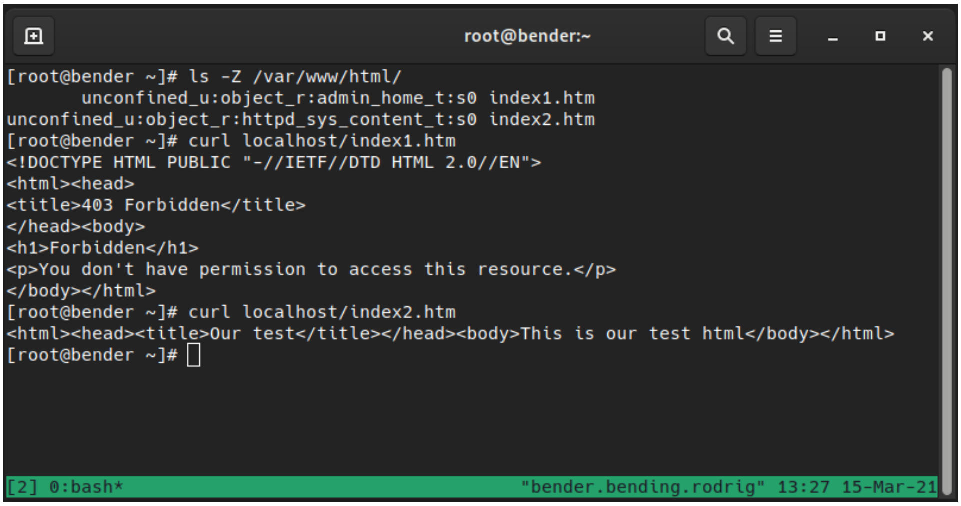


Figure 10.6 – Apache behavior with the generated files

As we can see, each file has one SELinux context, but on top of that, Apache is denying access to the one we moved (`index1.htm`), but showing the contents for the one we copied (`index2.htm`).

What has happened here? We copied one file and move the other, out of the same source, but they got two different SELinux contexts.

Let's extend the test as shown in the following screenshot:

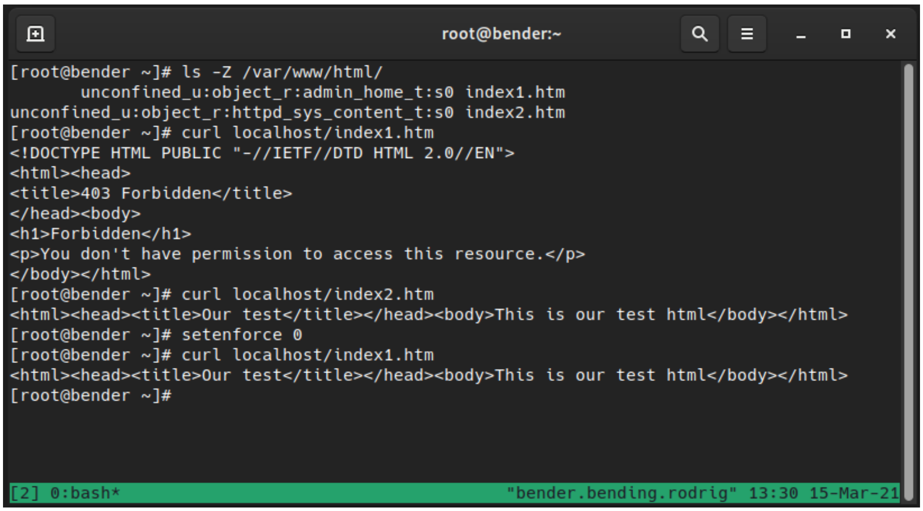


Figure 10.7 – Retrying with SELinux in permissive mode

As we can see in the preceding screenshot, we are now able to access file contents, so you could say *"What is wrong with SELinux that does not allow my site to work?"*, but the right way to express it would be *"Look how SELinux has protected us from disclosing a personal file on a website"*.

If, instead of directly moving a file into the Apache's **DocumentRoot** (`/var/www/html`), it was an attacker trying to reach our home folder files, SELinux would have denied those accesses by default. The `httpd_t` process cannot access the `admin_home_t` context.

A similar thing happens when we try to get Apache or any other service under the targeted policy to listen on a port that is not the one configured by default, and the best way to get familiar with what we can or cannot do is to learn about the `semanage` utility.

Using `semanage`, we can list, edit, add, or delete the different values in the policy, and even export and import our customizations, so let's use it to learn a bit more about it using our example with `httpd`.

Let's learn about `semanage` in the following section.

Tweaking the policy with semanage

As we introduced earlier, the targeted policy contains some configurations that are enforced for the services it has defined, allowing the protection of those services while not interfering with the ones it does not know about.

Still, sometimes we need to tweak a number of settings, such as allowing `http` or the `ssh` daemon to listen on alternate ports or accessing some other file types, but without losing the additional layer of protection provided by SELinux.

First, let's ensure that `policycoreutils` and `policycoreutils-python-utils` are installed in our system with `dnf –y install policycoreutils-python-utils policycoreutils` as they provide the tools we will use in this and the next sections of this chapter.

Let's learn with the help of an example. Let's see which ports `httpd_t` can access with `semanage port -l|grep http`:

```
http_cache_port_t              tcp      8080, 8118, 8123,
10001-10010
http_cache_port_t              udp      3130
http_port_t                    tcp      80, 81, 443, 488, 8008,
8009, 8443, 9000
```

As we can see, `http_port_t`, used by Apache Daemon, is allowed, by default, to use the ports `80`, `81`, `443`, `488`, `8008`, `9009`, `8443`, and `9000` via `tcp`.

That means that if we want to run Apache on any of those ports, no changes to policy will be required.

If we repeat the command but for `ssh`, we only see port `22` opened (executing `semanage port -l|grep ssh`):

```
ssh_port_t                     tcp      22
```

For example, we might want to add another port, let's say `2222`, to the list of possible ports, so that we hide the standard one being tested by port scanners. We will be able to do it via `semanage port -a -p tcp -t ssh_port_t 2222` and then validate with the prior command `semanage port –l|grep ssh`, which now shows the following:

```
ssh_port_t                     tcp      2222, 22
```

As we can see, port `2222` has been added to the list of available ports for the `ssh_port_t` type, and that enables the `ssh` daemon to start listening on it (this, of course, requires additional configuration for the `ssh` daemon configuration and to the firewall before we get a working service).

In the same way, for example, some web services require writing to specific folders for storing configurations, but by default, the context on `/var/www/html` is `httpd_sys_content_t`, which does not allow writing to disk.

We can check the available file contexts with `semanage fcontext –l` in a similar way to what we did with the ports, but the list of files is huge, as a web server might use common locations such as `logs` and `cgi-bin`, as well as filesystem files for certificates, configuration, and home directories, and extensions such as PHP and others. When you check the contexts with the preceding command, pay attention to the different types that are available and what the structure is for one listing, for example:

```
/var/www/html(/.*)?/wp-content(/.*)?                all files
system_u:object_r:httpd_sys_rw_content_t:s0
```

As we can see, there is a regular expression that matches the files in the `wp-content` folder inside the `/var/www/html` path applying to all files and sets a SELinux context of `httpd_sys_rw_content_t`, which allows read-write access. This folder is used by the popular blog software **WordPress**, so the policy is already prepared for covering some of the most popular services, folders, and requirements without requiring system administrators to write them ad hoc.

When invoking `semanage`, it will output that it has some subcommands we can use, such as the following:

- `import`: This allows the importing of local modifications.
- `export`: This allows the exporting of local changes.
- `login`: This allows the login and SELinux user associations to be managed.
- `user`: This manages SELinux users with roles and levels.
- `port`: This manages port definitions and types.
- `ibpkey`: This manages InfiniBand definitions.
- `ibendport`: This manages end port InfiniBand definitions.
- `interface`: This defines network interface definitions.
- `module`: This manages policy modules for SELinux.
- `node`: This manages definitions of network nodes.
- `fcontext`: This manages file context definitions.
- `boolean`: This manages Booleans for tweaking policies.
- `permissive`: This manages the enforcing mode.
- `dontaudit`: This manages the `dontaudit` rules in the policy.

For each one of the preceding commands, we can use the `-h` argument to list, help, and learn about the extra arguments that can be used for each one.

For the day-to-day use case, most of the time we'll be using `port` and `fcontext` as those will cover extending or tuning the available services that come with Red Hat Enterprise Linux, like the example we have showcased with `ssh` listening on an additional port.

> **Important tip**
>
> Traditionally, **Red Hat Certified System Administrator** (**RHCSA**) and **Red Hat Certified Engineer** (**RHCE**) courses used to have a reboot validation. This meant that for each service that was installed and started, it was also mandatory to remember to enable it to be active on the next reboot. A similar thing happens with SELinux. If we are adding a piece of software that will stay in our system, the best approach is to define, via `semanage`, `regexp` for the path that will be used. When following this approach, if the filesystem is relabeled or the context restored, the application will continue to work.

Let's see how to manually set the context for files and how to restore the defaults in the next section.

Restoring changed file contexts to the default policy

In the previous section, we mentioned how `semanage` enables us to perform changes to the policy, which is the recommended way to perform changes and to persist them for future files and folders, but that is not the only way we can perform operations.

From the command line, we can use the `chcon` utility to change the context for a file. This will allow us to define the user, the role, and the type for the file we want to alter, and similar to other filesystem utilities such as `chmod` or `chown`, we can also affect files recursively, so it's easy to set a full folder hierarchy to the desired context.

One feature that I always found very interesting is the ability to copy the context of a file via the `--reference` flag, so that the same context as the referenced file is applied to the target one.

When we were introducing the example of `httpd` earlier in this chapter, we did a test with two files, `index1.htm` and `index2.htm`, that were moved and copied to the `/var/www/html` folder. To go deeper into this example, we will make additional copies of `index1.htm` to demonstrate in the next screenshot the usage of `chcon`. Bear in mind that creating the files directly in the `/var/www/html` folder will set the files to have the proper context, so we need to create them at `/root` and then move them to the target folder, as we can see in the following screenshot:

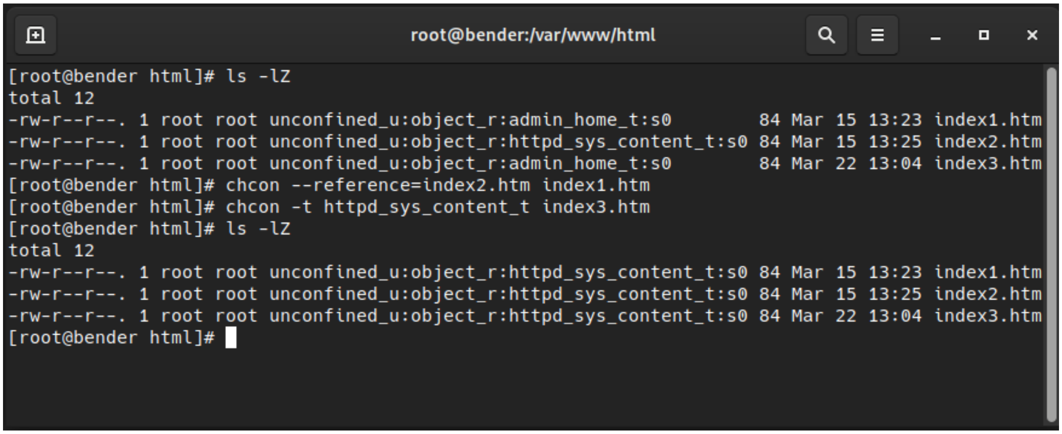


Figure 10.8 – Demonstrating chcon usage

As we can see, both the `index1.htm` and `index3.htm` files now have the proper context, in the first case, using the reference, and in the second, defining the type to use.

Of course, this is not the only method. As we indicated earlier, the recommended way for setting context for applications is to define the `regexps` path via `semanage`, and this empowers us to use the `restorecon` command to apply the right context, according to the configuration, to the files. Let's check how it operates in the following screenshot:

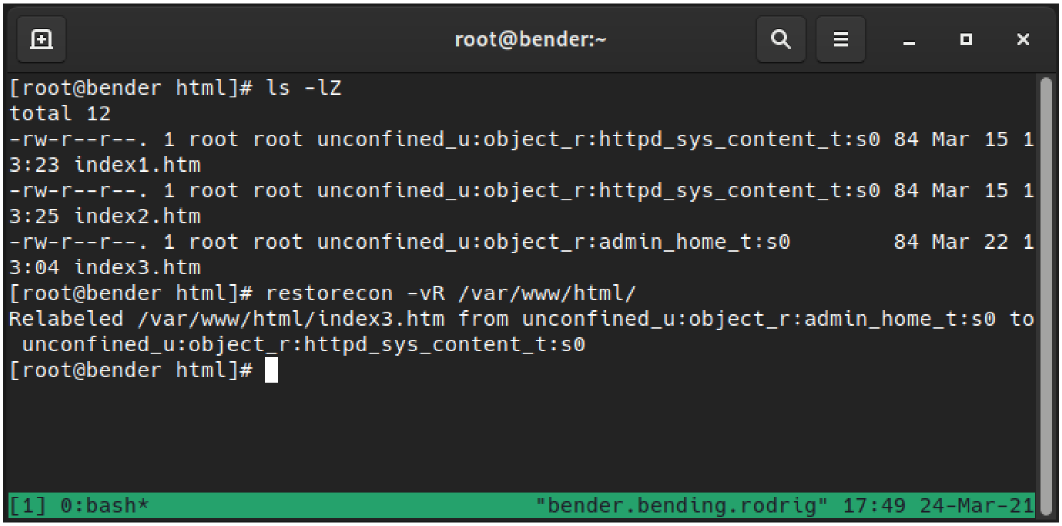


Figure 10.9 – Using restorecon to restore context

As we can see, we used `restorecon -vR /var/www/html/` and it automatically changed the `index3.htm` file into `httpd_sys_content_t`, which is defined for that folder as we saw when we were testing `semanage` to list the contexts. The arguments used, `v` and `R`, make the utility report the changes (verbose) and work recursively on the paths provided.

Let's say we have messed the system up by running `chcon` over the root filesystem. What would be the way to fix it? In this case, as we mentioned earlier, we should do the following:

- Set the operation mode to `permissive` to not block further access via `setenforce 0`.
- Put the marker to have the filesystem relabeled via `touch /.autorelabel`.
- Modify the `/etc/selinux/config` file to set the boot mode to `permissive`.
- Reboot the system to let relabeling happen.
- Once the system reboots, edit `/etc/selinux/config` again to define the operation mode as `enforcing`.

By operating in this way, instead of just running `restorecon -R /`, we are making sure that the system is operational and will continue to operate after reboot and a full relabel is applied to the filesystem, so it is left ready to re-enable the `enforcing` mode safely.

In the next section, let's see how to tune the policy within itself, using the Booleans to tune how it works.

Using SELinux Boolean settings to enable services

Many services have a wide range of configuration options for many common cases, but not always the same. For example, the `http` server should not access user files, but at the same time, it's a common way of operation to enable personal websites from the www or `public_html` folders in each user's home directory.

To overcome that use case and, at the same time, provide enhanced security, the SELinux policy makes use of Booleans.

A Boolean is a tunable that can be set by the administrator that can enable or disable conditionals in the policy code. Let's see, for example, a list of Booleans available for `httpd` by executing `getsebol -a|grep ^http` (list reduced):

```
httpd_can_network_connect --> off
httpd_can_network_connect_db --> off
httpd_can_sendmail --> off
httpd_enable_homedirs --> off
httpd_use_nfs --> off
```

This list is a reduced subset of the Booleans available, but it does give us an idea of what it can accomplish; for example, `http`, by default, cannot use the network to connect to other hosts, or send an email (usually done in PHP scripts), or can't even access home folders for users.

For example, if we want to enable users in a system to publish their personal web pages from the www folder in their home directory, that is `/home/user/www/`, we will have to enable the `httpd_enable_homedirs` Boolean by running the following command:

```
setsebool -P httpd_enable_homedirs=1
```

This will tweak the policy to enable `http` to access the user's home directory to serve the pages there. If, additionally, the servers will be stored on a **Network FileSystem** (**NFS**) or **Common Internet FileSystem** (**CIFS**) mount, additional Booleans will be required. We're still using the same targeted policy, but we've enabled the internal conditionals to permit that the accesses are not to be blocked by SELinux.

> **Important tip**
>
> The `-P` parameter to `setsebool` is required to make the change *permanent*. That means writing the change so that it is persisted; without it, the change will be lost once we restart our server.

As we've seen, `getsebool` and `setsebool` allow us to query and to set the values for the Booleans that tune the policy, but also, `semanage boolean -l` can help here, as we can see in the following screenshot:

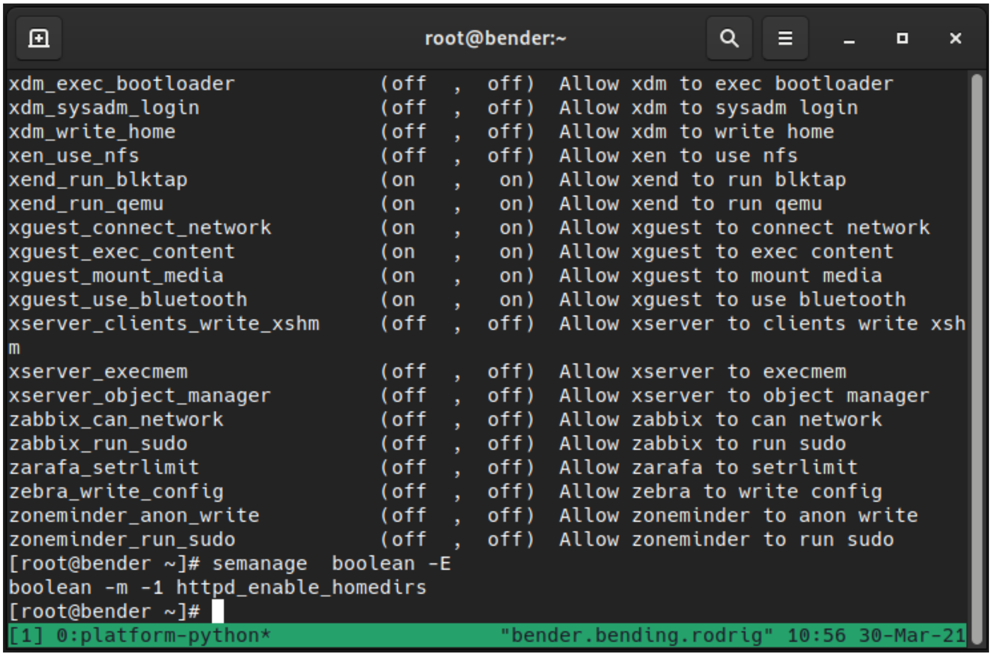


Figure 10.10 – Using semanage to manage Booleans

In the previous screenshot, we can see not only the Boolean we edited using `setsebool`, but also a description of the intended behavior.

One of the benefits is that `semanage`, as we introduced, allows us to export and import the local changes to the policy, so any customization made can be exported and imported to another system to ease the setup of similar server profiles.

All the possible Booleans in the policy can be checked with `semanage boolean -l`, similar to what we did to list the binding ports for applications in our `http` example.

We have learned about using Booleans to tune how the policy is adapting to some specific but pretty common cases. Next, we will explore probably the most frequently used part for administrators, that is, troubleshooting, but with the focus on SELinux.

SELinux troubleshooting and common fixes

One of the main problems in getting used to SELinux is that many people who are not familiar with it blame it for things not working; however, this argument is getting a bit outdated: SELinux was introduced in Red Hat Enterprise Linux 4, which was back in 2005.

Most of the time, issues with SELinux and our systems are related to changed file contexts and changing ports for services, and fewer time issues with the policy itself.

First of all, there are several places where we can check for errors, but in our list we should start with the audit log or the system messages. For example, we can start with the `/var/log/audit/audit.log` file that we introduced earlier in this chapter.

Also bear in mind that SELinux **Mandatory Access Control** (**MAC**) only plays once we have cleared access from regular **Discretionary Access Controls** (**DAC**), that is, if we've no permission to check a file (for example, mode 400 and our user not being the owner). In this case, it's highly unlikely that SELinux is blocking access.

Most of the time, our system will have installed the `setroubleshoot-server` and `setroubleshoot-plugins` packages that provide several tools, including `sealert`, to query the received SELinux messages and many times, also, to suggest changes.

Let's cover some of the basics that we should always validate:

- Review all the other controls (user and group ownership and permissions are properly set).
- Do not disable SELinux.

  If a program is not working properly and it was shipped with the OS, it might be a bug and should be reported via a support case or Bugzilla at `https://bugzilla.redhat.com`.

  Only if a program is not working properly might it be made to run unconfined, but leaving all remaining system services protected via the targeted policy.

- Think about what was done before the error happened if this is an existing program.

  Perhaps files were moved instead of copied or created upon reaching their destination, or perhaps the ports or folders for the software were changed.

Having arrived at this point, we should check `audit.log` for relevant messages. For example, as regards the example we mentioned regarding the wrong context with the files in `/var/www/html/`, an example audit entry would be as follows:

```
type=AVC msg=audit(1617210395.481:1603680): avc:
denied  { getattr } for  pid=2826802 comm="httpd"
path="/var/www/html/index3.htm" dev="dm-0"
ino=101881472 scontext=system_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:admin_home_t:s0 tclass=file
permissive=0
```

It looks strange, but if we check the parameters, we see the path of the affected file, the PID, the source context (`scontext`), and the target context (`tcontext`), so in brief, we can see that `httpd_t` tried to access (get attributes) for a target context `admin_home_t` and that was denied.

At the same time, if we're using `setroubleshoot`, we will get a message like this in the system journal:

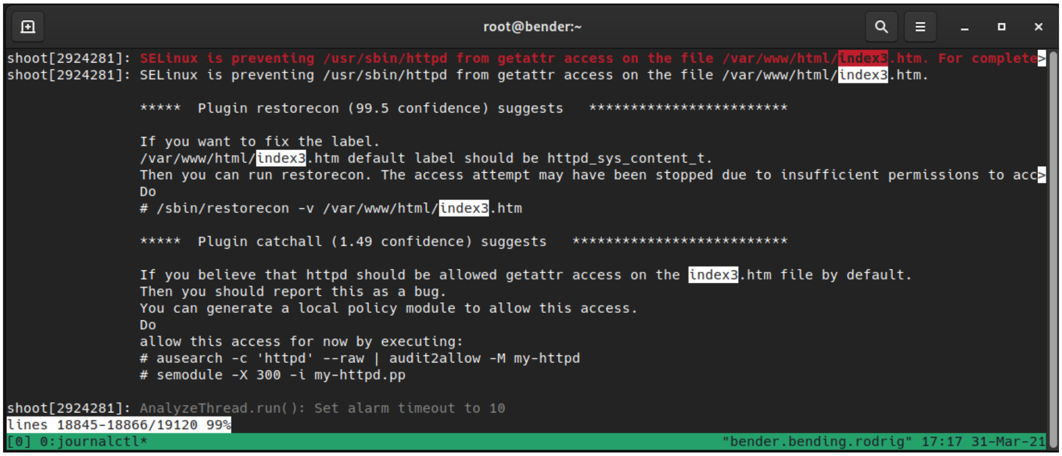


Figure 10.11 – setroubleshoot logging in the system journal

As we can see in the preceding screenshot, it already identified that one of the plugins suggests applying the command `restorecon` over the file, as it's not matching the one for the folder it's in, and even suggests the exact command to use for restoring the label.

Another plugin suggests generating a custom policy using the following two commands:

```
# ausearch -c 'httpd' --raw | audit2allow -M my-httpd
# semodule -X 300 -i my-httpd.pp
```

However, this kind of recommendation should be taken with knowledge of what is being done, which means that the preceding commands will fix `httpd_t` in terms of getting access to the `home_admin_t` file. We can learn about what would happen by only running the first command, together with the `audit2allow` pipe.

Running `ausearch -c 'httpd' --raw | audit2allow -M my-httpd` creates several files named `my-httpd` in the current folder, one named `my-httpd.te`, and another named `my-httpd.pp`. The second command that we will *not use* installs the modified policy, but please, don't ever do that until you have an understanding of what's going on, as we will see in the following lines.

The interesting file for us now is the `my-httpd.te` one (where *te* means *type enforcement*):

```
module my-httpd 1.0;
require {
        type httpd_t;
        type admin_home_t;
        class file getattr;
}
#============= httpd_t ==============
allow httpd_t admin_home_t:file getattr;
```

From there, we can see that it uses a requirements session for the types involved, and later, the rule itself, which allows `httpd_t` access to `admin_home_t` files for using the `getattr` function, nothing else, nothing more.

As has been said previously, will this fix our issue? It will effectively allow `httpd_t` to obtain access to the `index3.html` file, so there will no longer be any errors, but this comes with a significant cost. From that point, `httpd_t` could also read home directory files without any complaints.

> **Important note**
>
> I don't know how many times this fact needs to be reinforced, but think twice before acting on a system. SELinux is a protection mechanism for increasing the safety of your system; do not disable it, do not blindly accept `audit2allow` created policies without some initial investigation and understanding of what the issue might be and what the proposed resolution does, as it may almost be equivalent to disabling SELinux.

If, at this point, we have installed that module, we can use `semodule` to do the following:

- List `semodule -l`.
- Install `semodule -i $MODULE_NAME`.
- Remove `semodule -r $MODULE_NAME`.

With the preceding commands, we can check or alter the current status for the policy-loaded modules.

Going back to reviewing system logs, we may realize that something is actually failing sometime after it began, but not from the very beginning, so using `ausearch` or passing the full logs to `audit2allow` might not prove helpful; however, we can use the command suggested by `setroubleshootd` to list them:

```
Mar 31 17:06:41 bender setroubleshoot[2924281]: SELinux is
preventing /usr/sbin/httpd from getattr access on the file /
var/www/html/index3.htm. For complete SELinux messages run:
sealert -l 1b4d549b-f566-409f-90eb-7a825471aca8
```

If we execute `sealert -l <ID>`, we will receive the output provided by the different plugins to fix the issue as well as context information similar to what is shown in *Figure 10.11*.

In the case of new software being deployed that has no SELinux support, we can do the following checks the other way around in a test system:

- Set SELinux to `permissive` mode.
- Deploy the software.
- Analyze all the alerts received to see whether anything is unexpected.
- Contact software vendors and initiate a support case with Red Hat to work on a policy.

In case we're getting locked out of our system because SELinux is enforcing and we have badly messed the labels up, for example, by running a bad `chcon` command recursively against our root folder (for example, scripting a context change depending on a variable and that variable being empty), we still have the following ways to get out of trouble:

- Use `setenforce 0` to put SELinux in `permissive` mode.
- Run `touch /.autorelabel`.
- Reboot the host so that at the next boot, SELinux restores the appropriate labels

If we are in a really bad situation and, for example, are unable to use `setenforce 0` or the system cannot even boot or perform relabeling correctly, there is still hope, but some additional steps are required.

When a system is rebooting, we can see the list of installed kernels at the grub prompt and use it to edit the kernel boot parameters.

Using the `selinux=0` parameter, we completely disable SELinux, and this is something we don't want, but we can use `enforcing=0` to accomplish having SELinux enabled, but in `permissive` mode.

Once we have our system booting into `permissive` mode, we can repeat the previous procedure to get back to the previous behavior and continue debugging the situation within the system itself with the prior indications given (checking system logs, and others).

Summary

This chapter has introduced SELinux, how it works, how we can check the processes, files, and ports, and how to fine-tune them either by adding new options or using Booleans. We also covered several initial troubleshooting skills that we should explore further to enhance our knowledge and experience.

SELinux, as we've seen, is a powerful tool for keeping our system secured with an extra layer that protects our system even from unknown issues that might come from defects in the software itself.

We have covered how to find the SELinux context in files and processes, how those are applied via the policy, and how to tune it so that our system is protected and still able to provide the expected service.

Troubleshooting SELinux is a skill that will help us in adapting the software that doesn't come with Red Hat Enterprise Linux to still perform properly.

In the next chapter, we will learn about security profiles with OpenSCAP to continue keeping our system safe.

11
System Security Profiles with OpenSCAP

SCAP stands for **Security Content Automation Protocol**, a standardized way to check, verify, and report vulnerability assessment and policy assessment. Red Hat Enterprise Linux (RHEL) 8 includes the tool **OpenSCAP**, and profiles to audit and manage the security in the systems. This helps ensure the systems you are managing comply with standard security policies such as the **Payment Card Industry Data Security Standard** (**PCI DSS**) or the **Protection Profile for General Purpose Operating Systems**, or **Operating System Protection Profile** (**OSPP**) for short, as well as discovering vulnerabilities.

RHEL 8 includes this tool to review security profiles in order to discover possible attack vectors (misconfigurations or vulnerabilities) and can obtain guidance on how to better harden the system. We will learn how to perform a scan on a system, and discover what needs to be changed to prepare it, in order to ensure it is completely aligned with the regulatory requirements. We will also learn how this tool can be used to improve the security of a system for general use by reviewing it and applying the recommended changes.

To review how to use OpenSCAP, in this chapter we will go through the following topics:

- Getting started with OpenSCAP and discovering system vulnerabilities
- Using OpenSCAP with security profiles for OSPP and PCI DSS

Getting started with OpenSCAP and discovering system vulnerabilities

Let's get started in OpenSCAP in a practical way by first reviewing the `Security Tools` software group, which has some tools that are good to know, and then proceeding to run some scans.

Our initial step will be to get information on `Security Tools`:

```
[root@rhel8 ~]# dnf group info "Security Tools"
Updating Subscription Management repositories.
Last metadata expiration check: 0:37:16 ago on dom 14 mar 2021 16:55:55 CET.

Group: Security Tools
Description: Security tools for integrity and trust verification.
Default Packages:
   scap-security-guide
Optional Packages:
   aide
   hmaccalc
   openscap
   openscap-engine-sce
   openscap-utils
   scap-security-guide-doc
   scap-workbench
   tpm-quote-tools
   tpm-tools
   tpm2-tools
   trousers
   udica
```

This group includes several security tools, such as `aide`, to ensure file integrity in the system; `tpm-tools` to manage the **Trusted Platform Module** (**TPM**) to store encryption keys; and `openscap-utils` to review the security policies in the system.

We can get more information on those tools by using `dnf`. Let's review the one that is more relevant for this chapter, `openscap-utils`:

```
[root@rhel8 ~]# dnf info openscap-utils
Updating Subscription Management repositories.
Last metadata expiration check: 0:03:24 ago on dom 14 mar 2021 17:38:49 CET.
Available Packages
Name         : openscap-utils
Version      : 1.3.3
Release      : 6.el8_3
Architecture : x86_64
Size         : 43 k
Source       : openscap-1.3.3-6.el8_3.src.rpm
Repository   : rhel-8-for-x86_64-appstream-rpms
Summary      : OpenSCAP Utilities
URL          : http://www.open-scap.org/
License      : LGPLv2+
Description  : The openscap-utils package contains command-line tools build on top
             : of OpenSCAP library. Historically, openscap-utils included oscap
             : tool which is now separated to openscap-scanner sub-package.
```

We can see in the output of the previous command what the `openscap-utils` package is about, with a brief description and a link to the main web page, which includes more extensive information.

> **Tip**
>
> It would be useful to run the `dnf info` command for each of the tools mentioned and visit their web pages. This way you will be able to gain a better understanding of the capabilities these tools provide and be able to use them.

Let's now install `openscap-utils`:

```
[root@rhel8 ~]# dnf install openscap-utils -y
Updating Subscription Management repositories.
Last metadata expiration check: 0:04:25 ago on dom 14 mar 2021
17:38:49 CET.
Dependencies resolved.
================================================================
=====================================
Package                    Arch   Version
Repository                             Size
================================================================
=====================================
Installing:
openscap-utils       x86_64 1.3.3-6.el8_3
rhel-8-for-x86_64-appstream-rpms  43 k
Installing dependencies:
GConf2               x86_64 3.2.6-22.el8
rhel-8-for-x86_64-appstream-rpms 1.0 M
[omitted]
  rpmdevtools-8.10-8.el8.noarch
  rust-srpm-macros-5-2.el8.noarch
  zstd-1.4.4-1.el8.x86_64

Complete!
```

Now let's install `scap-security-guide`, which includes the RHEL-specific SCAP profiles:

```
[root@rhel8 ~]# dnf install scap-security-guide -y
Updating Subscription Management repositories.
Last metadata expiration check: 15:06:55 ago on dom 14 mar 2021
17:38:49 CET.
Dependencies resolved.
================================================================
=====================================
Package                       Arch       Version
Repository                                 Size
```

```
=================================================================
=====================================
Installing:
scap-security-guide     noarch     0.1.50-16.el8_3      rhel-8-
for-x86_64-appstream-rpms     7.4 M
Installing dependencies:
xml-common              noarch     0.6.3-50.el8         rhel-8-
for-x86_64-baseos-rpms         39 k

[omitted]

Installed:
  scap-security-guide-0.1.50-16.el8_3.noarch
xml-common-0.6.3-50.el8.noarch

Complete!
```

With this package comes the SCAP Security Guides, including the one related to vulnerabilities for RHEL 8, which is located at `/usr/share/xml/scap/ssg/content/ssg-rhel8-oval.xml`. We can now run an initial scan using all the checks included in the profile. Please note that this will include 2,323 tests and that this will be done as an exercise to learn about possible vulnerabilities and actions to harden a system. So, let's run it:

```
[root@rhel8 ~]# oscap oval eval --report \
vulnerability.html \
/usr/share/xml/scap/ssg/content/ssg-rhel8-oval.xml
Definition oval:ssg-zipl_vsyscall_argument:def:1: false
Definition oval:ssg-zipl_slub_debug_argument:def:1: false
Definition oval:ssg-zipl_page_poison_argument:def:1: false
Definition oval:ssg-zipl_bootmap_is_up_to_date:def:1: false
[omitted]
Definition oval:ssg-accounts_logon_fail_delay:def:1: false
Definition oval:ssg-accounts_have_homedir_login_defs:def:1:
true
Definition oval:ssg-account_unique_name:def:1: true
Definition oval:ssg-account_disable_post_pw_expiration:def:1:
false
Evaluation done.
```

A file called `vulnerability.html` will be generated with the output of the scan. The results will look like this:

OVAL Results Generator Information				
Schema Version	Product Name	Product Version	Date	Time
5.11	cpe:/a:open-scap:oscap	1.3.3	2021-03-21	10:38:37
#×	#✓	#Error	#Unknown	#Other
490	314	355	17	49

OVAL Definition Generator Information				
Schema Version	Product Name	Product Version	Date	Time
5.11	combine_ovals.py from SCAP Security Guide	ssg: [0, 1, 50], python: 3.6.8	2020-10-12	07:40:55
#Definitions	#Tests	#Objects	#States	#Variables
1225 Total 1176 49 0 0 0	2323	2438	1071	776

System Information		
Host Name	rhel8.example.com	
Operating System	Red Hat Enterprise Linux	
Operating System Version	8.3 (Ootpa)	
Architecture	x86_64	
Interfaces	Interface Name	lo
	IP Address	127.0.0.1
	MAC Address	00:00:00:00:00:00
	Interface Name	enp1s0
	IP Address	192.168.122.8
	MAC Address	52:54:00:E6:B4:A4
	Interface Name	lo
	IP Address	::1
	MAC Address	00:00:00:00:00:00
	Interface Name	enp1s0
	IP Address	fe80::9fb8:54a0:643b:ab20
	MAC Address	52:54:00:E6:B4:A4

OVAL System Characteristics Generator Information				
Schema Version	Product Name	Product Version	Date	Time
5.11	cpe:/a:open-scap:oscap	ssg: [0, 1, 50], python: 3.6.8	2021-03-21	10:38:37

OVAL Definition Results

× ✓ Error Unknown Other

ID	Result	Class	Reference ID	Title
oval:ssg-zipl_vsyscall_argument:def:1	false	compliance	[CCE-83381-4], [zipl_vsyscall_argument]	Ensure that BLS-compatible boot loader is configured to run Linux operating system with argument vsyscall=none
oval:ssg-zipl_slub_debug_argument:def:1	false	compliance	[CCE-83371-5], [zipl_slub_debug_argument]	Ensure that BLS-compatible boot loader is configured to run Linux operating system with argument slub_debug=P
oval:ssg-zipl_page_poison_argument:def:1	false	compliance	[CCE-83351-7], [zipl_page_poison_argument]	Ensure that BLS-compatible boot loader is configured to run Linux operating system with argument page_poison=1
oval:ssg-zipl_bootmap_is_up_to_date:def:1	false	compliance	[CCE-83486-1], [zipl_bootmap_is_up_to_date]	Ensure zIPL bootmap is up to date

Figure 11.1 – Initial results of an OpenSCAP test scan

Let's check some of the details of the report. In the top-left corner, we will find **OVAL Results Generator Information**, with the details of the run, and a summary of the results:

OVAL Results Generator Information				
Schema Version	Product Name	Product Version	Date	Time
5.11	cpe:/a:open-scap:oscap	1.3.3	2021-03-21	10:38:37
#×	#✓	#Error	#Unknown	#Other
490	314	355	17	49

Figure 11.2 – OpenSCAP test scan summary

In the top-right corner, we can see **OVAL Definition Generator Information** with a summary of the definitions used for checks:

OVAL Definition Generator Information				
Schema Version	Product Name	Product Version	Date	Time
5.11	combine_ovals.py from SCAP Security Guide	ssg: [0, 1, 50], python: 3.6.8	2020-10-12	07:40:55
#Definitions	#Tests	#Objects	#States	#Variables
1225 Total 1176 49 0 0 0	2323	2438	1071	776

Figure 11.3 – OpenSCAP test scan definitions summary

Right below those tokens of information, we can see a basic summary of the system, useful if we have a long list of scans and we want to assign this scan to the proper system:

System Information		
Host Name	rhel8.example.com	
Operating System	Red Hat Enterprise Linux	
Operating System Version	8.3 (Ootpa)	
Architecture	x86_64	
Interfaces	**Interface Name**	lo
	IP Address	127.0.0.1
	MAC Address	00:00:00:00:00:00
	Interface Name	enp1s0
	IP Address	192.168.122.8
	MAC Address	52:54:00:E6:B4:A4
	Interface Name	lo
	IP Address	::1
	MAC Address	00:00:00:00:00:00
	Interface Name	enp1s0
	IP Address	fe80::9fb8:54a0:643b:ab20
	MAC Address	52:54:00:E6:B4:A4

Figure 11.4 – OpenSCAP test scan system summary

Underneath it, we have information on the generator:

OVAL System Characteristics Generator Information				
Schema Version	**Product Name**	**Product Version**	**Date**	**Time**
5.11	cpe:/a:open-scap:oscap	ssg: [0, 1, 50], python: 3.6.8	2021-03-21	10:38:37

Figure 11.5 – OpenSCAP test scan generator info

And finally, the results of the checks:

OVAL Definition Results

× ✓ Error Unknown Other

ID	Result	Class	Reference ID	Title
oval:ssg-zipl_vsyscall_argument:def:1	false	compliance	[CCE-83381-4], [zipl_vsyscall_argument]	Ensure that BLS-compatible boot loader is configured to run Linux operating system with argument vsyscall=none
oval:ssg-zipl_slub_debug_argument:def:1	false	compliance	[CCE-83371-5], [zipl_slub_debug_argument]	Ensure that BLS-compatible boot loader is configured to run Linux operating system with argument slub_debug=P
oval:ssg-zipl_page_poison_argument:def:1	false	compliance	[CCE-83351-7], [zipl_page_poison_argument]	Ensure that BLS-compatible boot loader is configured to run Linux operating system with argument page_poison=1
oval:ssg-zipl_bootmap_is_up_to_date:def:1	false	compliance	[CCE-83486-1], [zipl_bootmap_is_up_to_date]	Ensure zIPL bootmap is up to date
oval:ssg-zipl_audit_backlog_limit_argument:def:1	false	compliance	[CCE-83341-8], [zipl_audit_backlog_limit_argument]	Ensure that BLS-compatible boot loader is configured to run Linux operating system with argument audit_backlog_limit=8192
oval:ssg-zipl_audit_argument:def:1	false	compliance	[CCE-83321-0], [zipl_audit_argument]	Ensure that BLS-compatible boot loader is configured to run Linux operating system with argument audit=1
oval:ssg-xwindows_runlevel_setting:def:1	false	compliance	[xwindows_runlevel_setting]	Disable X Windows Startup By Setting Default SystemD Target
oval:ssg-usbguard_rules_not_empty_not_missing:def:1	false	compliance	[usbguard_rules_not_empty_not_missing]	Check that file storing USBGuard rules exists and is not empty
oval:ssg-usbguard_allow_hub:def:1	false	compliance	[CCE-82273-4], [usbguard_allow_hub]	Check that USB hubs are allowed by USBGuard rules
oval:ssg-usbguard_allow_hid_and_hub:def:1	false	compliance	[CCE-82368-2], [usbguard_allow_hid_and_hub]	Check that USB human interface devices and hubs are allowed by USBGuard rules

Figure 11.6 – OpenSCAP test scan results

With this test, we have run a vulnerability scan on our system, obtaining a set of results that, depending on the usage of the system, will need to be addressed. In many cases, the warnings received do not apply, so we need to review them carefully. This kind of exercise shall be done carefully on production systems, taking care of having a proper backup and snapshot of the system before proceeding to apply changes. It is recommended to run the hardening in test environments while building the service before moving it to production when possible.

> **Important note**
>
> The *Red Hat Enterprise Linux System Design Guide* for RHEL 8 is a great piece of documentation to get started on system security. It is recommended to read through it in order to extend the knowledge acquired in this chapter. It's available at `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/system_design_guide/index`.

Let's learn more about the basics. For this scan, we have used the Red Hat security advisories **Open Vulnerability Assessment Language** (**OVAL**) feed, as provided by system packages. To check, we have run the OpenSCAP tool to review different security advisories and vulnerabilities as written in OVAL.

OVAL requires that the analyzed resources are in a certain state to consider them correct. It does so in a declarative manner, which means the end state is described and reviewed, not how to get to it.

The Red Hat security team generates Red Hat security advisories to address the different vulnerabilities that the system may be incurring and releases an OVAL definition for each one of them. These are released openly and are available at `https://www.redhat.com/security/data/oval/v2/`.

Now let's take a look at one example found in our report:

- **ID**: `oval:ssg-accounts_logon_fail_delay:def:1`
- **Result**: `false`
- **Class**: `compliance`
- **Reference ID**: `[accounts_logon_fail_delay]`
- **Title**: `Ensure that FAIL_DELAY is Configured in /etc/login.defs`

We can check the manual page for it by running `man login.defs`. In it, we will find the following:

```
FAIL_DELAY (number)
     Delay in seconds before being allowed another attempt after
a
     login failure.
```

This is the value to establish how long a user will have to wait after a failed login attempt. It is intended to avoid brute force attacks on accounts in the system. We may take, for example, two approaches to fix it:

- Add the `FAIL_DELAY` variable and value to `login.defs`.
- Enforce access to the system by only allowing login access using SSH keys and not passwords.

Or even better, do both (security in depth). We could continue to review each of the entries in the list and understand each of them to complete the hardening of the system to avoid as much exposure as possible. This is a task that is usually run in coordination with the security teams and is continuously being reviewed.

Now that we have run our first vulnerability scan, let's see how we can do it for compliance in the next section.

Using OpenSCAP with security profiles for OSPP and PCI DSS

There are several security profiles used for compliance in the industry. Two of the most common, which we will review here, are the **Operating System Protection Profile** (**OSPP**) and the PCI DSS.

The OSPP standard is heavily used in the public sector, serving general-purpose systems and also as the baseline for other more restrictive environments (that is, defense accredited systems).

PCI DSS is one of the most widely used standards in the finance sector, which also applies to other sectors that want to provide online payments using credit cards.

RHEL 8 provides references to verify those profiles using OpenSCAP tools. Let's move to the `/usr/share/xml/scap/ssg/content/` directory, where they reside, and take a look:

```
[root@rhel8 ~]# cd   /usr/share/xml/scap/ssg/content/
[root@rhel8 content]# ls *rhel8*
ssg-rhel8-cpe-dictionary.xml
ssg-rhel8-ds-1.2.xml
ssg-rhel8-ocil.xml
ssg-rhel8-xccdf.xml
ssg-rhel8-cpe-oval.xml
ssg-rhel8-ds.xml
ssg-rhel8-oval.xml
```

As you can see, we have different types of descriptions that can be used with OpenSCAP. We already know OVAL. Let's check the most important ones:

- **Extensible Configuration Checklist Description Format (XCCDF)**: XCCDF is used to build security checklists. It's very common for compliance testing and scoring.
- **Common Platform Enumeration (CPE)**: CPE helps identify systems by assigning unique identifier names. This way, it can correlate tests and names.
- **Open Checklist Interactive Language (OCIL)**: OCIL is part of the SCAP standard. It is a way to aggregate other checks from different datastores.
- **DataStream (DS)**: DS is a format that puts together several components into a single file. It is used to distribute profiles easily.

> **Tip**
> More information on the different security descriptions and components can be found on the OpenSCAP web page by checking the components URL: `https://www.open-scap.org/features/scap-components/`.

In this case, we will use the `ssg-rhel8-ds.xml` file. Let's check the information related to it:

```
[root@rhel8 content]# oscap info ssg-rhel8-ds.xml
Document type: Source Data Stream
[omitted]
```

```
Profiles:
Title: CIS Red Hat Enterprise Linux 8 Benchmark
Id: xccdf_org.ssgproject.content_profile_cis
Title: Unclassified Information in Non-federal Information
Systems and Organizations (NIST 800-171)
Id: xccdf_org.ssgproject.content_profile_cui
Title: Australian Cyber Security Centre (ACSC) Essential Eight
Id: xccdf_org.ssgproject.content_profile_e8
Title: Health Insurance Portability and Accountability Act
(HIPAA)
Id: xccdf_org.ssgproject.content_profile_hipaa
Title: Protection Profile for General Purpose Operating Systems
Id: xccdf_org.ssgproject.content_profile_ospp
Title: PCI-DSS v3.2.1 Control Baseline Red Hat Enterprise Linux
8
Id: xccdf_org.ssgproject.content_profile_pci-dss
Title: [DRAFT] DISA STIG for Red Hat Enterprise Linux 8
Id: xccdf_org.ssgproject.content_profile_stig
Referenced check files: ssg-rhel8-oval.xml
system: http://oval.mitre.org/XMLSchema/oval-definitions-5
ssg-rhel8-ocil.xml
system: http://scap.nist.gov/schema/ocil/2
security-data-oval-com.redhat.rhsa-RHEL8.xml
system: http://oval.mitre.org/XMLSchema/oval-definitions-5
Checks:
Ref-Id: scap_org.open-scap_cref_ssg-rhel8-oval.xml
Ref-Id: scap_org.open-scap_cref_ssg-rhel8-ocil.xml
Ref-Id: scap_org.open-scap_cref_ssg-rhel8-cpe-oval.xml
Ref-Id: scap_org.open-scap_cref_security-data-oval-com.redhat.
rhsa-RHEL8.xml
Dictionaries:
Ref-Id: scap_org.open-scap_cref_ssg-rhel8-cpe-dictionary.xml
```

As you can see, it includes the profiles for both OSPP and PCI DSS for RHEL 8. Let's give them a try.

Scanning for OSPP compliance

We can use the `--profile` option for `oscap` to get information specific to the **OSPP** profile:

```
[root@rhel8 content]# oscap info --profile \
ospp ssg-rhel8-ds.xml
Document type: Source Data Stream
Imported: 2020-10-12T09:41:22

Stream: scap_org.open-scap_datastream_from_xccdf_ssg-rhel8-
xccdf-1.2.xml
Generated: (null)
Version: 1.3
WARNING: Datastream component 'scap_org.open-scap_cref_
security-data-oval-com.redhat.rhsa-RHEL8.xml' points out to the
remote 'https://www.redhat.com/security/data/oval/com.redhat.
rhsa-RHEL8.xml'. Use '--fetch-remote-resources' option to
download it.
WARNING: Skipping 'https://www.redhat.com/security/data/
oval/com.redhat.rhsa-RHEL8.xml' file which is referenced from
datastream
Profile
Title: Protection Profile for General Purpose Operating Systems
Id: xccdf_org.ssgproject.content_profile_ospp

Description: This profile reflects mandatory configuration
controls identified in the NIAP Configuration Annex to the
Protection Profile for General Purpose Operating Systems
(Protection Profile Version 4.2.1).  This configuration profile
is consistent with CNSSI-1253, which requires U.S. National
Security Systems to adhere to certain configuration parameters.
Accordingly, this configuration profile is suitable for use in
U.S. National Security Systems.
```

We can see in the information that the OSPP profile comes described as `xccdf`. We can now run `oscap`, indicating that we want to use that format with the `xcddf` option and that the action we want to take is to evaluate the system with `eval`. The command will be as follows:

```
[root@rhel8 content]# oscap xccdf eval \
--report ospp-report.html --profile ospp ssg-rhel8-ds.xml
[omitted]
Title   Set Password Maximum Consecutive Repeating Characters
Rule    xccdf_org.ssgproject.content_rule_accounts_password_
pam_maxrepeat
Ident   CCE-82066-2
Result  fail

Title   Ensure PAM Enforces Password Requirements - Maximum
Consecutive Repeating Characters from Same Character Class
Rule    xccdf_org.ssgproject.content_rule_accounts_password_
pam_maxclassrepeat
Ident   CCE-81034-1
Result  fail
[omitted]
Title   Disable Kerberos by removing host keytab
Rule    xccdf_org.ssgproject.content_rule_kerberos_disable_no_
keytab
Ident   CCE-82175-1
Result  pass
```

We will obtain the `ospp-report.html` file with the full report on the OSPP rule results:

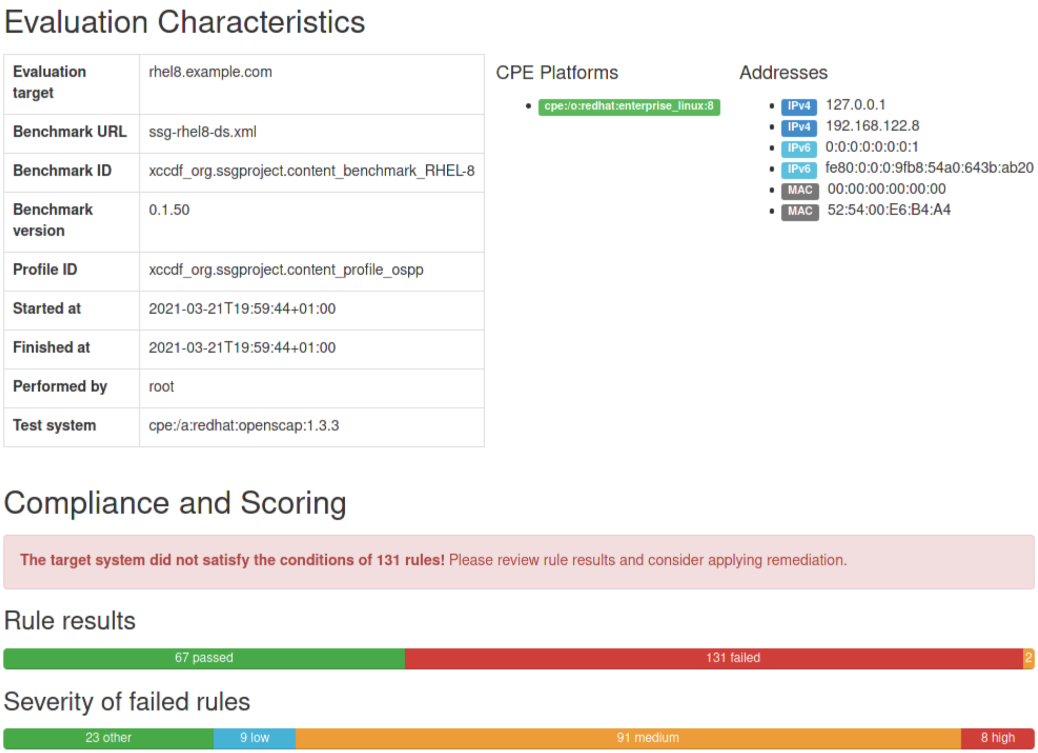


Figure 11.7 – OpenSCAP OSPP scan results

It will show the points that require modification to be compliant with the profile:

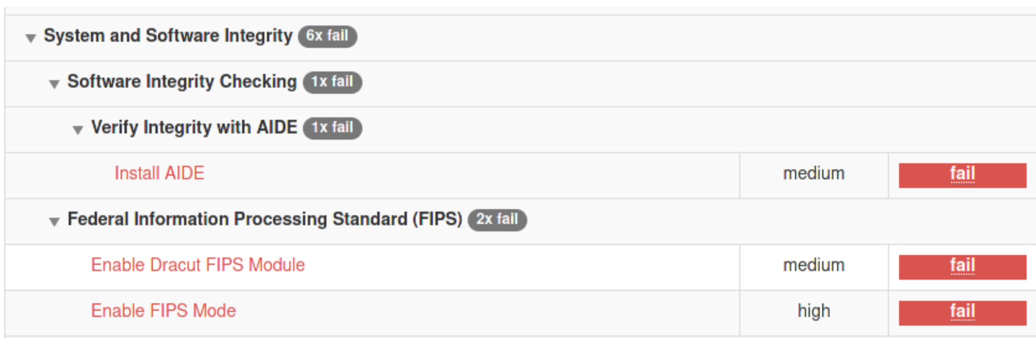


Figure 11.8 – OpenSCAP OSPP scan results, detail rules that require action

We can now follow, step by step, the recommendations and fix them in order to be fully OSPP compliant. Also, we can use this scan to harden systems that, even when they do not need to be OSPP compliant, will be in an exposed network such as a DMZ, and we want to have them hardened.

> **Important note**
>
> Red Hat provides a way to automatically apply all those changes. It is based on the automation tool **Ansible**. It is provided in the form of a playbook, a set of descriptions for Ansible that will apply all the changes required to the system. The playbook for OSPP is located at `/usr/share/scap-security-guide/ansible/rhel8-playbook-ospp.yml`.

Now that we have reviewed the system for OSPP compliance, let's move to the next target, PCI DSS compliance.

Scanning for PCI DSS compliance

We can follow the same procedure as before, also using the `--profile` option for `oscap` to get information specific to the PCI DSS profile:

```
[root@rhel8 content]# oscap info --profile pci-dss \
ssg-rhel8-ds.xml
Document type: Source Data Stream
Imported: 2020-10-12T09:41:22

Stream: scap_org.open-scap_datastream_from_xccdf_ssg-rhel8-
xccdf-1.2.xml
Generated: (null)
Version: 1.3
WARNING: Datastream component 'scap_org.open-scap_cref_
security-data-oval-com.redhat.rhsa-RHEL8.xml' points out to the
remote 'https://www.redhat.com/security/data/oval/com.redhat.
rhsa-RHEL8.xml'. Use '--fetch-remote-resources' option to
download it.
WARNING: Skipping 'https://www.redhat.com/security/data/
oval/com.redhat.rhsa-RHEL8.xml' file which is referenced from
datastream
Profile
Title: PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise
Linux 8
```

```
Id: xccdf_org.ssgproject.content_profile_pci-dss

Description: Ensures PCI-DSS v3.2.1 security configuration
settings are applied.
```

We can run `oscap` with the same options as in the previous section but specifying `pci-dss` as the profile. It will generate the proper report:

```
[root@rhel8 content]# oscap xccdf eval –report \
pci-dss-report.html --profile pci-dss ssg-rhel8-ds.xml
WARNING: Datastream component 'scap_org.open-scap_cref_
security-data-oval-com.redhat.rhsa-RHEL8.xml' points out to the
remote 'https://www.redhat.com/security/data/oval/com.redhat.
rhsa-RHEL8.xml'. Use '--fetch-remote-resources' option to
download it.
WARNING: Skipping 'https://www.redhat.com/security/data/
oval/com.redhat.rhsa-RHEL8.xml' file which is referenced from
datastream
WARNING: Skipping ./security-data-oval-com.redhat.rhsa-RHEL8.
xml file which is referenced from XCCDF content
Title   Ensure PAM Displays Last Logon/Access Notification
Rule    xccdf_org.ssgproject.content_rule_display_login_
attempts
Ident   CCE-80788-3
Result  pass
[omitted]
Title   Specify Additional Remote NTP Servers
Rule    xccdf_org.ssgproject.content_rule_chronyd_or_ntpd_
specify_multiple_servers
Ident   CCE-80764-4
Result  fail

[root@rhel8 content]# ls -l pci-dss-report.html
-rw-r--r--. 1 root root 3313684 mar 21 20:16 pci-dss-report.
html
```

We can start reviewing the items in the report and start fixing them.

> **Important note**
> As in the previous section, Red Hat also provides a way to automatically apply all those changes with Ansible. The playbook for PCI DSS is located at `/usr/share/scap-security-guide/ansible/rhel8-playbook-pci-dss.yml`.

We have seen that changing from one profile to a different one with OpenSCAP is very easy and we can scan for as many profiles as available.

Summary

By learning the basics of **OpenSCAP**, we are ready to review and harden systems to make them compliant with the regulations we need them to run under.

Now if you are requested to comply with any regulatory requisitions, you can find the right SCAP profile for it (or build it if it doesn't exist) and ensure that your systems are fully compliant.

Also, even when no regulatory requirements apply, the use of OpenSCAP can help you find vulnerabilities in the system, or apply a more secure (and restrictive) configuration to your systems in order to reduce the risks.

There are ways to extend our knowledge and skills by learning Ansible and being able to automatically apply changes to our systems in a way that is easy to scale, as well as Red Hat Satellite, which can help run SCAP scans to the whole IT base we are managing even when we could be talking about thousands of systems.

Now that our security skills are improving and being consolidated, let's dive deeper into more low-level topics such as local storage and filesystems, as described in the next chapter.

Section 3: Resource Administration – Storage, Boot Process, Tuning, and Containers

Managing the resources of machines running RHEL is fundamental in terms of a performant, efficient IT environment. Understanding the storage, tuning the performance (including the configuration required to make it permanent in the boot process), and then using containers to isolate processes and assign resources more efficiently are areas into which a system administrator will surely step in their daily work.

The following chapters are included in this section:

- *Chapter 12, Managing Local Storage and Filesystems*
- *Chapter 13, Flexible Storage Management with LVM*
- *Chapter 14, Advanced Storage Management with Stratis and VDO*
- *Chapter 15, Understanding the Boot Process*
- *Chapter 16, Kernel Tuning and Managing Performance Profiles with tuned*
- *Chapter 17, Managing Containers with Podman, Buildah, and Skopeo*

12
Managing Local Storage and Filesystems

In the previous chapters, we have learned about security and system administration. In this chapter, we will focus on the administration of resources—specifically, storage administration.

Storage administration is an important part of keeping a system running: the system logs can eat available space, new applications might require additional storage set up for them (even on separate disks to improve performance), and such issues may require our action in order to solve them.

In this chapter, we will learn about the following topics:

- Partitioning disks (**Master Boot Record** (**MBR**) and **Globally Unique Identifier** (**GUID**) **Partition Table** (**GPT**) disks)
- Formatting and mounting filesystems
- Setting default mounts and options in `fstab`
- Using network filesystems with **Network File System** (**NFS**)

This will provide us basic knowledge to build upon our storage administration skills to keep our systems running.

Let's get hands-on!

Technical requirements

You can continue the practice using the **virtual machine** (**VM**) created at the beginning of this book in *Chapter 1*, *Installing RHEL8*. Any additional packages required for this chapter will be indicated alongside the text. You will also need partitioning disks (MBR and GPT disks).

Let's start with a definition

A partition is a logical division of a storage device and it's used to logically separate the available storage into smaller pieces.

Now, let's move on to learning a bit about the origins of storage to better understand it.

A bit of history

Storage is also related to a system's ability to use it, so let's explain a bit about the history of **personal computers** (**PCs**), the software that allows them to boot (**Basic Input/Output System** (**BIOS**)), and how that influenced storage administration.

It might sound a bit strange, but initial storage needs were just a small amount of **kilobytes** (**KB**), and for the first hard drives in PCs, storage was just a few **megabytes** (**MB**).

PCs also come with one feature and limitation: PCs were compatible, which means that subsequent models had compatibility with the initial **International Business Machines** (**IBM**) PC design.

Traditional disk partitioning uses a space at the beginning of disks after the MBR that allows four partition registers (start, end, size, partition type, active flag), called **primary** partitions.

When the PC is booting, BIOS will check the partition table of the disk by running a small program in the MBR, and then it loads the boot area of the active partition and executes it to get the operating system booting.

The IBM PC that contained a **Disk Operating System** (**DOS**) and compatibles (MS-DOS, DR-DOS, FreeDOS, and others) also used a filesystem named a **File Allocation Table** (**FAT**). The FAT contained several structures based on its evolution, indicated as the cluster addressing size (as well of some other features).

With a limit in the number of clusters, having bigger disks meant having bigger blocks, so if a file was using just a limited amount of space, the remaining ones couldn't be used by other files. Thus, it became more or less normal to break bigger hard drives into smaller logical partitions so that small files would not eat up the available space because of limits.

Think about this as an agenda with a maximum number of entries, similar to a speed dial in your phone: if you have only nine slots for the speed dial, a short number such as calling voicemail will still count as having stored a big international number as both still use one slot.

Some of those limitations became reduced by subsequent versions of the FAT sizing, which in parallel increased the maximum supported disk size.

Of course, other operating systems introduced their own filesystems, but using this same partitioning schema.

Later, a new partition type was created: the **extended partition**, which used one of the four available **primary partition** slots and allowed extra partitions to be defined inside it, enabling us to create logical disks to be assigned as needed.

Additionally, having several primary partitions also allowed the installation in the same computer of different operating systems with their own dedicated space that were completely independent of other operating systems.

So... partitions allowed computers to have different operating systems, have a better usage of the available storage, or even logically sorting the data by keeping it on different areas, such as keeping operating system space separate from user data so that a user filling the available space would not affect the computer's operation.

As we said, many of those designs came with the compatibility restriction of the original IBM PC, so when new computers using the **Extensible Firmware Interface** (**EFI**) appeared to overcome the limitations of traditional BIOS, a new partition table format called **GPT** arrived.

Systems using GPT make use of 32-bit and 64-bit support versus the 16-bit support used by BIOS (inherited from IBM PC compatibility), so bigger addressing can be used for the disks, as well as extra features such as extended controllers loading.

Now, let's learn about disk partitioning in the next section.

Partitioning disks (MBR and GPT disks)

As mentioned, using disk partitions allows us to more efficiently use the space available in our computers and servers.

Let's dig into disk partitioning by first identifying the disk to act on.

> **Important note**
>
> Once we have learned about what caused disks to be partitioned and the limitations of this, we should follow one schema or another based on our system specifications, but bear in mind that EFI requires GPT and BIOS requires MBR, so a system supporting UEFI, but having a disk partitioned with MBR, will boot the system into BIOS-compatible mode.

Linux uses different notation for the disks based on the way those are connected to the system, so—for example—you can see disks as `hda` or `sda` or `mmbclk0` depending on that connection being used. Traditionally, disks connected using the **Integrated Drive Electronics** (**IDE**) interface used to have disks named `hda`, `hdb`, and so on, while disks using the **Small Computer System Interface** (**SCSI**) used to have disks named `sda`, `sdb`, and so on.

We can list the available devices with `fdisk -l` or `lsblk -fp`, as we can see in the following screenshot:

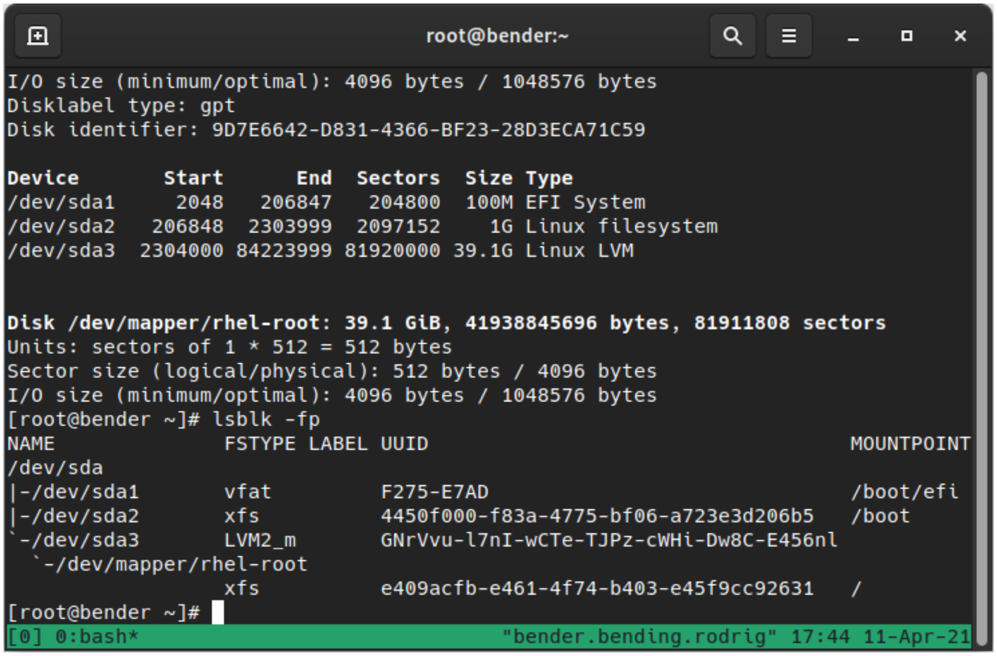


Figure 12.1 – lsblk-fp and fdisk –l output

As we can see, our disk named `/dev/sda` has three partitions: `sda1`, `sda2`, and `sda3`, with `sda3` being a `LVM` volume group that has a volume named `/dev/mapper/rhel-root`.

To demonstrate disk partitioning in a safe way and to make it easier for readers using a VM for testing, we will create a fake **virtual hard drive** (**VHD**) for testing. In doing so, we will use the `truncate` utility that comes with the `coreutil` package and the `losetup` utility that comes with the `util-linux` package.

In order to create a VHD, we will execute the following sequence of commands as they appear in *Figure 12.2*:

1. `truncate -s 20G myharddrive.hdd`

> **Note**
>
> This command creates a 20 **gigabytes** (**GB**)-sized file but this will be an empty file, which means that the file is not really using 20 GB on our disk, just showing that size. Unless we use it, it will not consume more disk space (this is called a **sparse file**).

2. `losetup -f`, which will find the next available device
3. `losetup /dev/loop0 myharddrive.hdd`, which will associate `loop0` with the file created
4. `lsblk -fp`, to validate the newly looped disk
5. `fdisk -l /dev/loop0`, to list the available space in the new disk

The following screenshot shows the output of the preceding sequential commands:

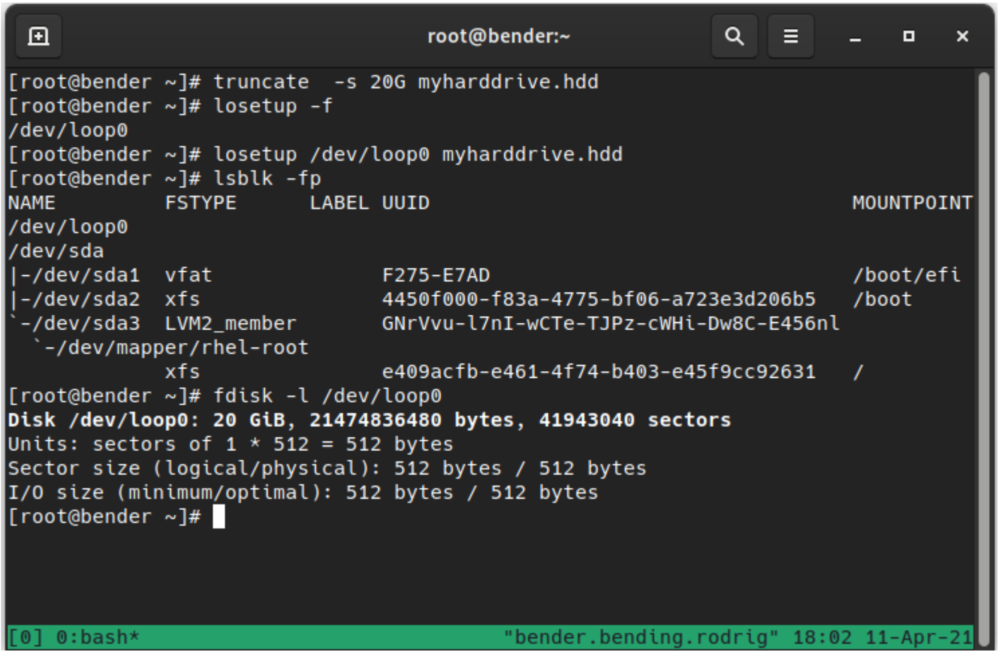


Figure 12.2 – Execution of the indicated commands for creating a fake hard drive

The `losetup -f` command finds the next available loopback device, which is a device used for looping back accesses to a backing file. This is often used for mounting ISO files locally, for example.

With the third command, we use the previously available loopback device to set up a loop connection between device `loop0` device and the file we created with the first command.

As we can see, in the remaining commands, the device now appears when running the same commands, we executed in *Figure 12.1*, showing that we have a disk of 20 GB available.

> **Important note**
>
> Partitioning operations on disks can be dangerous and can render a system unusable and in need of restoration or reinstallation. To reduce that chance, the examples in this chapter will use the `/dev/loop0` fake created disk and only interact with this. Pay attention when performing this over real volumes, disks, and so on.

Let's start creating partitions by executing `fdisk /dev/loop0` on our newly created device, as seen in the following screenshot:

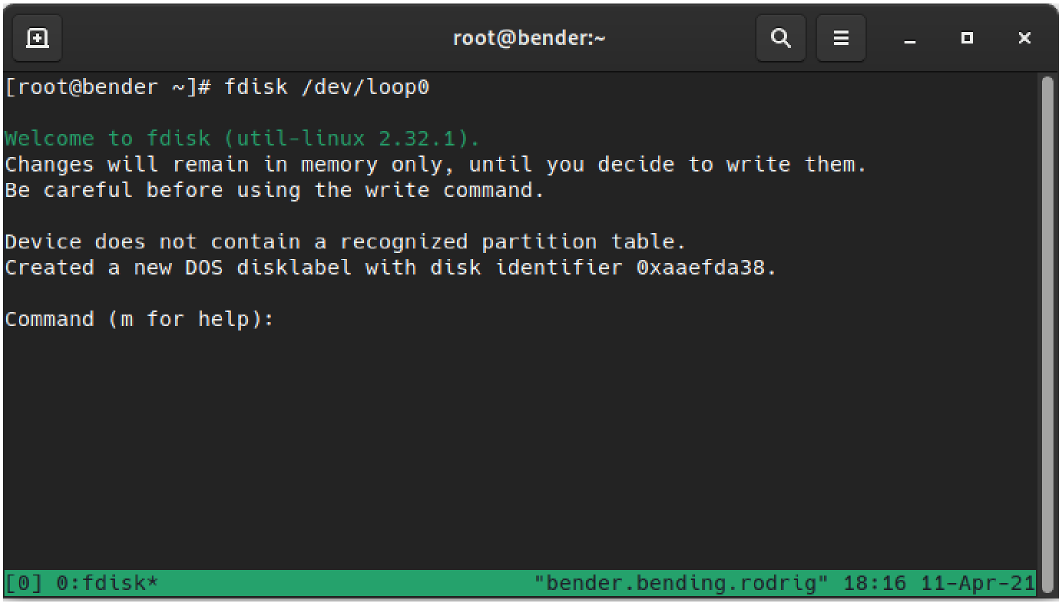


Figure 12.3 – fdisk execution over /dev/loop0

As we can see in *Figure 12.3*, the disk doesn't contain a recognized partition table, so a new DOS partition disk label is created, but the changes only remain in memory until written back to disk.

Inside the `fdisk` command, we can use several options to create a partition. The first one we should be aware of is `m`, as indicated in *Figure 12.3*, which shows the help functionality and available commands.

The first thing to take into consideration is our previous explanation about UEFI, BIOS, and so on. By default, `fdisk` is creating a DOS partition, but as we can see inside the manual (`m`), we can create a GPT one by running the `g` command inside `fdisk`.

One important command to remember is `p`, which prints the current disk layout and partition, as defined in the next screenshot:

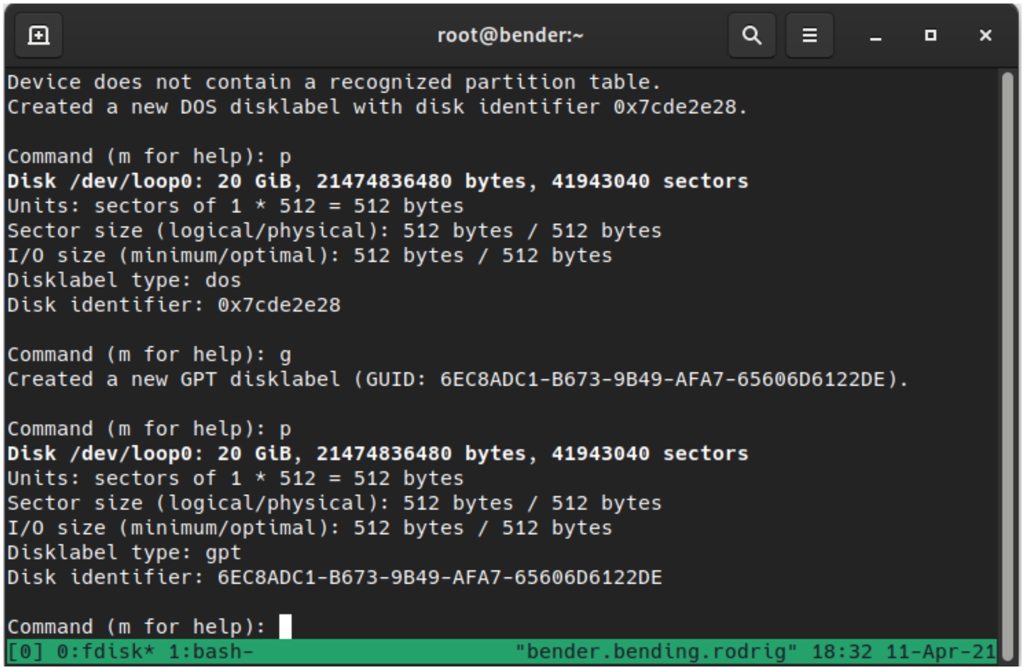


Figure 12.4 – fdisk creating a new partition table

As we can see, the initial `disklabel` type was `dos` and now is `gpt`, compatible with EFI/UEFI.

Let's review some of the basic commands we can use, as follows:

- `n`: Creates a new partition
- `d`: Deletes a partition
- `m`: Shows manual page (help)
- `p`: Prints current layout
- `x`: Enters advanced mode (extra functionality intended for experts)
- `q`: Quits without saving

- `w`: Writes changes to disk and exits
- `g`: Creates new GPT disk label
- `o`: Creates a DOS disk label
- `a`: In DOS mode, sets the bootable flag to one of the primary partitions

What will be the sequence for creating a new traditional disk partition layout with a bootable partition for the operating system and another one for the user data with half the disk size each?

This will be the sequence of commands (these are also shown in *Figure 12.5*):

1. `o` and press *Enter* to create a new DOS disk label
2. `n` and press *Enter* to create a new partition
3. Press *Enter* to accept a primary partition type
4. Press *Enter* to confirm use of the first partition (`1`)
5. Press *Enter* to accept initial sector
6. `+10G` and press *Enter* to indicate 10 GB in size from first sector
7. `n` and press *Enter* to create a second new partition
8. Press *Enter* to accept it as a primary partition type
9. Press *Enter* to accept the partition number (`2`)
10. Press *Enter* to accept the first sector as default proposed by fdisk
11. Press *Enter* to accept the end sector as default proposed by fdisk
12. `a` and press *Enter* to mark a partition as bootable
13. `1` and press *Enter* to mark the first partition

As you can see, most of the options accept the defaults; the only change was to specify a partition size of `+10G`, meaning it should be 10 GB (the disk was 20 GB), and then start with the second partition with the new `n` command, now not specifying the size as we want to use all the remaining ones. The last step is to mark the first partition as active for booting.

Of course, remember what we said before: unless we execute the w command, the changes are not written to disk, and we can use p to review them, as shown in the following screenshot:

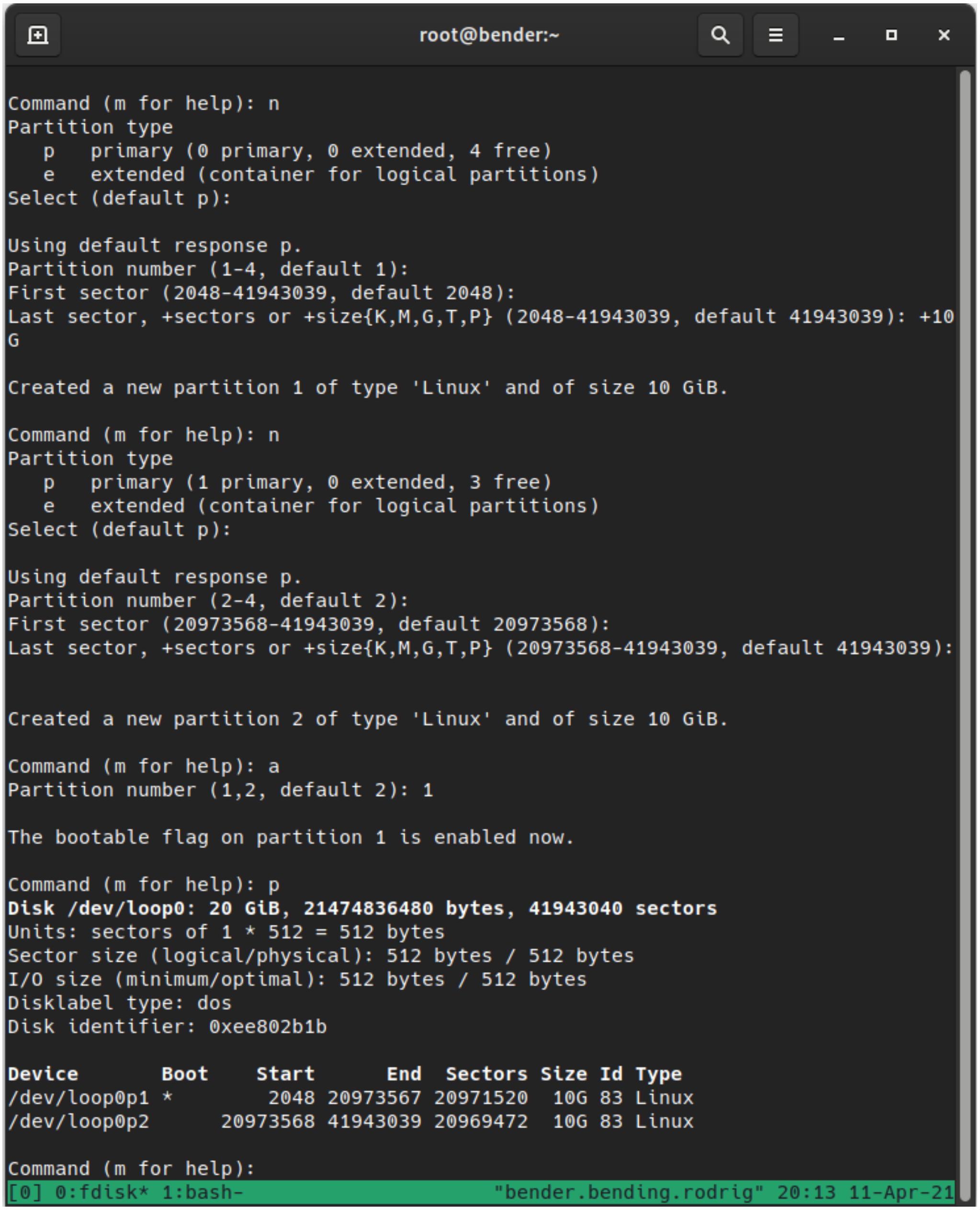


Figure 12.5 – Disk partition layout creation and verification before writing it back to disk

To conclude this section, let's write the changes to disk with the `w` command and let's move on to discuss filesystems in the next section. Before that, however, let's execute `partprobe /dev/loop0` to make the kernel update its internal view on the disk and find the two new partitions. Without this, the `/dev/loop0p1` and `/dev/loop0p2` special files might not be created and will not be usable.

Note that some partition modifications will not be updated even after a `partprobe` execution and might require the system to be rebooted. This is, for example, happening in disks that have partitions in use, such as the one holding the root filesystem in our computer.

Formatting and mounting filesystems

In the previous section, we learned how to logically divide our disk, but that disk is still not usable for storing data. In order to enable this to do so, we need to define a **filesystem** on it as the first step to make it available to our system.

A filesystem is a logical structure that defines how files, folders, and more are stored and provides, based on each type, a different set of features.

The number and types of filesystems supported depends on the operating system version, as during its evolution, new filesystems might be added, removed, and so on.

> **Tip**
>
> Remember that **Red Hat Enterprise Linux** (**RHEL**) focuses on stability, so there are strict controls about which features are added or phased out for newer releases, but not within the current release. You can read more about this at `https://access.redhat.com/articles/rhel8-abi-compatibility`.

In RHEL 8, the default filesystem is the **eXtended File System** (**XFS**), but you can see a list of available ones at the RHEL documentation found at `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/system_design_guide/overview-of-available-file-systems_system-design-guide`, and of course, others such as **Fourth Extended Filesystem** (**EXT4**) can be used.

The choice of the filesystem depends on several factors such as usage intention, type of files that are going to be used, and more, as different filesystems might have performance implications.

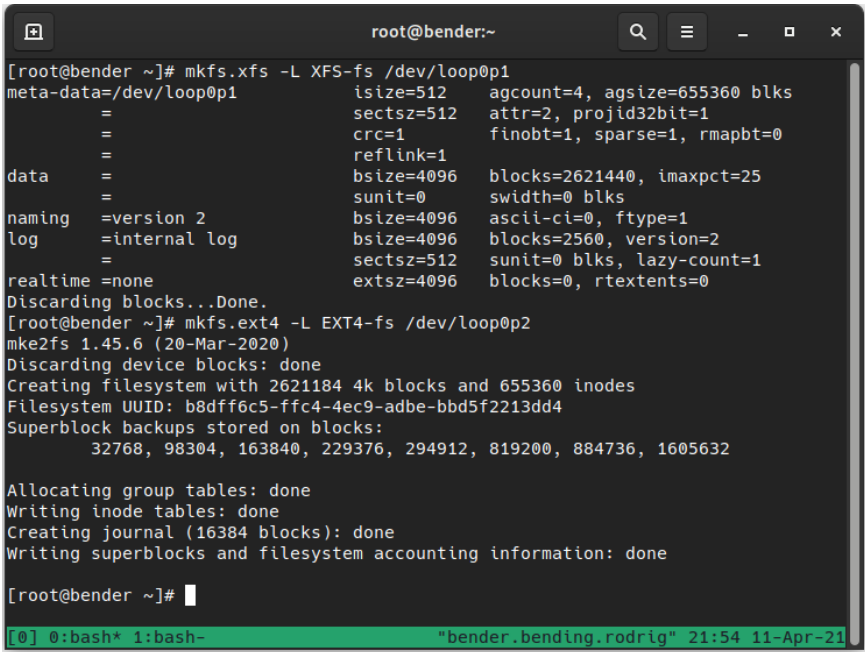
For example, both EXT4 and XFS are journaled filesystems that provide more protection against power failures, but the maximum filesystem differs in terms of other aspects such as the likeliness of becoming fragmented, and so on.

Before choosing a filesystem, it is a good practice to get an idea of the kind of files being deployed and their usage pattern, as choosing the wrong one might affect system performance.

As we defined in the previous section two partitions on our VHD, we can try creating both XFS and EXT4 filesystems. Again, however, be very careful when performing operations, as filesystem creation is a destructive operation that writes new structures back to the disk, and when operating as root user of the system, which is required, selecting the wrong one can destroy within seconds the available data we had on our system.

> **Important note**
>
> Remember to check the man page for the commands being used in order to get familiar with the different recommendations and options available for each one.

Let's then use the two partitions we created to test with two filesystems, XFS and EXT4, by using the `mkfs.xfs` and `mkfs.ext4` commands against each one of the devices respectively, as follows:



```
root@bender:~
[root@bender ~]# mkfs.xfs -L XFS-fs /dev/loop0p1
meta-data=/dev/loop0p1           isize=512    agcount=4, agsize=655360 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=2621440, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
Discarding blocks...Done.
[root@bender ~]# mkfs.ext4 -L EXT4-fs /dev/loop0p2
mke2fs 1.45.6 (20-Mar-2020)
Discarding device blocks: done
Creating filesystem with 2621184 4k blocks and 655360 inodes
Filesystem UUID: b8dff6c5-ffc4-4ec9-adbe-bbd5f2213dd4
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

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

[root@bender ~]#
[0] 0:bash* 1:bash-                    "bender.bending.rodrig" 21:54 11-Apr-21
```

Figure 12.6 – Filesystem creation on the VHD created

Note that we have specified the different loop device partition and we also specified one `-L` parameter for each command. We will look at this again later.

Now that the filesystem has been created, we can run `lsblk -fp` to verify this, and we can see both devices, now indicating the filesystem is in use as well as `LABEL` and `UUID` values (the ones shown when we created the filesystem with `mkfs`), as we can see in the following screenshot:

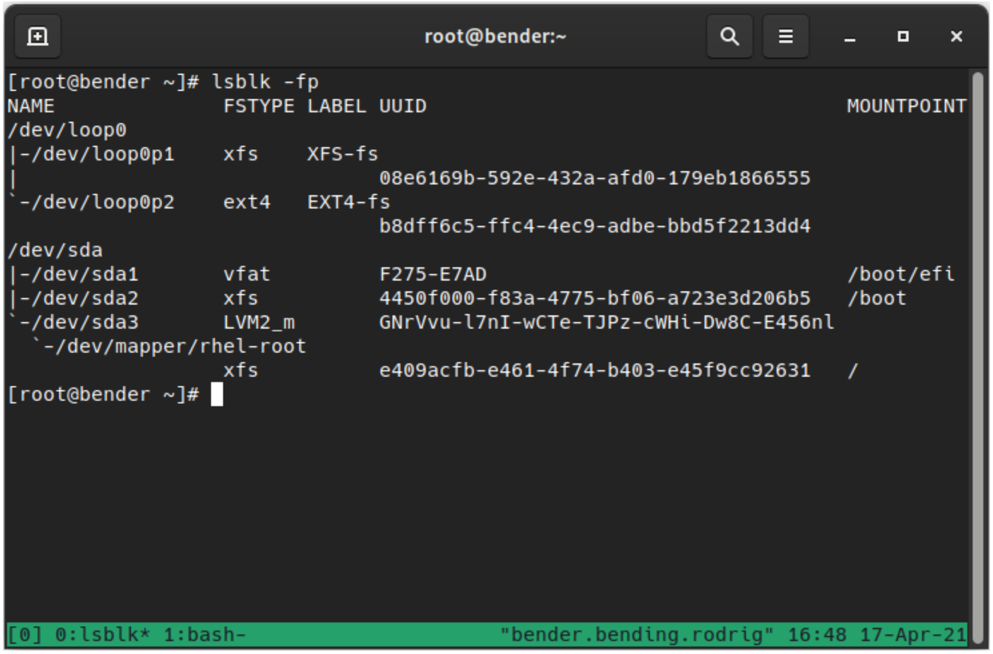


Figure 12.7 – Output of lsblk –fp after creating the filesystems

From the preceding output, it's important to pay attention to the `UUID` and `LABEL` values (if you remember, the value listed is the one we specified in the `mkfs` command with the `–L` option), as we will be using them later in this chapter.

Now that the filesystems have been created, in order to use them we need to mount them, which means making the filesystem available at a path in our system so that every time we store inside that path, we will be using that device.

Mounting a filesystem can be done in several ways, but the simplest way is to use autodetection and just specify the device to mount and the local path to mount it at, but more complex ones that allow several options to be defined can be found when checking the `man mount` help page.

For mounting our two created filesystems, we will create two folders and then proceed to mounting each device by executing the following commands:

1. `cd`
2. `mkdir first second`
3. `mount /dev/loop0p1 first/`
4. `mount /dev/loop0p2 second/`

At this point, the two filesystems will be available in our home folder (root user) in the subfolders named `first` and `second`.

The kernel has automatically found which filesystem is in use for each device and has loaded it via the appropriate controller and this worked, but sometimes we might want to define specific options— for example, forcing the filesystem type, which was used in the past when `ext2` and `ext3` were the common filesystems to enable or disable journaling, or, for example, to disable the built-in features that update the file or directory access time to reduce the disk I/O and increase performance.

All options specified on the command line, or filesystems mounted, will not be available once the system is rebooted as those are only runtime changes. Let's move on to the next section to learn how to define default options and filesystem mounts when the system is being started.

Setting default mounts and options in fstab

In the previous section, we introduced how disks and partitions can be mounted so that our services and users can make use of them. In this section, we will learn how to make those filesystems available in a persistent way.

The `/etc/fstab` file contains the filesystem definitions for our system and, of course, it has a dedicated manual page that can be checked with `man fstab` containing useful information about the formatting, fields, ordering, and so on that must be taken into consideration, as this file is critical for the smooth operation of the system.

The file format is defined by several fields separated by tabs or spaces, with lines starting with a # considered as comments.

For example, we will use this line to look at each field description:

```
LABEL=/ / xfs defaults 0 0
```

The first field is the device definition, which can be a special block device, a remote filesystem, or—as we can see—a selector made by `LABEL`, `UUID`, or, for **GPT** systems, also a `PARTUUID` or `PARTLABEL`. The `man` page for `mount`, `blkid`, and `lsblk` provides more information about device identifiers.

The second field is the mount point for the filesystem, which is where to make the contents of that filesystem available based on our system directory hierarchy. Some special devices/partitions such as swap areas have this defined as `none`, as effectively the contents are not made available via the filesystem.

The third field is the filesystem type as supported by the `mount` command or `swap`, for swap partitions.

The fourth field is the mount options as supported by `mount` or `swapon` commands (check their `man` pages for more details), being at its default setting an alias for most common options (read/write, allow devices, allow execution, automount on boot, async access, and so on). Other common options might be `noauto`, which defines the filesystem but doesn't mount at boot (often used with removable devices), `user`, which allows users to mount and unmount it, and `_netdev`, which defines remote paths that require networking to be up before attempting the mount.

The fifth field is used by `dump` to determine which filesystems should be used—its value defaults to `0`.

The sixth field is used by `fsck` to determine the order for filesystems to be checked on boot. The root filesystem should have a value of 1 and the others should have a value of 2 (the default is 0, not `fsck`). Checks are performed in parallel to speed up the booting process. Note that with filesystems that have a journal, the filesystem itself can perform a fast validation instead of a full one.

In the following screenshot, let's see how it looks in our system with the output of `cat /etc/fstab`:

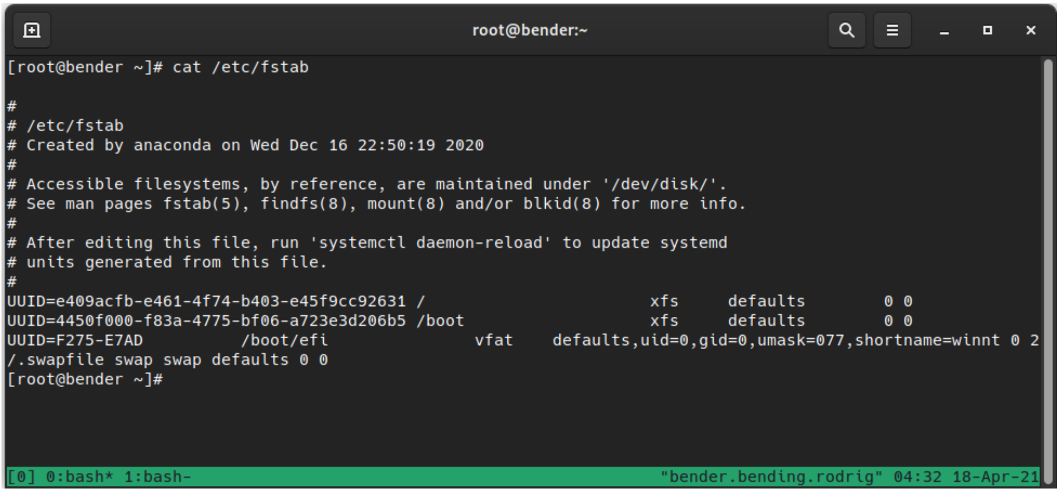


Figure 12.8 – fstab example from our system

Why should we use `UUID` or `LABEL` instead of devices such as `/dev/sda1`?

Disk ordering might change when a system is booting, as some kernels might introduce differences in the devices in how those are accessed, and so on, causing changes in the enumeration of the devices; this happens not only for removable devices such as **Universal Serial Bus** (**USB**) ones, but also to internal devices such as network interfaces or hard drives.

When, instead of specifying the devices, we use `UUID` or `LABEL`, even in the event of a device reordering, the system will still be able to find the right device to use and boot from it. This was especially important when systems used to have **IDE** and **Serial Advanced Technology Attachment** (**SATA**) drives and **SCSI** drives, or even today when **internet SCSI** (**iSCSI**) devices might be connected in a different order than expected, resulting in device name changes and failures when reaching them.

Remember to use `blkid` or `lsblk -fp` commands to check the filesystems' labels and **universally unique identifiers** (**UUIDs**) that could be used when referring to them.

> **Important note**
> When editing the `/etc/fstab` file, be extremely careful: altering the mount points used by the system might render your system unusable. If in doubt, double-check for any change and be sure to be familiar with system recovery methods and have rescue media available in case this is needed.

Let's learn about mounting a remote NFS in the next section

Using network filesystems with NFS

Mounting a remote NFS is not much different than mounting local devices, but instead of specifying a local device as we did in the previous section with our `/dev/loop0p1` file, we provide `server:export` as a device.

We can find a range of available options by checking the manual page via `man mount`, and this will show us several of the options and the way the device looks.

When a NFS mount is going to be used, the administrator will need to use the host and the export name to mount that device—for example, based on the following data about the NFS export:

- **Server**: `server.example.com`
- **Export**: `/isos`
- **Mount point**: `/mnt/nfs`

With the preceding data, it's easy to construct the `mount` command, which will look like this:

```
mount -t nfs sever.example.com:/isos /mnt/nfs
```

If we analyze the preceding command, it will define the type of filesystem to mount as `nfs`, provided by a `server.example.com` hostname, and using the `/isos` NFS export, and will be made available locally under the `/mnt/nfs` folder.

If we want to define this filesystem as available at boot, we should add an entry in `/etc/fstab`, but... how should we indicate this?

Based on the settings explained during this chapter, the constructed entry would look something like this:

```
server.example.com:/isos /mnt/nfs nfs defaults,_netdev 0 0
```

The preceding line of code contains the parameters we indicated on the command line, but it also adds that it is a resource that required network access before attempting to mount it, as networking is required to be able to reach the NFS server, similar to what will be required for other network-based storage such as Samba mounts, iSCSI, and so on.

Important note

Reinstating the idea of keeping our system bootable, once we make modifications to the `/etc/fstab` configuration file, it is recommended to execute `mount -a` so that the validation is performed from a running system. If after the execution the new filesystems are available and shown when executing—for example, `df`—and no error appeared, it should be safe.

Summary

In this chapter, we've learned about how a disk is divided logically for optimal use of storage and how to later create a filesystem on that disk division so that it can be used to actually store data.

Once the actual filesystem was created, we learned how to make it accessible in our system and how to ensure that it will be available after the next system restart via modification of the `/etc/fstab` configuration file.

Finally, we also learned about using a remote filesystem with NFS based on the data that was provided for us and how to add it to our `fstab` file to make it persistent.

In the next chapter, we will learn how to make storage even more useful via **Logical Volume Management** (**LVM**), which empowers the definition of different logical units that can be resized, combined to provide data redundancy, and so on.

13
Flexible Storage Management with LVM

Managing local storage can be done in a more flexible manner than in *Chapter 12, Managing Local Storage and Filesystems*, by using the **Logical Volume Manager** (**LVM**). LVM allows you to assign more than one disk to the same logical volume (the equivalent in LVM to a partition), have data replicated across different disks, and make snapshots of a volume.

In this chapter, we will review the basic usage of LVM and the main objects that are used to manage storage. We will learn how to prepare disks to be used with LVM, and then aggregate them into a pool, thereby not only increasing the available space, but also enabling you to use it consistently. We will also learn how to distribute that aggregated disk space into partition-like chunks that can easily be extended if necessary. To do so, we will go through the following topics:

- Understanding LVM
- Creating, moving, and removing physical volumes
- Combining physical volumes into volume groups
- Creating and extending logical volumes
- Adding new disks to a volume group and extending a logical volume
- Removing logical volumes, volume groups, and physical volumes
- Reviewing LVM commands

Technical requirements

For this chapter, we will add two more disks to the machine we are working with to be able to follow the examples mentioned in this chapter. These are your options:

- If you are using a physical machine, you may add a couple of USB drives.
- If you are using a local virtual machine, you will need to add two new virtual drives.
- If you are using a cloud instance, you can add two new block devices to it.

As an example, let's see how these disks are added to our virtual machine in Linux. First, we power off the virtual machine we installed in *Chapter 1*, *Installing RHEL8*, called `rhel8`. Then we open the characteristics page of the virtual machine. There we find the **Add Hardware** button:

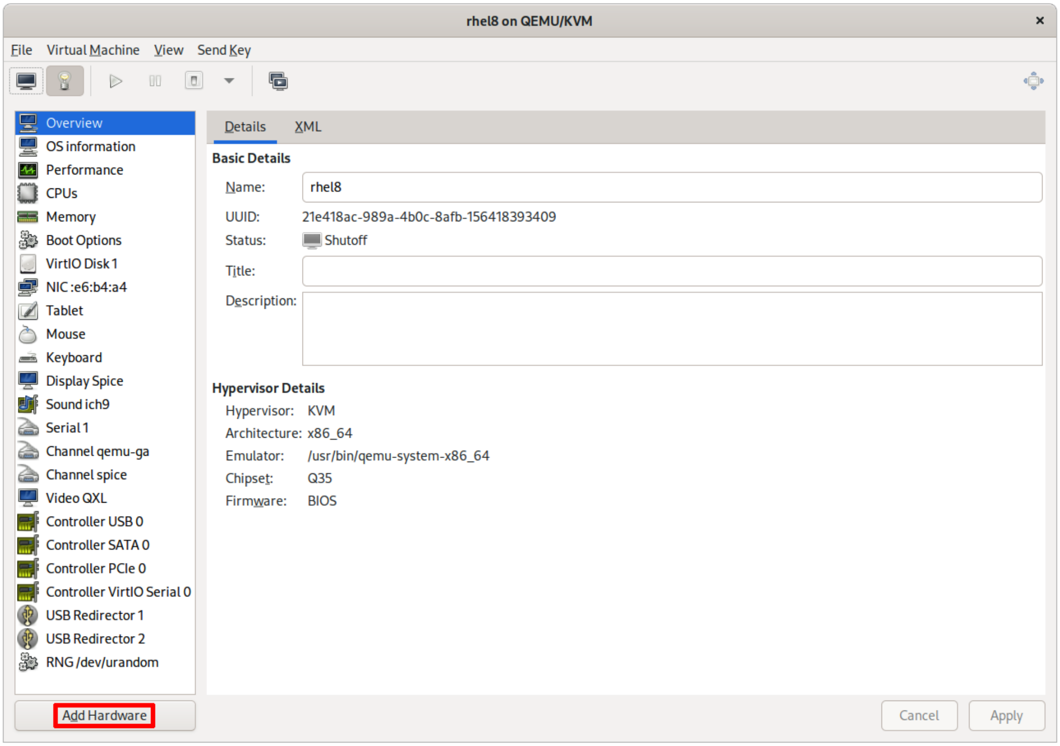


Figure 13.1 – Editing virtual machine properties

Tip

Depending on the virtualization platform you are using, there are different paths to reach the virtual machine characteristics. However, it's very common that there is an option directly accessible from the virtual machine menu.

Clicking on **Add Hardware** will open the dialog in the following screenshot. In it, we will select the **Storage** option and specify the size of the virtual disk to be created and attached to the virtual machine, in this case, 1 GiB, and then click **Finish**:

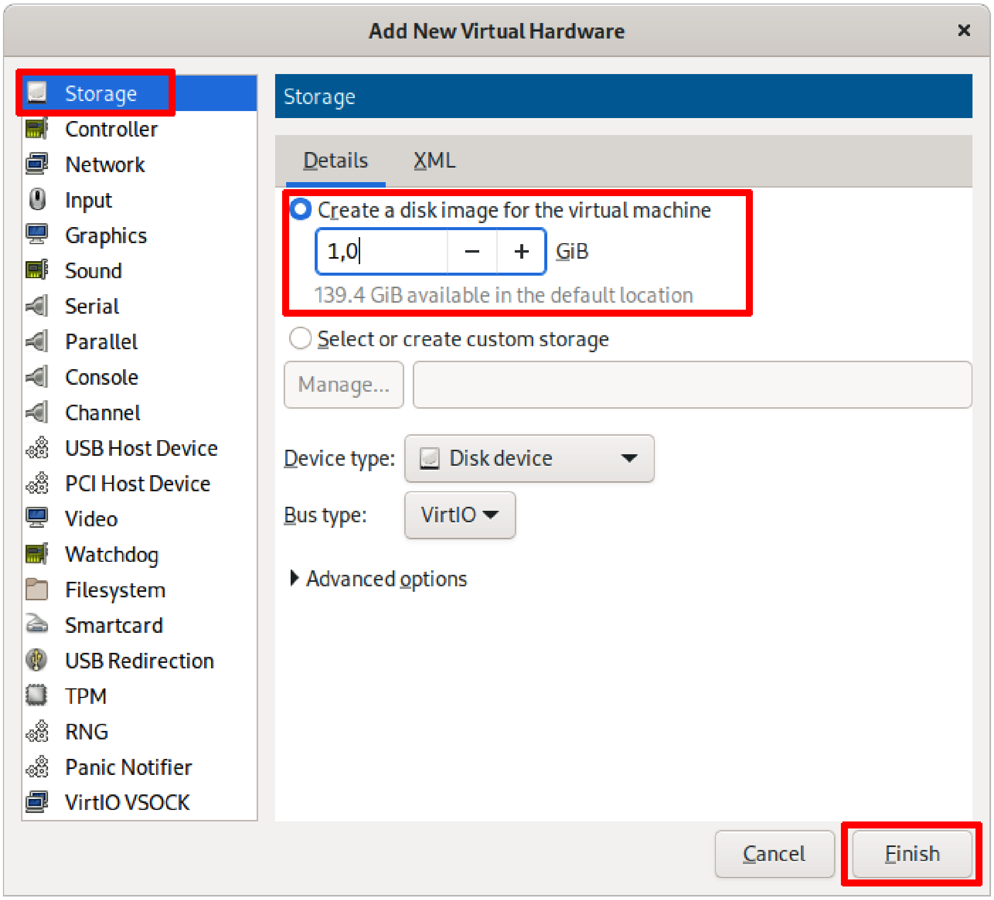


Figure 13.2 – Adding a disk to a virtual machine

We will repeat this procedure twice to add two disks. The end result will look like this:

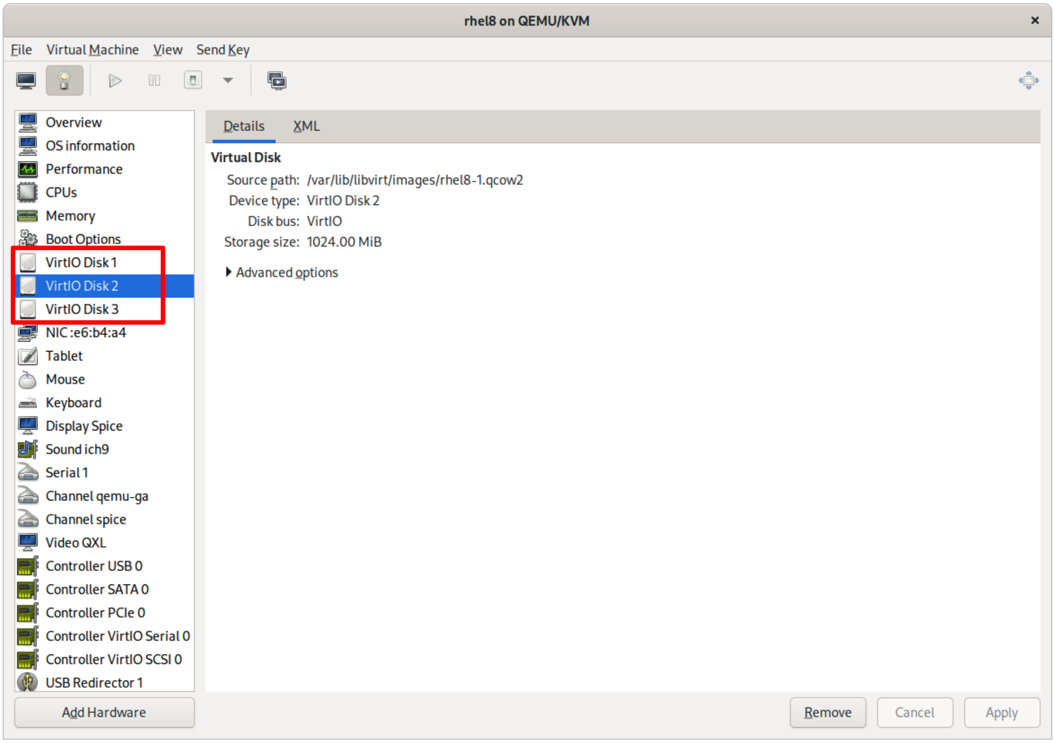


Figure 13.3 – Two new disks added to a virtual machine, making a total of three

We will now power on the virtual machine and log in to it to check the availability of new devices:

```
[root@rhel8 ~]# lsblk
NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda             252:0    0   10G  0 disk
├─vda1          252:1    0    1G  0 part /boot
└─vda2          252:2    0    9G  0 part
  ├─rhel-root   253:0    0    8G  0 lvm  /
  └─rhel-swap   253:1    0    1G  0 lvm  [SWAP]
vdb             252:16   0    1G  0 disk
vdc             252:32   0    1G  0 disk
```

We can see that the new 1 GiB disks, `vdb` and `vdc`, are available. Now that we have a system disk, where we installed the RHEL 8 operating system and two more disks to work with, we are ready to continue with this chapter.

> **Tip**
> The naming of the disk devices in Linux depends on the driver they use. Devices attached as SATA or SCSI show as `sd` and a letter, such as `sda` or `sdb`. Devices connected with IDE buses use `hd` and a letter, such as `hda` or `hdb`. Devices as in the example, which use the VirtIO paravirtualized drivers, use `vd` and a letter, for example, `vda` or `vdb`.

Understanding LVM

LVM uses three layers to manage the storage devices in our systems. These layers are as follows:

- **Physical Volumes** (**PV**): The first layer of LVM . Assigned to the block devices directly. A physical volume can be either a partition on a disk, or a full raw disk itself.
- **Volume Groups** (**VG**): The second layer of LVM. It groups the physical volumes to aggregate space. This is an intermediate layer and not very visible, but its role is very important.
- **Logical Volumes** (**LV**): The third layer of LVM. It distributes the space that the volume groups aggregate.

Let's see the example that we want to implement using the two newly added disks:

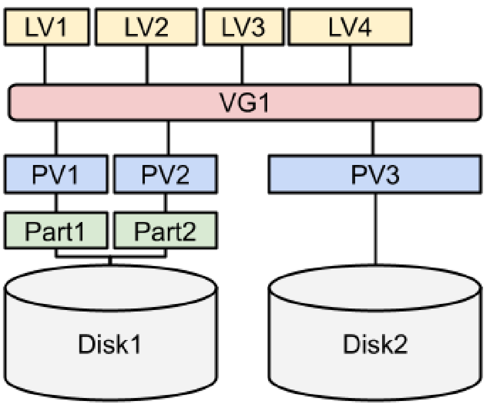


Figure 13.4 – LVM example using two disks

Let's explain this example diagram to understand all the layers:

- We have two disks, which, in the diagram, are **Disk1** and **Disk2**.
- **Disk1** is partitioned into two partitions, **Part1** and **Part2**.
- **Disk2** is not partitioned.
- There are three physical volumes. The mission of these is to prepare the disk space to be used in LVM. The physical volumes are as follows:

  – **PV1**, created on the **Part1** partition of **Disk1**

  – **PV2**, created on the **Part2** partition of **Disk1**

  – **PV3**, created directly on **Disk2**
- One single volume group, **VG1**, aggregates all three physical volumes, **PV1**, **PV2**, and **PV3**. Now, all that disk space is consolidated and can be easily redistributed.
- To distribute the space, there are four logical volumes – **LV1**, **LV2**, **LV3**, and **LV4**. Please note that the logical volumes do not use the whole disk. This way, if we need to extend a volume or create a snapshot, it will be possible.

This is a basic description of how the layers are distributed, without diving into complex cases such as mirroring, thin provisioning, or snapshot.

As a rule of thumb, we need to understand that PVs are designed to prepare devices to be used by LVM, VGs to aggregate PVs, and LVs to distribute the aggregated space.

It is interesting to see that if we create a VG, we can add an extra disk to it, thereby increasing its size without having to stop or reboot the machine. Likewise, we can distribute the added space along the LVs that require it without having to stop or reboot the machine. This is one of the main reasons why LVM is so powerful and recommended for every server, with very few exceptions.

Now that we know the layers into which LVM is divided, let's begin using them to start understanding how they work.

Creating, moving, and removing physical volumes

Having our machine ready with the two new disks, `vdb` and `vdc`, as the *Technical requirements* section explains, we can get started on implementing the example diagram, as shown in *Figure 13.4*, in our machine.

The first step is not directly related to LVM, but it is still important to continue with the example. This first step involves partitioning the `vdb` disk. Let's take a look at this with the tool to manage partitions, `parted`:

```
[root@rhel8 ~]# parted /dev/vdb print
Error: /dev/vdb: unrecognised disk label
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:
```

> **Important note**
> Your disk device, if you are using a physical machine or a different disk driver, may be different. For example, if we were using SATA disks, it would be `/dev/sdb` instead of `/dev/vdb`.

The disk is completely unpartitioned, as we can see in the `unrecognised disk label` message. As is explained in *Chapter 12*, *Managing Local Storage and Filesystems*, there are two types of disk labels that we can use; `msdos` (also referred to as **MBR**), the old type that machines with a **Basic Input Output System** (**BIOS**) can use to boot, and `gpt`, the new type that machines with a **Unified Extensible Firmware Interface** (**UEFI**) can use to boot. In case of doubt, use `gpt`, as we will do in this example. The option to be used with `parted` to create a new label is `mklabel`:

```
[root@rhel8 ~]# parted /dev/vdb mklabel gpt
Information: You may need to update /etc/fstab.

[root@rhel8 ~]# parted /dev/vdb print
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start  End  Size  File system  Name  Flags
```

> **Tip**
>
> To create an `msdos` label, the command would be `parted /dev/vdb mklabel msdos`.

Now we have a disk with a `gpt` label, but without partitions. Let's create a partition using the `mkpart` option in interactive mode:

```
[root@rhel8 ~]# parted /dev/vdb mkpart
```

Now we can enter the partition name, `mypart0`:

```
Partition name?  []? mypart0
```

For the next step, specifying the filesystem, we will use `ext2`:

```
File system type?  [ext2]? ext2
```

Now it is time to set the start point. We will use the first sector available, which is `2048s`:

```
Start? 2048s
```

> **Tip**
>
> The first sector in modern disks is, by definition, `2048s`. This is not provided by the tool. We could review other existing disks, when in doubt, by running `parted /dev/vda unit s print`.

And then we come to the final step, setting the endpoint, which can be described as the size of the partition we want to make:

```
End? 200MB
```

The command is complete with the following warning:

```
Information: You may need to update /etc/fstab.
```

To ensure that the partition table is refreshed in the system, and to allow the devices to be generated under `/dev`, we can run the following command:

```
[root@rhel8 ~]# udevadm settle
```

> **Tip**
>
> The full command to run in non-interactive mode is `parted /dev/vdb mkpart mypart0 xfs 2048s 200MB`.

We can see the new partition available:

```
[root@rhel8 ~]# parted /dev/vdb print
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End    Size   File system  Name     Flags
1       1049kB  200MB  199MB               mypart0
```

We need to change the partition to be able to host `LVM` physical volumes. The `parted` command uses the `set` option to change the partition type. We need to specify the number of the partition, which is `1`, and then type `lvm` and `on` to activate:

```
root@rhel8 ~]# parted /dev/vdb set 1 lvm on
Information: You may need to update /etc/fstab.

[root@rhel8 ~]# udevadm settle
[root@rhel8 ~]# parted /dev/vdb print
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End    Size   File system  Name     Flags
1       1049kB  200MB  199MB               mypart0  lvm
```

We see the flag of the partition now set as `lvm`.

Let's add the second partition, `mypart1`:

```
[root@rhel8 ~]# parted /dev/vdb mkpart mypart1 xfs \
200MB 100%
Information: You may need to update /etc/fstab.

[root@rhel8 ~]# parted /dev/vdb set 2 lvm on
Information: You may need to update /etc/fstab.

[root@rhel8 ~]# parted /dev/vdb print
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name     Flags
1       1049kB  200MB   199MB                mypart0  lvm
2       200MB   1073MB  872MB                mypart1  lvm
```

Now that we have created two partitions, `/dev/vdb1` (with the name `mypart0`) and `/dev/vdb2` (with the name `mypart1`), this is what our storage looks like:

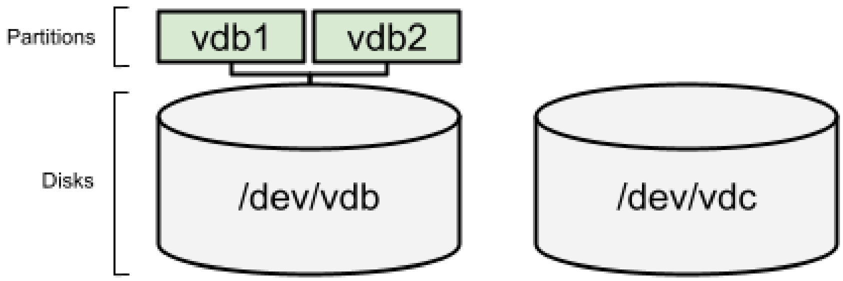


Figure 13.5 – Partitions created in our two new disks

> **Tip**
>
> There is another tool that comes by default in RHEL8 to manage partitions, which is `fdisk`. You may want to give it a try to see whether you find it easier to use.

Now it is time to create persistent volumes. We will do it only on the newly created partitions. First, we check the available persistent volumes with the `pvs` command:

```
[root@rhel8 ~]# pvs
  PV         VG   Fmt  Attr PSize  PFree
  /dev/vda2  rhel lvm2 a--  <9,00g    0
```

Now, we proceed to create the persistent volumes with `pvcreate`:

```
[root@rhel8 ~]# pvcreate /dev/vdb1
  Physical volume "/dev/vdb1" successfully created.
[root@rhel8 ~]# pvcreate /dev/vdb2
  Physical volume "/dev/vdb2" successfully created.
```

And we check they have been created correctly with `pvs` again:

```
[root@rhel8 ~]# pvs
  PV         VG   Fmt  Attr PSize   PFree
  /dev/vda2  rhel lvm2 a--   <9,00g      0
  /dev/vdb1       lvm2 ---  190,00m 190,00m
  /dev/vdb2       lvm2 ---  832,00m 832,00m
```

Notice that persistent volumes do not have their own name, but the name of the partition (or device) they are created on. We can refer to them as `PV1` and `PV2` to draw the diagram.

This is now the status:

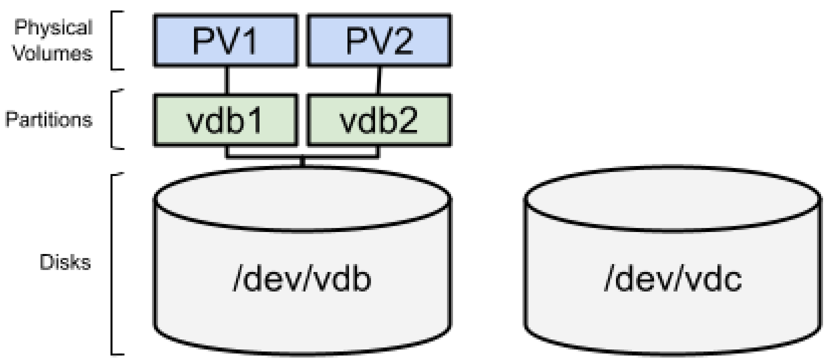


Figure 13.6 – Persistent volumes created in the two new partitions

We can also create a persistent volume directly on the disk device, `vdc`. Let's do it:

```
[root@rhel8 ~]# pvcreate /dev/vdc
  Physical volume "/dev/vdc" successfully created.
[root@rhel8 ~]# pvs
  PV         VG   Fmt  Attr PSize   PFree
  /dev/vda2  rhel lvm2 a--   <9,00g      0
  /dev/vdb1       lvm2 ---  190,00m 190,00m
  /dev/vdb2       lvm2 ---  832,00m 832,00m
  /dev/vdc        lvm2 ---    1,00g   1,00g
```

As in the previous example, there is no name for the physical volume, which we will refer to as `PV3`. The result looks like the following:

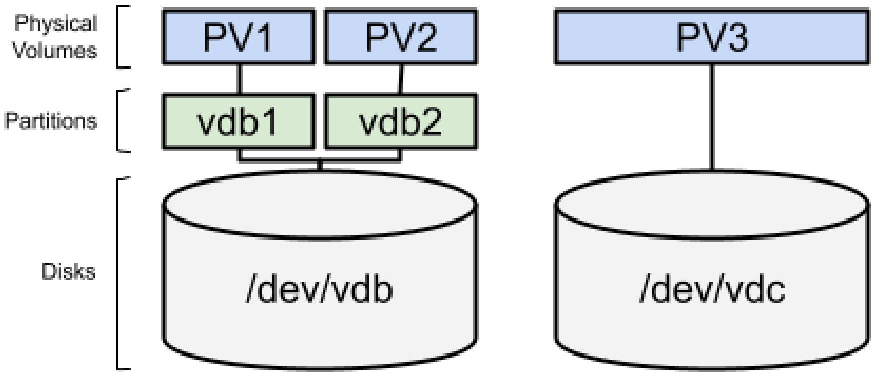


Figure 13.7 – Persistent volumes created in the two new partitions and the new disk device

Now that we have the persistent volumes, let's group them using virtual groups in the next section.

Combining physical volumes into volume groups

It's time to create a new volume group with the physical volumes added previously. Before doing that, we can check the volume groups available with the `vgs` command:

```
[root@rhel8 ~]# vgs
  VG   #PV #LV #SN Attr   VSize  VFree
  rhel   1   2   0 wz--n- <9,00g    0
```

We can see that only the volume group created during installation for the operating system is available. Let's create our `storage` volume group with the `/dev/vdb1` and `/dev/vdb2` partitions using the `vgcreate` command:

```
[root@rhel8 ~]# vgcreate storage /dev/vdb1 /dev/vdb2
  Volume group "storage" successfully created
[root@rhel8 ~]# vgs
  VG      #PV #LV #SN Attr   VSize    VFree
  rhel      1   2   0 wz--n-   <9,00g        0
  storage   2   0   0 wz--n- 1016,00m 1016,00m
```

As you can see, the new `storage` volume group has been created. The diagram of the current status would now look like this:

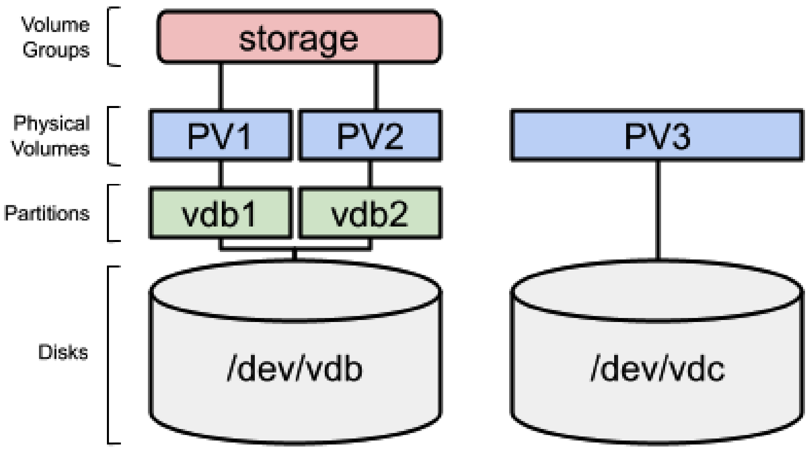


Figure 13.8 – First volume group created with two physical volumes

> **Important note**
>
> **Volume groups** are a very thin layer in LVM whose only goal is to aggregate disks or partitions into a pool of storage. The advanced management of that storage, such as, for example, having data mirrored in two different disks, is done with logical volumes.

We have already prepared the partitions and disks as physical volumes and aggregated them into volume groups, so we have a pool of disk space. Let's move on to the next section to learn how the distribution of that disk space can be done using logical volumes.

Creating and extending logical volumes

We currently have several physical volumes created and two of them grouped into a volume group. Let's move to the next layers and check the logical volumes with the `lvs` command:

```
[root@rhel8 ~]# lvs
  LV   VG   Attr       LSize  Pool Origin Data%  Meta%  Move
Log Cpy%Sync Convert
  root rhel -wi-ao---- <8,00g
  swap rhel -wi-ao----  1,00g
```

We see the `root` and `swap` volumes on the `rhel` volume group, which hosts the operating system.

Now, we can create a simple logical volume called `data`, 200 MB in size, on the `storage` volume group:

```
[root@rhel8 ~]# lvcreate --name data --size 200MB storage
  Logical volume "data" created.
[root@rhel8 ~]# lvs
  LV   VG      Attr       LSize   Pool Origin Data%  Meta%
Move Log Cpy%Sync Convert
  root rhel    -wi-ao----  <8,00g
  swap rhel    -wi-ao----   1,00g
  data storage -wi-a----- 200,00m
```

Our configuration now looks like this:

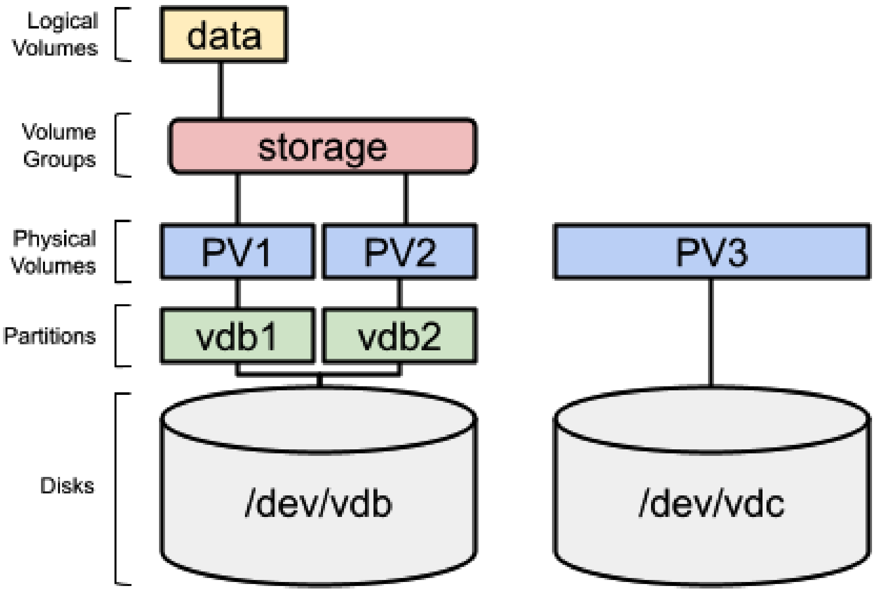


Figure 13.9 – First logical created using space from a volume group

The logical volume created is a block device, and behaves similarly to a disk partition. Therefore, in order to be used, we need to format it with a filesystem. Let's do this by formatting it with the `xfs` format:

```
[root@rhel8 ~]# mkfs.xfs /dev/storage/data
meta-data=/dev/storage/data      isize=512  agcount=4,
agsize=12800 blks
         =                       sectsz=512   attr=2,
projid32bit=1
         =                       crc=1 finobt=1, sparse=1,
rmapbt=0
         =                       reflink=1
data     =                       bsize=4096
blocks=51200,imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0,
ftype=1
log      =internal log           bsize=4096  blocks=1368,
version=2
```

```
         =                        sectsz=512 sunit=0 blks, lazy-
count=1
realtime =none                    extsz=4096   blocks=0,
rtextents=0
Discarding blocks...Done.
```

Now it's ready to be mounted. We can create the `/srv/data` directory and mount it there:

```
[root@rhel8 ~]# mkdir /srv/data
[root@rhel8 ~]# mount -t xfs /dev/storage/data /srv/data
[root@rhel8 ~]# df -h /srv/data/
Filesystem                 Size  Used Avail Use% Mounted on
/dev/mapper/storage-data  195M   12M  184M   6% /srv/data
```

We have set up our LVM-enabled space available in our system. Mounting a filesystem manually, as in the previous example, works while the system is not powered down or rebooted. To make it persistent, we need to add the following line to `/etc/fstab`:

```
/dev/storage/data   /srv/data    xfs    defaults        0 0
```

To test that the line is correctly written, we can run the following commands. First, dismount the filesystem:

```
[root@rhel8 ~]# umount /srv/data
```

Check the available space in the mount point:

```
[root@rhel8 ~]# df -h /srv/data/
Filesystem             Size  Used Avail Use% Mounted on
/dev/mapper/rhel-root  8,0G  2,8G  5,3G  35% /
```

The output of the `df` (for *disk free*) command shows that the space in the `/srv/data/` directory is related to the `root` partition, meaning that the folder does not have any filesystem associated with it. Let's now run the `mount` command when the system is starting:

```
[root@rhel8 ~]# mount -a
```

All filesystems in /etc/fstab that are not mounted will be mounted, or an error will be shown if there is any issue with them (such as a typo in /etc/fstab). Let's check that it is mounted:

```
[root@rhel8 ~]# df -h /srv/data/
Filesystem                  Size  Used Avail Use% Mounted on
/dev/mapper/storage-data    195M   12M  184M   6% /srv/data
```

> **Important note**
>
> The /dev/storage/data and /dev/mapper/storage-data devices are aliases (or symbolic links, to be more precise) of the same device generated by a component called **device mapper**. They are fully interchangeable.

As we can see, the filesystem is properly mounted. Now that we know how to create a logical volume and assign a filesystem and mount point to it, we can move on to more advanced tasks, such as extending disk space in our LVM layers and beyond.

Adding new disks to a volume group and extending a logical volume

One of the great things about LVM, and more specifically, volume groups, is that we can add a new disk to it and start using that newly extended space. Let's try it by adding the physical volume in /dev/vdc to the storage volume group:

```
[root@rhel8 ~]# vgs
  VG      #PV #LV #SN Attr   VSize    VFree
  rhel      1   2   0 wz--n-   <9,00g      0
  storage   2   1   0 wz--n- 1016,00m 816,00m
[root@rhel8 ~]# vgextend storage /dev/vdc
  Volume group "storage" successfully extended
[root@rhel8 ~]# vgs
  VG      #PV #LV #SN Attr   VSize  VFree
  rhel      1   2   0 wz--n- <9,00g    0
  storage   3   1   0 wz--n- <1,99g 1,79g
```

Now, our disk distribution looks like this:

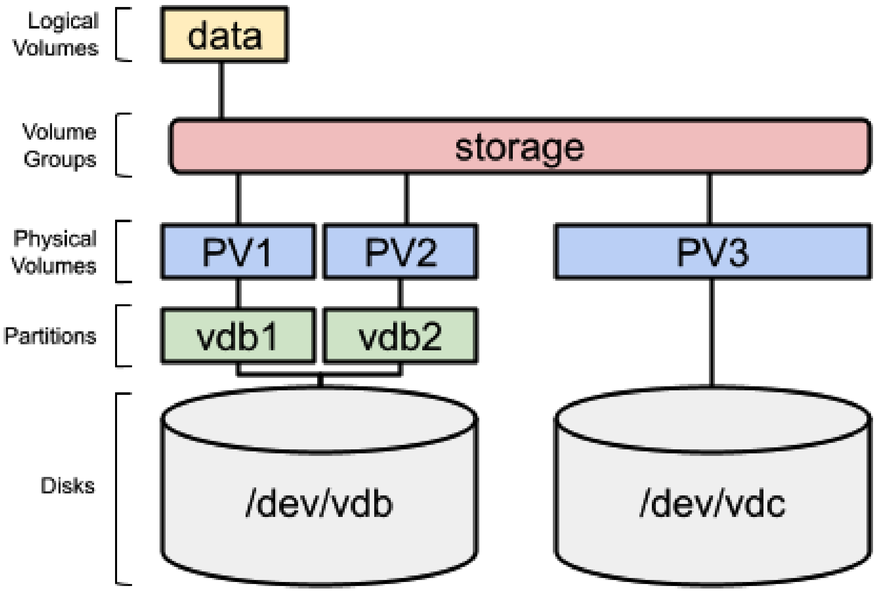


Figure 13.10 – Extended volume group with three physical volumes

Let's now extend the `data` logical volume by adding 200 MB to it:

```
[root@rhel8 ~]# lvs
  LV   VG      Attr       LSize   Pool Origin Data%  Meta%
 Move Log Cpy%Sync Convert
  root rhel    -wi-ao----  <8,00g
  swap rhel    -wi-ao----   1,00g
  data storage -wi-ao---- 200,00m
[root@rhel8 ~]# lvextend --size +200MB /dev/storage/data
  Size of logical volume storage/data changed from 200,00 MiB
(50 extents) to 400,00 MiB (100 extents).
  Logical volume storage/data successfully resized.
[root@rhel8 ~]# lvs
  LV   VG      Attr       LSize   Pool Origin Data%  Meta%
 Move Log Cpy%Sync Convert
  root rhel    -wi-ao----  <8,00g
  swap rhel    -wi-ao----   1,00g
  data storage -wi-ao---- 400,00m
```

The logical volume has been extended. However, the filesystem on top of it hasn't:

```
[root@rhel8 ~]# df -h /srv/data/
Filesystem                  Size  Used Avail Use% Mounted on
/dev/mapper/storage-data   195M   12M  184M   6% /srv/data
```

We need to extend the filesystem. The tool to do so depends on the type of filesystem. In our case, as it is `xfs`, the tool to extend it is `xfs_growfs`. Let's do it:

```
[root@rhel8 ~]# xfs_growfs /dev/storage/data
meta-data=/dev/mapper/storage-data isize=512    agcount=4,
agsize=12800 blks
         =                       sectsz=512   attr=2,
projid32bit=1
         =                       crc=1 finobt=1, sparse=1,
rmapbt=0
         =                       reflink=1
data     =                       bsize=4096 blocks=51200
imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0,
ftype=1
log      =internal log           bsize=4096   blocks=1368
version=2
         =                       sectsz=512   sunit=0 blks,
lazy-count=1
realtime =none                   extsz=4096   blocks=0,
rtextents=0
data blocks changed from 51200 to 102400
[root@rhel8 ~]# df -h /srv/data/
Filesystem                  Size  Used Avail Use% Mounted on
/dev/mapper/storage-data   395M   14M  382M   4% /srv/data
```

And now, the filesystem has some extra space added and available.

> **Important note**
>
> When doing this task, the logical volume can be mounted and be used by the system. LVM is ready to do volume extensions on production systems while running.

It's very easy to redistribute the space and add another logical volume:

```
[root@rhel8 ~]# lvcreate --size 100MB --name img storage
  Logical volume "img" created.
[root@rhel8 ~]# lvs
  LV   VG      Attr       LSize   Pool Origin Data%  Meta%
Move Log Cpy%Sync Convert
  root rhel    -wi-ao----  <8,00g
  swap rhel    -wi-ao----   1,00g
  data storage -wi-ao---- 400,00m
  img  storage -wi-a----- 100,00m
[root@rhel8 ~]# mkfs.xfs /dev/storage/img
meta-data=/dev/storage/img       isize=512    agcount=4,
agsize=6400 blks
         =                       sectsz=512   attr=2,
projid32bit=1
         =                       crc=1 finobt=1, sparse=1,
rmapbt=0
         =                       reflink=1
data     =                       bsize=4096 blocks=25600
imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0,
ftype=1
log      =internal log           bsize=4096  blocks=1368,
version=2
         =                       sectsz=512   sunit=0 blks,
lazy-count=1
realtime =none                   extsz=4096   blocks=0,
rtextents=0
Discarding blocks...Done.
[root@rhel8 ~]# mkdir /srv/img
[root@rhel8 ~]# mount -t xfs /dev/storage/img /srv/img
[root@rhel8 ~]# df /srv/img/
Filesystem             1K-blocks  Used Available Use% Mounted
on
/dev/mapper/storage-img     96928  6068     90860   7% /srv/img
[root@rhel8 ~]# df -h /srv/img/
```

```
Filesystem                  Size  Used Avail Use% Mounted on
/dev/mapper/storage-img      95M  6,0M   89M   7% /srv/img
```

The `--size` and `--extents` options for the `lvcreate` command have several options that can be used to define the space to be consumed:

- **Human-readable**: We can define the size in human-readable blocks, such as gigabytes, using `GB`, or megabytes, using `MB` (in other words, `--size 3GB`).
- **Extents**: If we just provide a number after `--extents`, the command will use its internal measure, `extents`, which is similar to block size for disk partitions (that is, `--extents 125`).

The `--size` and `--extents` options also apply to the `lvextend` command. In this case, we can use the options shown previously for `lvcreate` to define the new size of a logical volume. We also have other options to define increments of space to be assigned to them:

- **Adding space**: If we provide the + symbol before the number to `lvextend`, this will increase the size in the measurement provided (that is, `--size +1GB` adds one extra gigabyte to the current logical volume).
- **Percentage of free space**: We can provide the percentage of free space to be created or extended by using `--extents`, and the percentage of free space to be used followed by `%FREE` (that is, `--extents 10%FREE`).

> **Tip**
>
> As we have seen previously in other tools, we can use the manual pages to remind ourselves of the options available. Please run `man lvcreate` and `man lvextend` to get familiar with the pages for these tools.

We are going to create a logical volume to be used as **swap**, which is a part of the disk that the system uses as a parking space for memory. The system puts processes that consume memory and that are not active there so that the physical memory (which is much faster than disk) is freed. It is also used when there is no more free physical memory in the system.

Let's create a swap device on LVM:

```
[root@rhel8 ~]# lvcreate --size 100MB --name swap storage
  Logical volume "swap" created.
[root@rhel8 ~]# mkswap /dev/storage/swap
Setting up swapspace version 1, size = 100 MiB (104853504
bytes)
no label, UUID=70d07e58-7e8d-4802-8d20-38d774ae6c22
```

We can check the memory and swap status with the `free` command:

```
[root@rhel8 ~]# free
              total        used        free      shared  buff/
cache   available
Mem:        1346424      218816      811372        9140
316236      974844
Swap:       1048572           0     1048572
[root@rhel8 ~]# swapon /dev/storage/swap
[root@rhel8 ~]# free
              total        used        free      shared  buff/
cache   available
Mem:        1346424      219056      811040        9140
316328      974572
Swap:       1150968           0     1150968
```

Important note

The two new changes would require adding a line for each to `/etc/fstab` to make use of them persistently across reboots.

Our disk space distribution would now look like this:

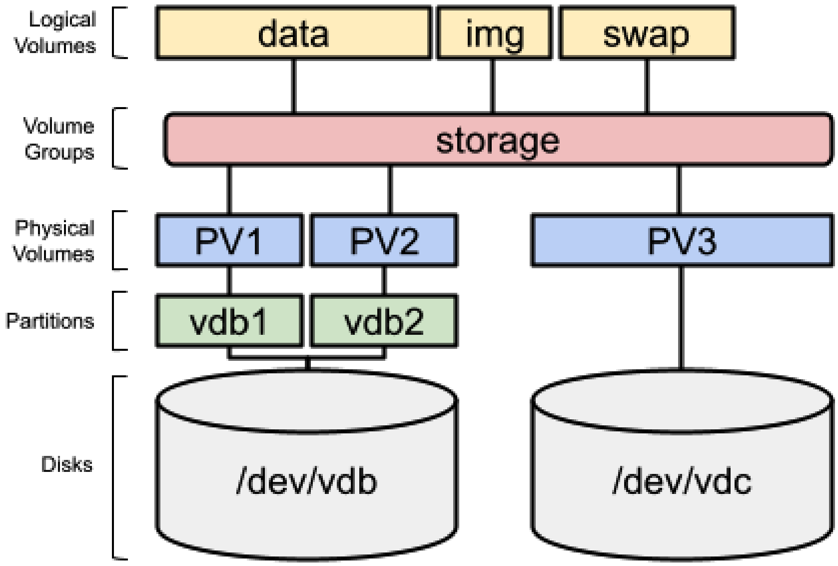


Figure 13.11 – Extended volume group with three physical volumes

This distribution looks a lot like the initial example we used to describe the layers of LVM. We now have practiced with all the layers to create the pieces required in each one of them. We know how to create, so now it's time to learn how to remove them in the next section.

Removing logical volumes, volume groups, and physical volumes

To start with the commands used to remove, let's do the simple step of removing the `img` logical volume. First, we need to check whether it's mounted:

```
[root@rhel8 ~]# mount | grep img
/dev/mapper/storage-img on /srv/img type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,
logbsize=32k,noquota)
```

As it is mounted, we need to dismount it:

```
[root@rhel8 ~]# umount /srv/img
[root@rhel8 ~]# mount | grep img
```

The last command shows an empty output, which means that it isn't mounted.
Let's proceed to remove it:

```
[root@rhel8 ~]# lvremove /dev/storage/img
Do you really want to remove active logical volume storage/img?
[y/n]: y
  Logical volume "img" successfully removed
```

Now, we can also remove the mount point:

```
[root@rhel8 ~]# rmdir /srv/img
```

And the removal of the logical volume is done. This process is not reversible, so run it carefully. Our disk distributions now look like this:

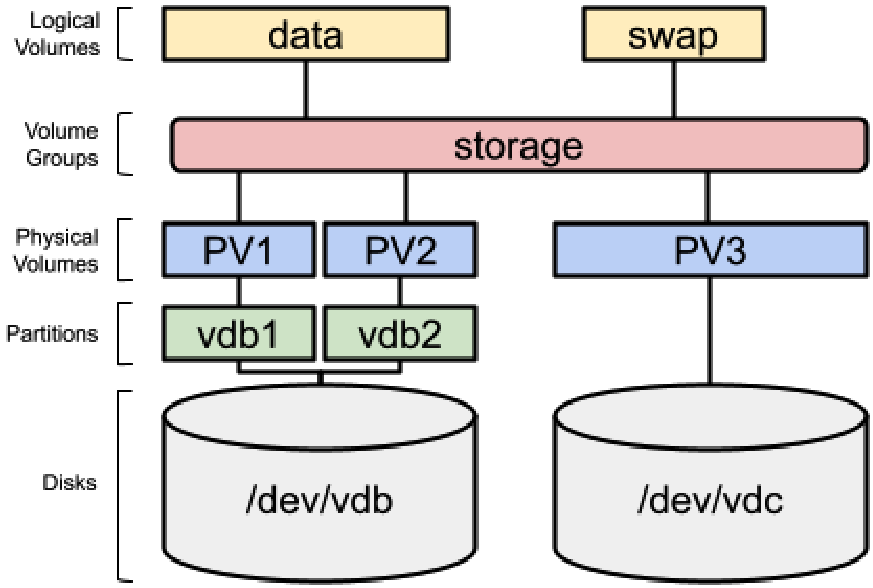


Figure 13.12 – Volume group with logical volume removed

Now it's time for a more complex task, removing a physical volume from a virtual group. The reason for doing so is that sometimes you want to transfer the data stored on a physical disk to a different disk, and then detach it and remove it from the system. This can be done, but first, let's add some files to the `data` logical volume:

```
[root@rhel8 ~]# cp -ar /usr/share/scap-security-guide \
/srv/data/
```

```
[root@rhel8 ~]# ls /srv/data/
scap-security-guide
[root@rhel8 ~]# du -sh /srv/data/
30M  /srv/data/
```

Now let's evacuate the data from `/dev/vdb1` using the `pvmove` command:

```
[root@rhel8 ~]# pvmove /dev/vdb1
  /dev/vdb1: Moved: 7,75%
  /dev/vdb1: Moved: 77,52%
  /dev/vdb1: Moved: 100,00%
```

> **Important note**
> Depending on the allocation of the extents, you may receive a message stating `no data to move for storage`. This means that the saved data was already allocated to the other disk. You can use `pvmove` with other devices to try it.

Now there is no data stored in `/dev/vdb1` and it can be removed from the volume group. We can do so by using the `vgreduce` command:

```
[root@rhel8 ~]# vgreduce storage /dev/vdb1
  Removed "/dev/vdb1" from volume group "storage"
```

We can see that there is now less space in the storage volume group:

```
[root@rhel8 ~]# vgs
  VG      #PV #LV #SN Attr   VSize  VFree
  rhel      1   2   0 wz--n- <9,00g    0
  storage   2   2   0 wz--n-  1,80g 1,30g
[root@rhel8 ~]# vgdisplay storage
  --- Volume group ---
  VG Name               storage
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  20
  VG Access             read/write
  VG Status             resizable
```

```
  MAX LV                0
  Cur LV                2
  Open LV               2
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               1,80 GiB
  PE Size               4,00 MiB
  Total PE              462
  Alloc PE / Size       129 / 516,00 MiB
  Free  PE / Size       333 / 1,30 GiB
  VG UUID               1B6Nil-rvcM-emsU-mBLu-wdjL-mDlw-66dCQU
```

We can also see that the physical volume, `/dev/vdb1`, is not attached to any volume group:

```
[root@rhel8 ~]# pvs
  PV         VG      Fmt  Attr PSize    PFree
  /dev/vda2  rhel    lvm2 a--    <9,00g       0
  /dev/vdb1          lvm2 ---   190,00m  190,00m
  /dev/vdb2  storage lvm2 a--   828,00m  312,00m
  /dev/vdc   storage lvm2 a--  1020,00m 1020,00m
[root@rhel8 ~]# pvdisplay /dev/vdb1
  "/dev/vdb1" is a new physical volume of "190,00 MiB"
  --- NEW Physical volume ---
  PV Name               /dev/vdb1
  VG Name
  PV Size               190,00 MiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               veOsec-WV0n-JP9D-WMz8-UYeZ-Zjs6-sJSJst
```

> **Tip**
> The `vgdisplay`, `pvdisplay`, and `lvdisplay` commands show detailed information on any of the parts of LVM.

The most important part is that we can do these operations while the system is running production workloads with confidence. Our disk distribution now looks like this:

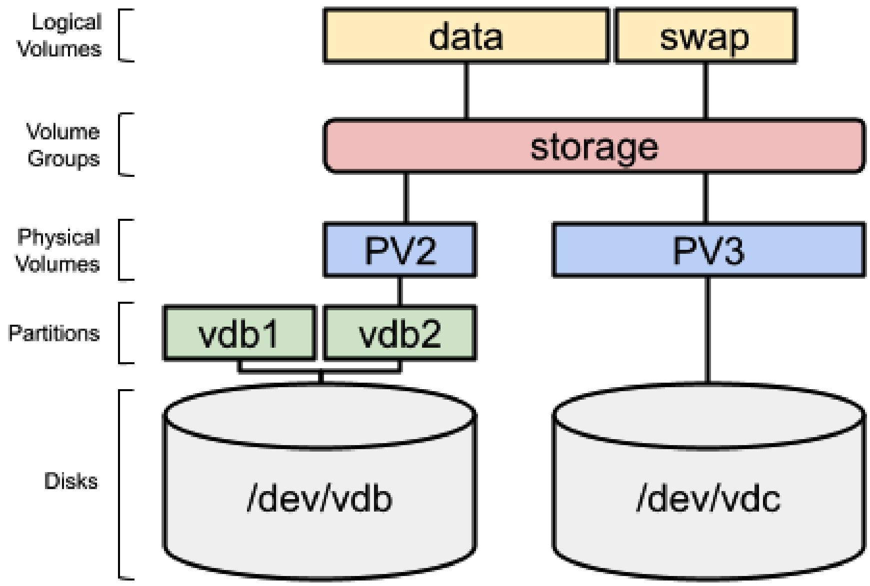


Figure 13.13 – Volume group with physical volumes removed

Now it's time to remove the volume group, but we need to remove the logical volumes first, just like we did before (feel free to run `lvs` and `vgs` before and after each command to check progress):

```
[root@rhel8 ~]# swapoff /dev/storage/swap
[root@rhel8 ~]# lvremove /dev/storage/swap
Do you really want to remove active logical volume storage/
swap? [y/n]: y
  Logical volume "swap" successfully removed
```

With this, we have removed `/dev/storage/swap`. Let's now remove `/dev/storage/data`, using the `--yes` option so that we do not get asked for confirmation (important when using this command in a script):

```
[root@rhel8 ~]# umount /dev/storage/data
[root@rhel8 ~]# lvremove --yes /dev/storage/data
  Logical volume "data" successfully removed
```

Now it's time to remove the `storage` volume group:

```
[root@rhel8 ~]# vgremove storage
```

The `storage` volume group has been removed successfully.

And finally, clean the physical volumes:

```
[root@rhel8 ~]# pvremove /dev/vdb1 /dev/vdb2
  Labels on physical volume "/dev/vdb1" successfully wiped.
  Labels on physical volume "/dev/vdb2" successfully wiped.
```

And with this, we know how to work with each part of the LVM in our RHEL8 systems. Let's review the commands used in the next section.

Reviewing LVM commands

As a summary of the commands used to manage physical volumes, let's take a look at the following table:

Command	Usage
`pvcreate`	Creates a physical volume on a partition or disk (also referred to as a block device)
`pvs`	Shows basic information of the physical volumes in the system
`pvdisplay`	Shows extended information of the physical volumes in the system
`pvmove`	Evacuates data from a physical volume, moving it to other available physical volumes
`pvremove`	Removes physical volumes

Now, let's review the commands used to manage volume groups:

Command	Usage
`vgcreate`	Creates a volume group aggregating different physical volumes
`vgs`	Shows basic information of the volume groups in the system
`vgdisplay`	Shows extended information of the volume groups in the system
`vgextend`	Extends an existing volume group by adding new physical volumes to it
`vgreduce`	Removes evacuated physical volumes from a volume group
`vgremove`	Removes a volume group

And finally, let's review the commands used to manage logical volumes:

Command	Usage
`lvcreate`	Creates a logical volume on a volume group, preparing it to be formatted with a filesystem
`lvs`	Shows basic information of the logical volumes in the system
`lvdisplay`	Shows extended information of the logical volumes in the system
`lvextend`	Extends an existing logical volume by adding available space from the volume group to it
`lvremove`	Removes a logical volume

Remember that you can always use the manual pages available for each command to get more information on the options you want to use and learn new ones, just by running `man <command>`.

> **Important note**
> The web administration interface Cockpit, has an extension for managing storage components. It can be installed as `root` (or with `sudo`) with the following command, `dnf install cockpit-storaged`. A good exercise for you would be to repeat the process done in this chapter using the storage interface in Cockpit.

Summary

The LVM is an incredibly useful part of Red Hat Enterprise Linux and provides the capabilities to manage, reallocate, distribute, and assign disk space without having to stop anything in the system. Battle-tested over the years, it is a key component for system administrators, as well as facilitating the incorporation of other extended capabilities in our systems (a flexible way to provide storage to be shared via iSCSI).

Practicing LVM on test machines is extremely important, so we can be sure that the command we will run on a production system will not mean the service being stopped or data being lost.

In this chapter, we have seen the most basic, yet important, tasks that can be done with LVM. We have learned how the different layers of LVM work: physical volumes, volume groups, and logical volumes. Also, we've seen how they interact with each other and how they can be managed. We have practiced creating, extending, and removing logical volumes, volume groups, and physical volumes. It will be important to practice them to consolidate the knowledge acquired and be able to use them in production systems. However, the basis for doing so is now already in place.

Now, let's move on to the next chapter to discover a new feature in RHEL8 to improve the storage layer further by adding deduplication capabilities to it – **Virtual Data Optimizer** (**VDO**).

14
Advanced Storage Management with Stratis and VDO

In this chapter, we will learn about **Stratis** and **Virtual Data Optimizer** (**VDO**).

Stratis is a storage management tool to simplify running the most typical daily tasks. It uses the underlying technologies explained in the previous chapters, such as LVM, partition schemas, and filesystems.

VDO is a storage layer that includes a driver that sits between our applications and the storage devices to provide deduplication and compression of the data stored, as well as tools to manage this functionality. This will allow us, for example, to maximize the ability of our system to hold virtual machine (VM) instances that will only consume disk space based on what makes them unique, but just storing once the data that is common to them.

We can also use VDO for storing different copies of our backups, knowing that disk usage will still be optimized.

By the end of this chapter, we will know how VDO works and what is required to set it up for our system.

We will explore how to prepare, configure, and use our systems in the following sections:

- Understanding Stratis
- Installing and enabling Stratis
- Managing storage pools and filesystems with Stratis
- Preparing systems to use VDO
- Creating a VDO volume
- Assigning a VDO volume to LVM
- Testing a VDO volume and reviewing stats

Let's jump into preparing our systems to use VDO.

Technical requirements

It is possible to continue the practice of using the VM created at the beginning of this book in *Chapter 1*, *Installing RHEL8*. Any additional packages required for this chapter will be indicated and can be downloaded from `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration`.

We will need, for the *Understanding Stratis* section, the same two disks added in *Chapter 13*, *Flexible Storage Management with LVM*, after all the LVM components have been cleaned up from them.

Understanding Stratis

As a new feature, to manage storage, **Stratis** was included in RHEL 8 as a technology preview (as of version 8.3 of RHEL). Stratis was created to manage local storage by combining a system service, **stratisd**, with the well-known tools in LVM (explained in *Chapter 13*, *Flexible Storage Management with LVM*) and the XFS filesystem (explained in *Chapter 12*, *Managing Local Storage and Filesystems*), which makes it very solid and reliable.

> **Important note**
>
> The filesystems/pools created with Stratis should always be managed with it, and not with the LVM/XFS tools. In the same way, already-created LVM volumes should not be managed with Stratis.

Stratis combines local disks into **pools** and then distributes the storage in **filesystems**, as shown in the following diagram:

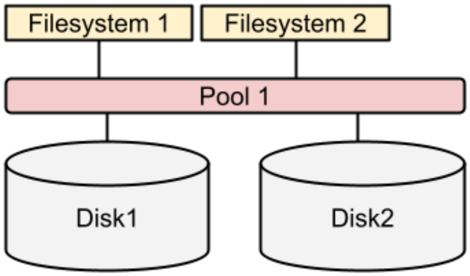


Figure 14.1 – Stratis simplified architecture diagram

As can be seen, when compared to LVM, Stratis provides a much simpler and easy-to-understand interface to storage management. In the following sections, we will install and enable Stratis and then use the same disks created in *Chapter 13*, *Flexible Storage Management with LVM*, to create a pool and a couple of filesystems.

Installing and enabling Stratis

To be able to work with Stratis, we will start by installing it. The two packages required to work with it are these:

- `stratis-cli`: A command-line tool to execute storage management tasks
- `stratisd`: A system service (also known as a daemon) that takes commands and executes the low-level tasks

To install them, we will use the `dnf` command:

```
[root@rhel8 ~]# dnf install stratis-cli stratisd
Updating Subscription Management repositories.
Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
17 MB/s |  32 MB     00:01
Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
12 MB/s |  30 MB     00:02
Dependencies resolved.
===============================================================
====================================
Package                                 Arch    Version
Repository                               Size
===============================================================
```

```
=====================================
Installing:
stratis-cli                          noarch  2.3.0-3.el8
rhel-8-for-x86_64-appstream-rpms      79 k
stratisd                             x86_64  2.3.0-2.el8
rhel-8-for-x86_64-appstream-rpms     2.1 M
[omitted]
Complete!
```

Now we can start the `stratisd` service with `systemctl`:

```
[root@rhel8 ~]# systemctl start stratisd
[root@rhel8 ~]# systemctl status stratisd
● stratisd.service - Stratis daemon
   Loaded: loaded (/usr/lib/systemd/system/stratisd.service;
enabled; vendor preset: enabled)
   Active: active (running) since Sat 2021-05-22 17:31:35 CEST;
53s ago
     Docs: man:stratisd(8)
Main PID: 17797 (stratisd)
    Tasks: 1 (limit: 8177)
   Memory: 1.2M
   CGroup: /system.slice/stratisd.service
           └─17797 /usr/libexec/stratisd --log-level debug
[omitted]
```

Now we shall enable it to start at boot:

```
[root@rhel8 ~]# systemctl enable stratisd
[root@rhel8 ~]# systemctl status stratisd
● stratisd.service - Stratis daemon
   Loaded: loaded (/usr/lib/systemd/system/stratisd.service;
enabled; vendor preset: enabled)
[omitted]
```

Tip

We can do both tasks with one command, which would be `systemctl enable --now stratisd`.

Let's check with `stratis-cli` that the daemon (also known as system service) is running:

```
[root@rhel8 ~]# stratis daemon version
2.3.0
```

We have it all ready, so it's time to start working on disks. Let's move on to the next sub-section.

Managing storage pools and filesystems with Stratis

In order to have some storage available for Stratis, we will use the `/dev/vdb` and `/dev/vdc` disks. We need to be sure that they do not have any logical volumes or partitions on them. Let's review them:

```
[root@rhel8 ~]# lvs
  LV   VG   Attr       LSize  Pool Origin Data%  Meta%  Move
Log Cpy%Sync Convert
  root rhel -wi-ao---- <8,00g
  swap rhel -wi-ao----  1,00g
[root@rhel8 ~]# vgs
  VG   #PV #LV #SN Attr   VSize  VFree
  rhel   1   2   0 wz--n- <9,00g    0
[root@rhel8 ~]# pvs
  PV         VG   Fmt  Attr PSize  PFree
  /dev/vda2  rhel lvm2 a--  <9,00g    0
```

We are good: all the LVM-created objects are on disk `/dev/vda`. Let's check the other two disks, `/dev/vdb` and `/dev/vdc`:

```
[root@rhel8 ~]# parted /dev/vdb print
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start  End  Size  File system  Name  Flags
```

```
[root@rhel8 ~]# parted /dev/vdc print
Error: /dev/vdc: unrecognised disk label
Model: Virtio Block Device (virtblk)
Disk /dev/vdc: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:
```

Disk `/dev/vdc` has no partition table label. We are good with this one. However, disk `/dev/vdb` has a partition table. Let's remove it:

```
[root@rhel8 ~]# dd if=/dev/zero of=/dev/vdb count=2048 bs=1024
2048+0 records in
2048+0 records out
2097152 bytes (2,1 MB, 2,0 MiB) copied, 0,0853277 s, 24,6 MB/s
```

> **Tip**
>
> The `dd` command, which stands for disk dump, is used to dump data from devices and to devices. The special device `/dev/zero` simply generates zeroes, which we use to overwrite the initial sectors of the disk, where the label lives. Please use `dd` with care; it may overwrite anything without warning.

Now we are ready to create the first pool with the `stratis` command:

```
[root@rhel8 ~]# stratis pool create mypool /dev/vdb
[root@rhel8 ~]# stratis pool list
Name                         Total Physical   Properties
mypool   1 GiB / 37.63 MiB / 986.37 MiB       ~Ca,~Cr
```

We currently have the pool created, as shown in the following diagram:

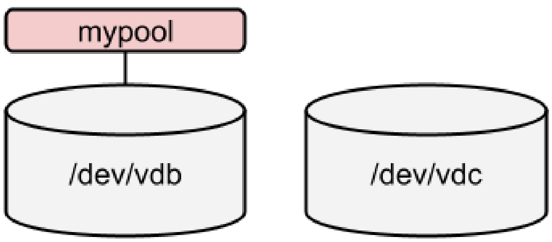


Figure 14.2 – Stratis pool created

We have the pool created; we can now create a filesystem on top of it:

```
[root@rhel8 ~]# stratis filesystem create mypool data
[root@rhel8 ~]# stratis filesystem list
Pool Name    Name    Used       Created              Device
UUID
mypool       data    546 MiB    May 23 2021 19:16    /dev/stratis/
mypool/data   b073b6f1d56843b888cb83f6a7d80a43
```

The status of the storage is as follows:

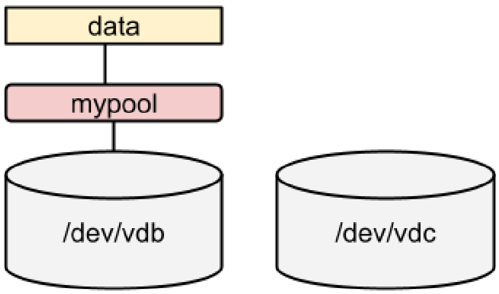


Figure 14.3 – Stratis filesystem created

Let's prepare to mount the filesystem. We need to add the following line in `/etc/fstab`:

```
/dev/stratis/mypool/data /srv/stratis-data      xfs
defaults,x-systemd.requires=stratisd.service        0 0
```

> **Important note**
> In order for a Stratis filesystem to be mounted correctly during boot, we shall add the `x-systemd.requires=stratisd.service` option so it is mounted after the `stratisd` service is started.

Now we can mount it:

```
[root@rhel8 ~]# mkdir /srv/stratis-data
[root@rhel8 ~]# mount /srv/stratis-data/
```

Let's now extend the pool:

```
[root@rhel8 ~]# stratis blockdev list mypool
Pool Name   Device Node   Physical Size   Tier
mypool      /dev/vdb              1 GiB   Data
[root@rhel8 ~]# stratis pool add-data mypool /dev/vdc
[root@rhel8 ~]# stratis blockdev list mypool
Pool Name   Device Node   Physical Size   Tier
mypool      /dev/vdb              1 GiB   Data
mypool      /dev/vdc              1 GiB   Data
```

As the underlying layer uses thin-pooling, we do not need to extend the filesystem. The storage is as follows:

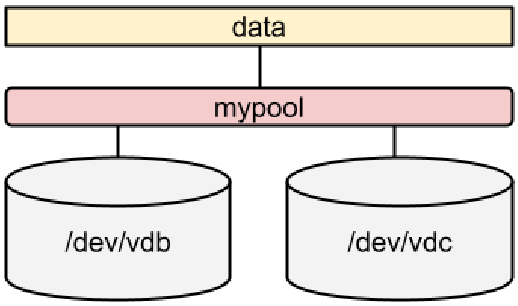


Figure 14.4 – Stratis pool extended

Time to use the `stratis snapshot` command to create a snapshot. Let's create some data and then snapshot it:

```
[root@rhel8 ~]# stratis filesystem
Pool Name   Name   Used      Created             Device
UUID
mypool      data   546 MiB   May 23 2021 19:54   /dev/stratis/
mypool/data   08af5d5782c54087a1fd4e9531ce4943
[root@rhel8 ~]# dd if=/dev/urandom of=/srv/stratis-data/file
bs=1M count=512
512+0 records in
512+0 records out
536870912 bytes (537 MB, 512 MiB) copied, 2,33188 s, 230 MB/s
[root@rhel8 ~]# stratis filesystem
Pool Name   Name   Used      Created             Device
```

```
UUID
mypool      data   966 MiB   May 23 2021 19:54   /dev/stratis/
mypool/data   08af5d5782c54087a1fd4e9531ce4943
[root@rhel8 ~]# stratis filesystem snapshot mypool data data-
snapshot1
[root@rhel8 ~]# stratis filesystem
Pool Name   Name             Used       Created
Device                                 UUID
mypool      data             1.03 GiB   May
23 2021 19:54   /dev/stratis/mypool/data
08af5d5782c54087a1fd4e9531ce4943
mypool      data-snapshot1   1.03 GiB   May 23
2021 19:56   /dev/stratis/mypool/data-snapshot1
a2ae4aab56c64f728b59d710b82fb682
```

> **Tip**
>
> To see the internal pieces of Stratis, you can run the `lsblk` command. With it, you will see the components used by Stratis in a tree: physical devices, allocations for metadata and data, pools, and filesystems. All of that is abstracted by Stratis.

With this, we have seen an overview of Stratis in order to cover the basics of its management. Remember that Stratis is in preview and therefore it should not be used in production systems.

Let's move on now to other advanced topics in storage management by reviewing data deduplication with VDO.

Preparing systems to use VDO

As mentioned earlier, VDO is a driver, specifically a Linux device-mapper driver, that uses two kernel modules:

- `kvdo`: This does data compression.
- `uds`: This is in charge of deduplication.

Regular storage devices such as local disks, **Redundant Array of Inexpensive Disks** (**RAID**), and so on are the final backend where data is stored; the VDO layer on top reduces disk usage via the following:

- The removal of zeroed blocks, only storing them in the metadata.
- Deduplication: Duplicate data blocks are referenced in the metadata but stored only once.
- Compression, using 4 KB data blocks with a lossless compression algorithm (LZ4: `https://lz4.github.io/lz4/`).

These techniques have been used in the past in other solutions, such as in thin-provisioned **VMs** that only kept the differences between VMs, but VDO makes this happen transparently.

Similar to thin-provisioning, VDO can mean faster data throughput, as data can be cached by the system controller and several services or even VMs can use that data without there being a need for additional disk reads to access it.

Let's install the required packages on our system in order to create VDO volumes by installing the `vdo` and `kmod-kvdo` packages:

```
dnf install vdo kmod-kvdo
```

Now, with the packages installed, we're ready to create our first volume in the next section.

Creating a VDO volume

To create a VDO device, we will make use of the loopback device we created in *Chapter 12*, *Managing Local Storage and Filesystems*, so we will check first whether it's mounted or not by executing this:

```
mount|grep loop
```

If no output is shown, we're set for creating our `vdo` volume on top of it with the following:

```
vdo create -n myvdo --device /dev/loop0 -force
```

The output is shown in the following screenshot:

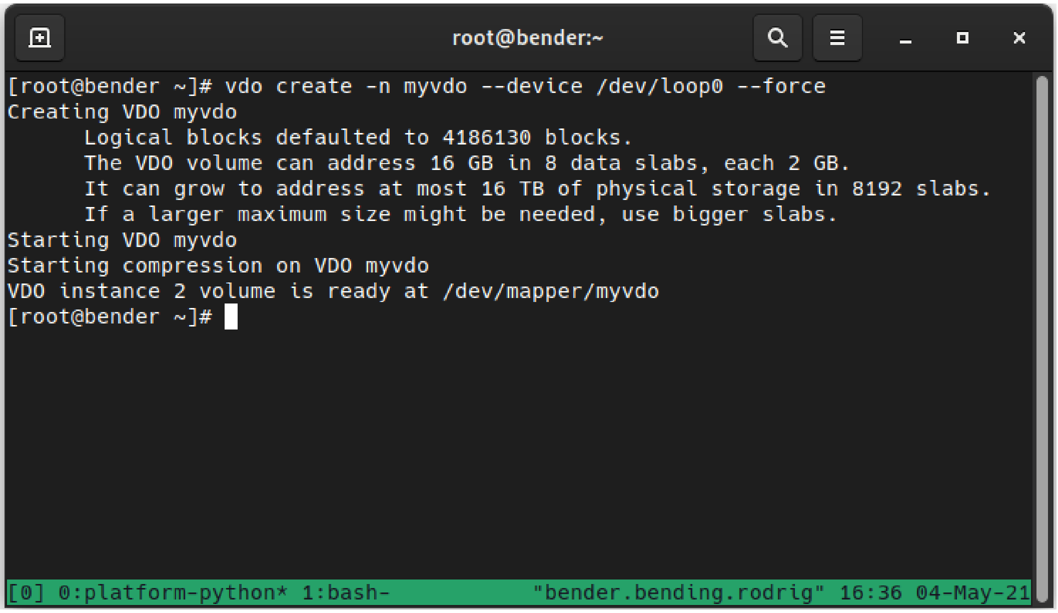


Figure 14.5 – vdo volume creation

Once the volume has been created, we can execute `vdo status` to get detailed information about the volume created, as seen in the following screenshot:

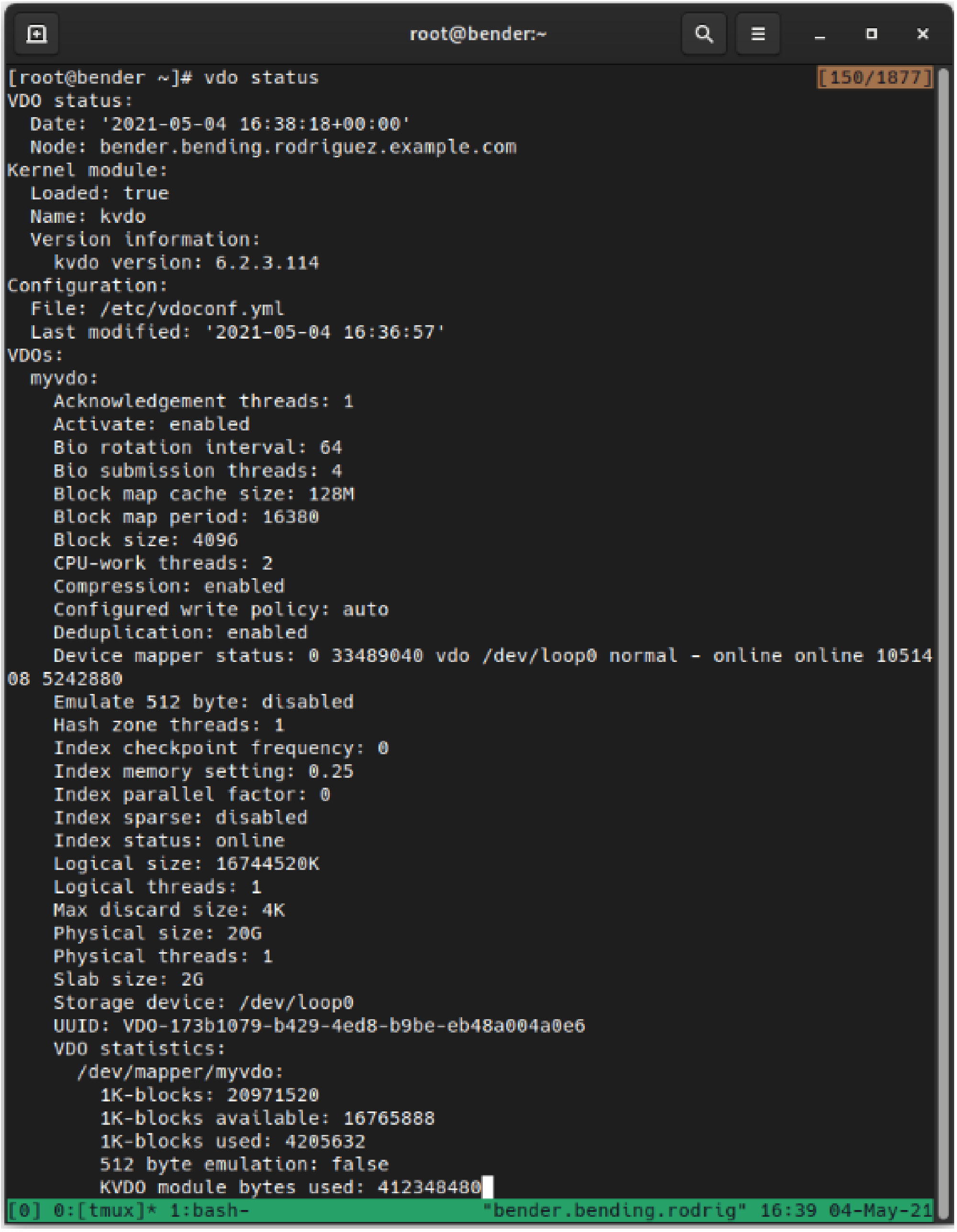


Figure 14.6 – Output of vdo status

As we can see, there's information about the `kvdo` version, the configuration file being used, and our volumes (size, compression status, and so on).

The new volume can now be seen via `/dev/mapper/myvdo` (the name we assigned with `-n`) and it's ready to be used.

We can execute `vdo status|egrep -i "compression|dedupli"` and get an output that looks as follows:

Figure 14.7 – Checking vdo status for compression and deduplication

This means that both compression and deduplication are enabled on our volume, so we're ready to test the functionality by adding it to an LVM volume in the next section.

Assigning a VDO volume to an LVM volume

In the previous section, we created a VDO volume, which will now become our **physical volume** (**PV**) when creating an LVM volume group and some logical volumes on top of it.

Let's create the PV by running the following sequence of commands:

1. `pvcreate /dev/mapper/myvdo`
2. `vgcreate myvdo /dev/mapper/myvdo`
3. `lvcreate -L 15G -n myvol myvdo`

At this point, our `/dev/myvdo/myvol` is ready to be formatted. Let's use the XFS filesystem:

```
mkfs.xfs /dev/myvdo/myvol
```

Once the filesystem has been created, let's put some data on it by mounting as follows:

```
mount /dev/myvdo/myvol /mnt
```

Now let's test the VDO volume in the next section.

Testing a VDO volume and reviewing the stats

In order to test deduplication and compression, we will test with a big file, such as the RHEL 8 KVM guest image available at `https://access.redhat.com/downloads/content/479/ver=/rhel---8/8.3/x86_64/product-software`.

Once downloaded, save it as `rhel-8.3-x86_64-kvm.qcow2` and copy it four times to our VDO volume:

```
cp rhel-8.3-x86_64-kvm.qcow2 /mnt/vm1.qcow2
cp rhel-8.3-x86_64-kvm.qcow2 /mnt/vm2.qcow2
cp rhel-8.3-x86_64-kvm.qcow2 /mnt/vm3.qcow2
cp rhel-8.3-x86_64-kvm.qcow2 /mnt/vm4.qcow2
```

This would be the typical case for a server holding VMs that start of the same base disk image, but do we see any improvement?

Let's execute `vdostats --human-readable` to verify the data. Note that the image downloaded is 1.4 GB, as reported by `ls –si`. The output obtained from `vdostats --human-readable` is as follows:

```
Device                    Size      Used Available Use% Space
saving%
/dev/mapper/myvdo        20.0G      5.2G     14.8G  25%
75%
```

The original volume (the loopback file) was 20 GB, so that's the size we can see, but the LVM volume we created was 15 GB, judging from the output, and we see that approximately only 1.2 GB has been consumed, even if we've got four files of 1.4 GB each.

The percentage is also very clear. We've saved 75% of the space (three files out of four are exact copies). If we make an additional copy, we will see that the percentage goes to 80% (1 out of 5 copies).

Let's check out one of the other approaches, by creating an empty file (filled with zeros):

```
[root@bender mnt]# dd if=/dev/zero of=emptyfile bs=16777216 count=1024
dd: error writing 'emptyfile': No space left on device
559+0 records in
558+0 records out
9361883136 bytes (9.4 GB, 8.7 GiB) copied, 97.0276 s, 96.5 MB/s
```

As we can see, we were able to write 9.4 GB before the disk completely filled, but let's check the vdo stats again with `vdostats --human-readable` as seen in the following screenshot:

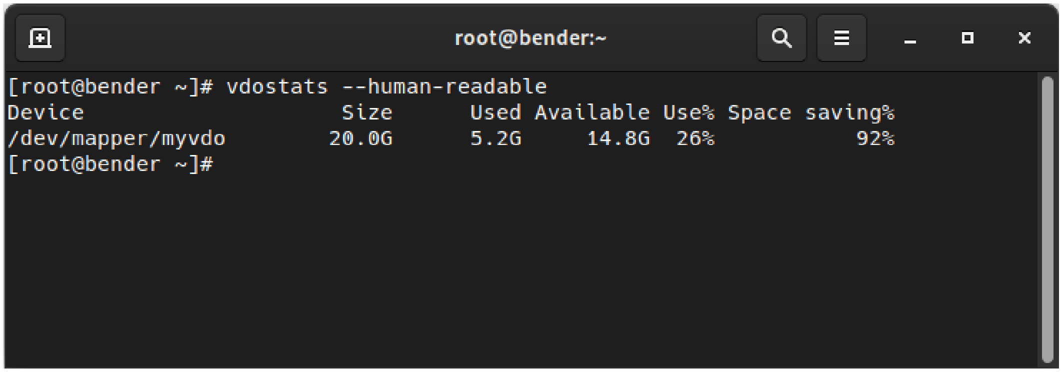


Figure 14.8 – Checking the vdostats output

As we can see, we still have 14.8 GB available and we've increased the disk space saved from 80% to 92%, because this big file is empty.

Wait – how, if we're using deduplication and compression, have we filled the volume if 92% of it has been saved?

As we did not indicate the logical size of the VDO volume, it set by default a 1:1 ratio with the underlying device. This is the safest approach, but we're not taking real advantage of the compression and deduplication beyond performance.

To make the most of the optimizations, we can create a bigger logical drive on top of the volume we have. For example, if after a long period of time we're pretty sure that the disk optimizations might be similar, we can grow the logical size with the following command:

```
vdo growLogical --name=myvdo --vdoLogicalSize=30G
```

This will, of course, not increase the available size, as we defined a PV with a volume group and a logical volume on top. So, we will also need to extend it by executing these commands:

1. `pvresize /dev/mapper/myvdo`
2. `lvresize -L +14G /dev/myvdo/myvol`
3. `xfs_growfs /mnt`

With this, we have extended the physical volume, increased the size of the logical volume, and extended the filesystem, so the space is now available to be used.

If we now execute `df|grep vdo`, we will see something like this:

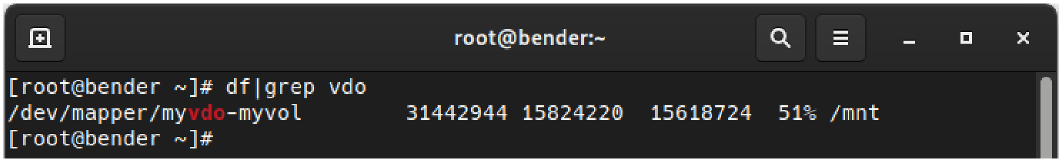


Figure 14.9 – Disk space availability after resizing the volume

From this point on, we must be extremely careful, as our real usage of disk space might not be as optimized in terms of possible compression as it was before, resulting in failures in writes. It is then required to monitor available disk space as well as VDO status to ensure that we're not attempting to use more space than is available, for example, if the files stored can't be compressed or deduplicated at the same ratio.

> **Important note**
>
> It's tempting to set a really big logical volume out of our real physical disk space, but we should plan ahead and think about avoiding future problems, such as the likelihood of compression ratios not being as high as our optimism. Adequately profiling the actual data being stored and the typical compression ratios for it can give us a better idea of what is a safe approach to be used while we continue to actively monitor disk usage evolution, both for the logical volume and for the physical one.

Long ago, when disk space was really expensive (and hard drives were 80 MB in total), it became very popular to use tools that allowed an *increase* in disk space by using a transparent layer of compression that could make some estimations and report bigger space; but in reality, we know that content such as images and movies don't compress as well as other document formats such as text files. Some document formats, such as the ones used by LibreOffice, are already compressed files, so no extra compression benefits are gained.

But this changes when we speak about VMs, where the base for each one is more or less equal (based on company policies and standards) and are deployed via cloning disk images and later performing small customization, but in essence, sharing most of the disk contents.

> **Tip**
> In general, bear in mind that optimizations really just mean trade-offs. In the case of tuned profiles, you're adjusting throughput for latency, and in our case, you're trading CPU and memory resources for disk availability. The only way to tell whether something's a worthwhile trade-off is to implement it and see how it performs, look at the benefits gained, and then continue to monitor performance over time.

Summary

In this chapter, we have learned about VDO and Stratis. We've looked at simple ways to manage storage, how to save disk space transparently, and how to gain some throughput in the process.

With Stratis, we have created a pool with two disks and assigned it to a mountpoint. It takes fewer steps than doing so with LVM, but on the other hand, we have less control over what we are doing. In any case, we learned how to use this preview technology in RHEL 8.

With VDO, we used the volume we created to define an LVM PV and, on top of it, a volume group and a logical volume that we've formatted using the knowledge gained in previous chapters to store a VM disk image several times, to simulate a scenario where several VMs are started from the same base.

We also learned how to check the VDO optimizations and the amount of disk saved.

Now, we're ready to use Stratis instead of LVM to group and distribute storage (though not for production). We can also implement VDO for our servers to start optimizing disk usage.

In the next chapter, we will learn about the boot process.

15
Understanding the Boot Process

The boot process is what happens between the moment you power on a machine (physical or virtual) and when the operating system is completely loaded.

Like in many good video games, there are three stages for it: the initial startup performed by the hardware (again physical or virtual), the load of the initial stages of the operating system, and then the mechanism that helps run the required services in the system. We will review the three stages in this chapter, and we will also add tips and tricks in order to intervene in a system and perform rescue operations.

The sections in this chapter in which we will cover all these topics are as follows:

- Understanding the boot process – BIOS and UEFI booting
- Working with GRUB, the bootloader, and initrd system images
- Managing the boot sequence with systemd
- Intervening in the boot process to gain access to a system

It is very likely that you will not need to make many changes in the first two stages of the boot process, but these are the points that could be extremely helpful in cases of emergency, forensics, or major failures. That's why it's important to go through them carefully.

The third stage, the one managed by **systemd**, is where more actions and changes will be performed in order to manage which services are running by default in the system. We have already seen examples of most of the tasks to be performed in previous chapters; however, in this one we will provide a comprehensive review.

Let's get started with stage one.

Understanding the boot process – BIOS and UEFI booting

Computers have hardware-embedded software controllers, also called **firmware**, that let you manage the very lowest layers of the hardware. This firmware is what performs the first recognition of what hardware is available in the system and what hardware features are enabled (such as **pre-boot network execution**, called **PXE**).

In the architecture known as **PC** (for **Personal Computer**), also referred to as x86, which Intel and IBM popularized, the embedded firmware is referred to as **BIOS**, which stands for **Basic Input and Output System**.

The BIOS boot process, with Linux, takes the following steps:

1. The machine is powered on and BIOS firmware is loaded.
2. The firmware initializes devices such as keyboard, mouse, storage, and other peripherals.
3. The firmware reads the configuration, including the boot order, specifying which storage device is the one to continue the boot process with.
4. Once the storage device is selected, BIOS will load the **Master Boot Record** (**MBR**) on it, which will enable running the **operating system loader**. In RHEL, the operating system loader is called **Grand Unified Bootloader** (**GRUB**).
5. GRUB loads the configuration and the **operating system kernel** and **initial RAM disk** as specified in its configuration. In **Red Hat Enterprise Linux** (**RHEL**) the kernel is stored in a file called `vmlinuz`, and the initial boot image in a file called `initrd`. All of the GRUB configuration `vmlinuz` and `initrd` files are stored in the `/boot` partition.
6. The initial boot image enables loading the first process of the system, also referred to as `init`, which in RHEL8 is **systemd**.
7. *systemd* loads the rest of the operating system.

For this process to happen, the disk has to have an MBR partition table, and the partition assigned to `/boot` has to be marked as bootable.

> **Tip**
> The MBR partition table format is very limited, allowing only four primary partitions and using extensions such as extended partitions to overcome this limit. It is not recommended to use this type of partition unless it is completely required.

The UEFI boot process is very similar to the BIOS boot process. **UEFI** stands for **Unified Extensible Firmware Interface**. The main difference in the boot sequence is that UEFI can access and read disk partitions directly. The flow for it is as follows:

1. The machine is powered on and the UEFI firmware is loaded.
2. The firmware initializes devices such as keyboard, mouse, storage, and other peripherals.
3. The firmware reads the configuration, in which it is specified which storage device and bootable partition to continue the boot process with (UEFI does not need an MBR to boot).
4. Once the storage device is selected, the partitions on it are read from the **GUID Partition Table** (**GPT**). The first partition with VFAT format is accessed. Then EFI Boot Loader is loaded and run. The EFI Boot Loader in RHEL lives in the `/boot/efi` partition, and it continues to load GRUB.
5. GRUB then loads the **operating system kernel**, which in RHEL is stored in a file called `vmlinuz`, and the **initial boot image**, which is stored in a file called `initrd`. GRUB configuration `vmlinuz` and `initrd` files are stored in the `/boot` partition.
6. The initial boot image enables loading the first process of the system, also referred to as `init`, which in RHEL8 is **systemd**.
7. *systemd* loads the rest of the operating system.

UEFI has several advantages over BIOS, enabling more complete pre-boot environments and other capabilities such as secure boot and support for GPT partitions that can go beyond the 2 TB limit that MBR partitions have.

The installer will take care of creating the boot and, if needed, UEFI partitions and binaries.

The part of pre-boot that needs to be known for the Red Hat Certified System Administrator certification is how to load the operating system loader from it. Through BIOS or UEFI we can select from which storage device the operating system will load and move to the next phase. Let's go to this next phase in the next section.

Working with GRUB, the bootloader, and initrd system images

Once the pre-boot execution is completed, the system will be running the GRUB bootloader.

GRUB has the mission to load the main file of an operating system, the **kernel**, pass parameters and options to it, and load the initial RAM disk, also known as **initrd**.

GRUB can be installed by using the `grub2-install` command. We will need to know which disk device will be used to boot, in this case, `/dev/vda`:

```
[root@rhel8 ~]# grub2-install /dev/vda
Installing for i386-pc platform.
Installation finished. No error reported.
```

> **Important note**
>
> You should point `grub-install` to the disk you will use to boot the system, the same one that you configured in the BIOS/UEFI to boot from.

This is intended to be used to manually rebuild a system or to fix a broken boot.

GRUB files are stored in `/boot/grub2`. The main configuration file is `/boot/grub2/grub.cfg`; however, if you take a close look at this file you will see the following header:

```
[root@rhel8 ~]# head -n 6 /boot/grub2/grub.cfg
#
# DO NOT EDIT THIS FILE
#
# It is automatically generated by grub2-mkconfig using
templates
# from /etc/grub.d and settings from /etc/default/grub
#
```

As you can see, this file is automatically generated and, therefore, not intended to be edited manually. How do we make changes to it then? There are two ways to do so:

- The first way is by following the instructions mentioned in the `grub.cfg` file. This means editing the `/etc/default/grub` file and/or the contents in the `/etc/grub.d/` directory, and then regenerating the GRUB configuration by running `grub2-mkconfig`.
- The second way is by using the `grubby` command-line tool.

> **Important note**
>
> In RHEL, when there is a new version of the kernel, it is not updated, but a new kernel is installed alongside the previous one, adding a new entry in GRUB. In this way, there is an easy way to roll back to a previous working kernel in case it's needed. During the installation, a new updated `initrd` is created for the new kernel.

Let's take a look at the current kernel configuration with `grubby`. The `--default-kernel` option will show which kernel file is loaded by default:

```
[root@rhel8 ~]# grubby --default-kernel
/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64
```

The `--default-title` option will show the name used during boot:

```
[root@rhel8 ~]# grubby --default-title
Red Hat Enterprise Linux (4.18.0-240.15.1.el8_3.x86_64) 8.3
(Ootpa)
```

We can see more information for the default kernel by using the `--info` option:

```
[root@rhel8 ~]# grubby --info=/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet $tuned_params"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-240.15.1.el8_3.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-240.15.1.el8_3.x86_64) 8.3 (Ootpa)"
id="21e418ac989a4b0c8afb156418393409-4.18.0-240.15.1.el8_3.x86_64"
```

We can see the options passed to GRUB:

- `index`: Shows the index number of the entry
- `kernel`: The file containing the kernel that will be loaded to run the core of the operating system
- `root`: The partition, or logical volume, that will be assigned to the root / directory and mounted
- `initrd`: File containing the RAM disk to perform the initial part of the boot process
- `title`: Descriptive title to be shown to the user during the boot process
- `id`: Identifier of the boot entry

Tip

You may want to run the `grubby` command to obtain the information for the kernel configured as default. To do so, you can do it by running the following command: `grubby --info=$(grubby --default-kernel)`.

Let's make the boot process more verbose by removing the `quiet` and `rhbg` arguments passed to the kernel:

```
[root@rhel8 ~]# grubby --remove-args="rhgb quiet" \
--update-kernel=/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64
[root@rhel8 ~]# grubby \
--info=/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap $tuned_params"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-240.15.1.el8_3.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-240.15.1.el8_3.x86_64) 8.3 (Ootpa)"
id="21e418ac989a4b0c8afb156418393409-4.18.0-240.15.1.el8_3.x86_64"
```

Let's test it by rebooting the machine with the `systemctl reboot` command. This is an example output:

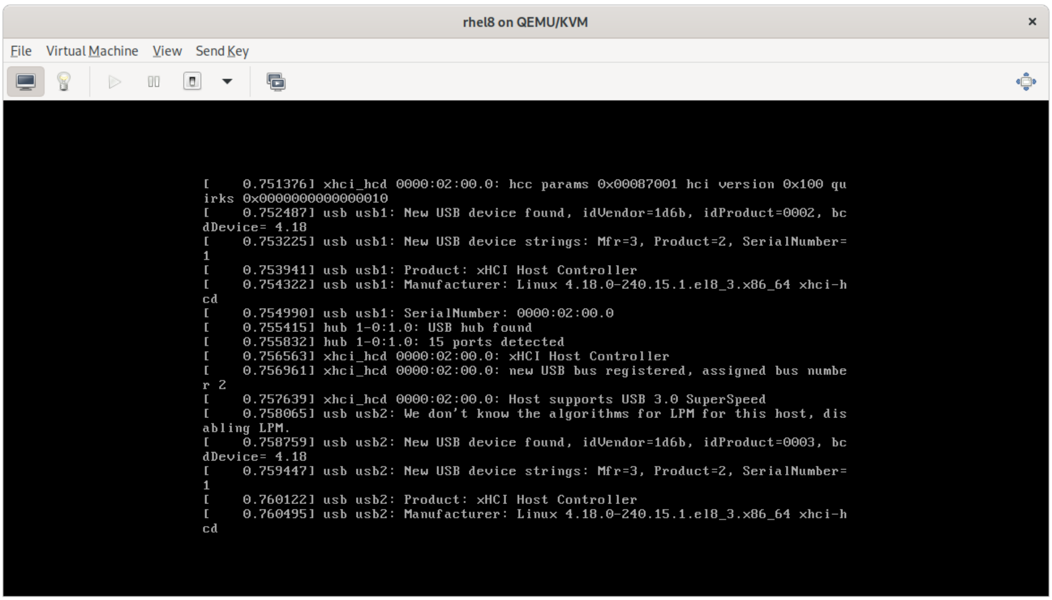


Figure 15.1 – Verbose boot

In a normal boot, this may not be very useful as it goes too fast. However, if there are issues, it can help debug the situation from the console. To review these messages after booting, the `dmesg` command can be used:

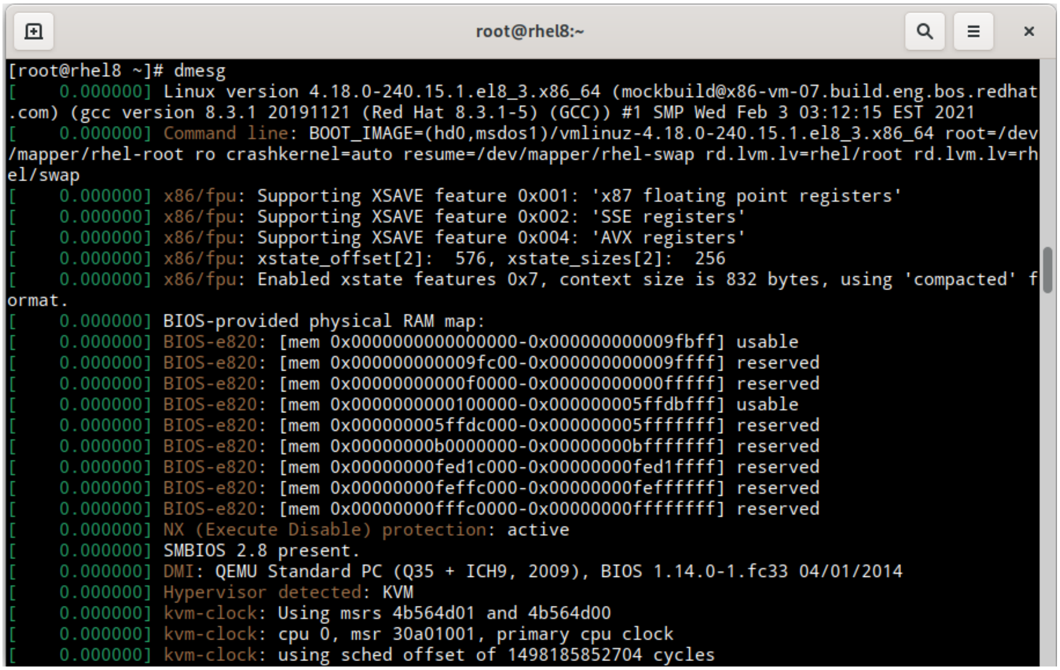


Figure 15.2 – Output of the dmesg command

We can add an argument to the kernel by using the `--args` option. Let's add the `quiet` option again:

```
[root@rhel8 ~]# grubby --args="quiet" \
--update-kernel=/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64
[root@rhel8 ~]# grubby \
--info=/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64
index=0
kernel="/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.
lv=rhel/root rd.lvm.lv=rhel/swap $tuned_params quiet"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-240.15.1.el8_3.x86_64.img
$tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-240.15.1.el8_3.x86_64)
8.3 (Ootpa)"
```

```
id="21e418ac989a4b0c8afb156418393409-4.18.0-240.15.1.el8_3.
x86_64"
```

> **Important note**
>
> The `--info` and `--update-kernel` options accept the ALL option to review or do actions to all the configured kernels.

If any administration task requires us to change kernel parameters, now we know how to do it. Let's move to the next section of the boot process, `initrd`.

The **initrd** file, or **initial RAM disk**, contains a minimal system that is used to prepare the system to start. We found it in the previous configuration as `/boot/initramfs-4.18.0-240.15.1.el8_3.x86_64.img`. It can be regenerated by using the `dracut` command. Let's see an example of how to rebuild the current `initrd` file:

```
[root@rhel8 ~]# dracut --force --verbose
dracut: Executing: /usr/bin/dracut --force --verbose
dracut: dracut module 'busybox' will not be installed, because
command 'busybox' could not be found!
[omitted]
dracut: *** Including module: shutdown ***
dracut: *** Including modules done ***
dracut: *** Installing kernel module dependencies ***
dracut: *** Installing kernel module dependencies done ***
dracut: *** Resolving executable dependencies ***
dracut: *** Resolving executable dependencies done***
dracut: *** Hardlinking files ***
dracut: *** Hardlinking files done ***
dracut: *** Generating early-microcode cpio image ***
dracut: *** Constructing GenuineIntel.bin ****
dracut: *** Constructing GenuineIntel.bin ****
dracut: *** Store current command line parameters ***
dracut: *** Stripping files ***
dracut: *** Stripping files done ***
dracut: *** Creating image file '/boot/initramfs-4.18.0-
240.15.1.el8_3.x86_64.img' ***
dracut: *** Creating initramfs image file '/boot/initramfs-
4.18.0-240.15.1.el8_3.x86_64.img' done ***
```

We can see in the previous output what kernel modules and files, required for early access, are included in the `initrd` file. This step is useful when our `initrd` file is corrupted and also when restoring a system from a backup, if done in different hardware, to include the proper storage drivers.

> **Tip**
>
> Check the manual page of `dracut` to learn more on options to create `initrd` files. There's a Red Hat knowledge base article to unpack `initrd`, an interesting exercise to learn more: `https://access.redhat.com/solutions/24029`.

We have learned the very basics of the early stages of the boot process in order to be able to start troubleshooting boot issues, as required to become an RHCSA. This advanced topic could be covered in an entire book, but very little of it will be used in your daily tasks as a system administrator. That's why we only included the required aspects of it. We will include a specific use case in the last section of this chapter called *Intervening in the boot process to gain access to a system* and fix disk issues. Let's move on to the next topic on how services are managed in RHEL with **systemd**.

Managing the boot sequence with systemd

We have already learned how the firmware of the system will take care of pointing at a disk to run the operating system loader, which in RHEL is GRUB.

GRUB will load the kernel and initrd to prepare the system to start. Then it's time to start the first process of the system, also referred to as process 1 or PID 1 (**PID** stands for **process identifier**). This process has to take care of loading all the required services in the system efficiently. In RHEL8, the PID 1 is run by **systemd**.

In *Chapter 4*, *Tools for Regular Operations*, we described services and targets management with systemd. Let's review in this chapter its interactions with the boot sequence.

The first two things related to the boot sequence that we can do with **systemd** are to reboot the system and to power it off. We will do it with the `systemctl` tool:

```
[root@rhel8 ~]# systemctl reboot
```

We will see that the system will reboot. We can check how long the system has been running with the `uptime` command:

```
[root@rhel8 ~]# uptime
11:11:39 up 0 min,  1 user,  load average: 0,62, 0,13, 0,04
```

Now it's time to check `poweroff`. Before doing so, remember that after running this command you will need to have a way to power on the machine again. Once we are aware of the process we are going to follow, let's run it:

```
[root@rhel8 ~]# systemctl poweroff
```

Now I shall power my machine on again.

There is a command that will stop the system but without sending the signal to power the machine off, which is `systemctl halt`. The cases where this can be used are rare; however, it's good to know that it exists and what it does.

> **Important note**
>
> The previously shown commands can be abbreviated to `reboot` and `poweroff`. If you check the file in `/usr/sbin/poweroff` you will see that it is a symbolic link pointing to `systemctl`.

In *Chapter 4*, *Tools for Regular Operations*, we also reviewed how to set a default **systemd target** with `systemctl`. However, we can override the default configuration during boot time by passing the `systemd.unit` parameter to the kernel. We can do that using `grubby`:

```
[root@rhel8 ~]# systemctl get-default
multi-user.target
[root@rhel8 ~]# grubby --args="systemd.unit=emergency.target"
--update-kernel=/boot/vmlinuz-4.18.0-240.15.1.el8_3.x86_64
[root@rhel8 ~]# systemctl reboot
```

Now the system is rebooting. The `systemd.unit=emergency.target` parameter has been passed by **GRUB** to the **kernel**, and from the **kernel** to **systemd**, which, in turn, will ignore the default configuration and load the services required for the **emergency target**.

Now the system is started in emergency mode and is waiting for the root password to give you control:

Figure 15.3 – RHEL system booted in emergency mode

In emergency mode, there is no network configured and no other process running. You can make changes to the system knowing that no other user is accessing it. Also, only the / filesystem is mounted in read-only mode.

If a filesystem in the system is broken, this would be a good way to check it without any service accessing it. Let's try it with the command for checking the filesystem, which is called `fsck`:

```
[root@rhel8 ~]# fsck /boot
fsck from util-linux 2.32.1
If you wish to check the consistency of an XFS filesystem or
repair a damaged filesystem, see xfs_repair(8).
```

The filesystem is OK. We could run `xfs_repair` on it, as it is an `xfs` filesystem (`fsck` detects the filesystem used) if it had issues that required a fix.

At this point we may be thinking, how can we make changes to the root filesystem if it's already mounted as read-only at `/`? The process starts by remounting the `/` filesystem as read-write:

```
[root@rhel8 ~]# mount -o remount -o rw /
```

Remember, you can access the manual pages for the command by running `man mount`. Now our root filesystem is mounted in `/` as read-write. We also need to mount `/boot`, so let's do it:

```
[root@rhel8 ~]# mount /boot
```

With the `/boot` mounted, let's do some admin tasks, such as removing the arguments we have used in GRUB:

```
[root@rhel8 ~]# grubby --remove-args="systemd.unit=emergency.
target" --update-kernel=/boot/vmlinuz-4.18.0-240.15.1.el8_3.
x86_64
[root@rhel8 ~]# reboot
```

And we are back to a regular boot in the system. This may not be a practical way to go to emergency mode in Linux, but it shows how to pass parameters to systemd at boot.

> **Tip**
>
> There is `rescue.target` that loads more services and makes the process somewhat easier. It does so by waiting for `sysinit.target` to complete, something that the emergency target does not do. A good exercise would be to repeat the previous sequence with `rescue.target`.

We will see in the coming section how to make this change, and similar ones, for a one-time boot only and more easily during the GRUB boot sequence, and in a way in which no password is required.

Intervening in the boot process to gain access to a system

Sometimes you need to intervene in a system that was handed over in which you do not have the password for the `root` user. This is an exercise that, though it sounds like an emergency situation, is more frequent than you would expect.

> **Important note**
>
> The boot sequence must not have any disk encrypted for it to work or you will need the password for the encrypted volumes.

The way to perform this procedure starts by stopping the boot process during the GRUB menu. This means we need to restart the system. Once the BIOS/UEFI checks have finished, the system will load GRUB. There we can stop the count by pressing the down or up arrow key while it's waiting for the selection of the kernel, as in the following screenshot:

```
Red Hat Enterprise Linux (4.18.0-240.15.1.el8_3.x86_64) 8.3 (Ootpa)
Red Hat Enterprise Linux (4.18.0-240.10.1.el8_3.x86_64) 8.3 (Ootpa)
Red Hat Enterprise Linux (4.18.0-240.8.1.el8_3.x86_64) 8.3 (Ootpa)
Red Hat Enterprise Linux (0-rescue-21e418ac989a4b0c8afb156418393409)
```

Figure 15.4 – GRUB menu to select the kernel

We move back to the first entry. Then we read the bottom of the screen where we find the instructions to edit the boot line:

```
Use the ↑ and ↓ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
```

Figure 15.5 – GRUB menu to select the kernel

If we press the *E* key, we will be able to edit the boot line selected in the menu. We will see the following five lines:

```
load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-240.15.1.el8_3.x86_64 root=/dev/mapper/rhel-root \
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv\
=rhel/swap
initrd  ($root)/initramfs-4.18.0-240.15.1.el8_3.x86_64.img $tuned_initrd
```

Figure 15.6 – GRUB menu to select the kernel

The first three lines with `load_video`, `set gfx_payload=keep`, and `insmod gzio` are setting options for GRUB. The next two options are the ones that are important. Let's review them:

- `linux`: Defines the kernel to be loaded and passes parameters to it
- `initrd`: Defines where to load the initrd and if there are any options for it

> **Tip**
>
> Please note that the `linux` line is so long that it is wrapped, as we can see by the \ symbols, which mean that the line continues below.

We shall now go to the end of the `linux` line and add the `rd.break` option, as in the following screenshot:

```
linux ($root)/vmlinuz-4.18.0-240.15.1.el8_3.x86_64 root=/dev/mapper/rhel-root \
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv\
=rhel/swap rd.break_
```

Figure 15.7 – linux kernel line edited with the rd.break option

To boot the edited line, we only need to press *Ctrl* + *X*. The `rd.break` option stops the boot process before the initrd is loaded. The situation now is the following:

- A single shell is loaded.
- The current root filesystem mounted on `/` is a minimal one with basic administration commands.
- The target root filesystem is mounted in `/sysroot` as read-only (instead of on `/`).
- No other filesystems are mounted.
- SELinux is not loaded.

The first thing we can do now is switch to the real, on-disk root filesystem with `chroot`:

```
switch_root:/# chroot /sysroot
sh-4.4#
```

Now our root filesystem is properly mounted, but read-only. Let's change that in the same way as we did in the previous section:

```
sh-4.4# mount -o remount -o rw /
```

Now we need to change the root user password with the `passwd` command:

```
sh-4.4# passwd
Changing password for user root
New password:
Retype new password:
passwd: all authentication tokens updated successfully
```

The password for the root user is now changed and the `/etc/shadow` file has been updated. However, it was modified without SELinux enabled, and therefore it could cause an issue in the next boot. To avoid that, there is a mechanism to fix the SELinux labels during the next boot. This mechanism consists of creating the `/.autorelabel` hidden empty file and then rebooting the system:

```
sh-4.4# touch /.autorelabel
```

Once the file is created, it's time to reboot it to apply the SELinux changes. In this status, the machine may require forcing the power off and then powering on. During the next boot, we will see the SELinux autorelabel happening:

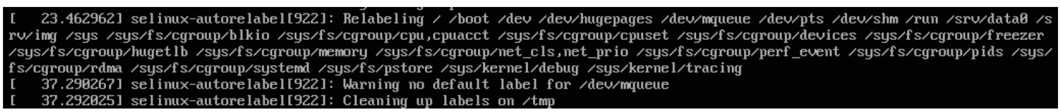


Figure 15.8 – SELinux autorelabel during boot

Now we can log in with the root user and its new password.

Summary

We have reviewed the boot sequence in this chapter. As you have seen, it is not long, but it is complex and it is also very important as no system can run if it cannot boot. We have learned the main differences between a BIOS-enabled system and a UEFI one, which enables some capabilities but also has its own requirements. We have also learned about GRUB and its important role in the boot sequence, how to modify entries permanently with `grubby`, and how to make a one-time modification. We now know the main files to boot, such as the kernel, `vmlinuz`, and the initial RAM disk, `initrd`.

This chapter also showed us how to start in emergency and rescue modes, as well as how to intervene in a system to reset the root password.

We are now more prepared to handle any difficult situation in our systems with these tools and procedures. It's time to dive deeper and learn about kernel tuning and performance profiles in the next chapter.

16
Kernel Tuning and Managing Performance Profiles with tuned

As described occasionally in previous chapters, each system performance profile must be adapted to the expected usage for our system.

Kernel tuning plays a key role in this optimization, and we will be exploring this further in this chapter in the following sections:

- Identifying processes, checking memory usage, and killing processes
- Adjusting kernel scheduling parameters to better manage processes
- Installing `tuned` and managing tuning profiles
- Creating a custom `tuned` profile

By the end of this chapter, you will know how kernel tuning is applied, how quick profiles can be used via `tuned` to suit general use cases for different system roles, and how to further extend those customizations for your servers.

Additionally, identifying processes that have become a resource hog and how to terminate them and or prioritize them will be a useful way of getting a bit more juice out of our hardware when most needed.

Let's get hands-on and learn about these topics!

Technical requirements

You can continue the practice of using the **virtual machine** (**VM**) created at the beginning of this book in *Chapter 1*, *Installing RHEL8*. Any additional packages required for this chapter will be indicated alongside the text.

Identifying processes, checking memory usage, and killing processes

A process is a program that runs on our system—it might be a user logged in via **Secure Shell** (**SSH**) that has a bash terminal process running, or even the portion of the SSH daemon listening and replying to remote connections, or it could be a program such as a mail client, a file manager, and so on being executed.

Of course, processes take up resources in our system: memory, **Central Processing Unit** (**CPU**), disk, and so on. Identifying or locating ones that might be misbehaving is a key task for system administrators.

Some of the basics were already covered in *Chapter 4*, *Tools for Regular Operations*, but it would be a good idea to have a refresher on these before continuing; however, we will be showing and using some of those tools in the context of performance tuning here, such as—for example—the `top` command, which allows us to see processes and sort lists based on CPU usage, memory usage, and so on. (Check the output of `man top` for a refresher on how to change the sorting criteria.)

One parameter to watch while checking system performance is the load average, which is a moving average made by the processes ready to run or waiting for **input/output** (**I/O**) to complete. It's composed of three values—`1`, `5`, and `15` minutes—and gives an idea of whether a load is increasing or lowering. A rule of thumb is that if a load average is below 1, there is no resource saturation.

The load average is shown with many other tools, such as the aforementioned `top`, or with `uptime` or `w`, and so on.

If the system-load average is growing, CPU or memory usage is spiking, and if some processes are listed there, it will be easier to locate. If the load average is also high and increasing, it might be possible that the I/O operations are increasing it. It is possible to install the `iotop` package, which provides the `iotop` command to monitor disk activity. When executed, it will show the processes in a system and the disk activity: reads, writes, and swaps that might give us some more hints about where to look.

Once a process has been identified as taking too many resources, we can send a **signal** to control it.

A signal list can be obtained with the `kill -l` command, as illustrated in the following screenshot:

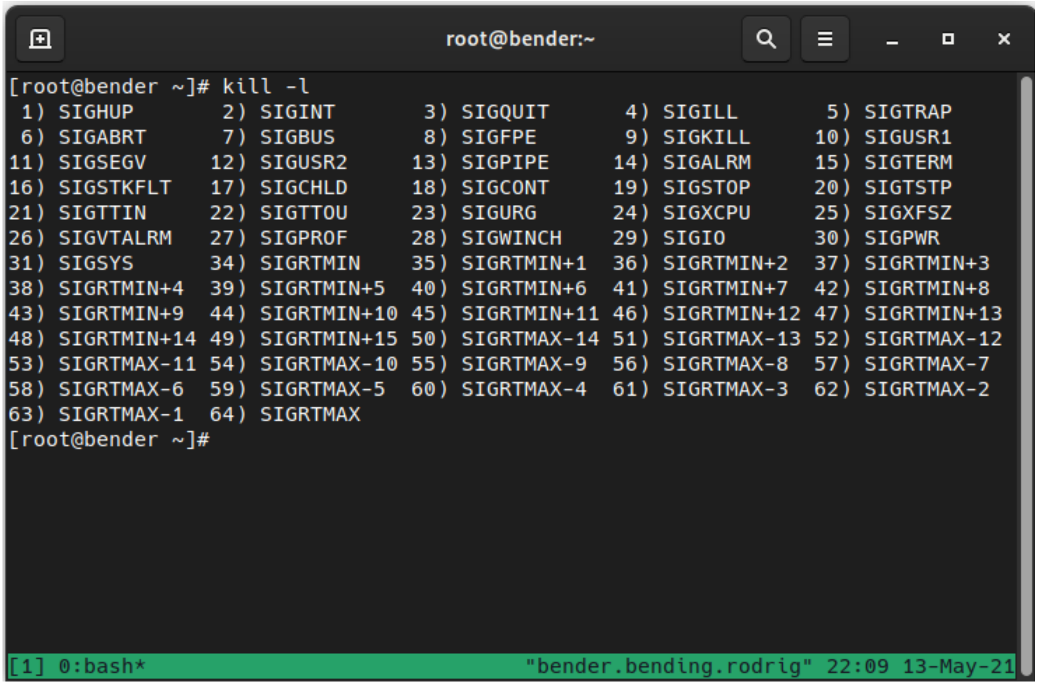


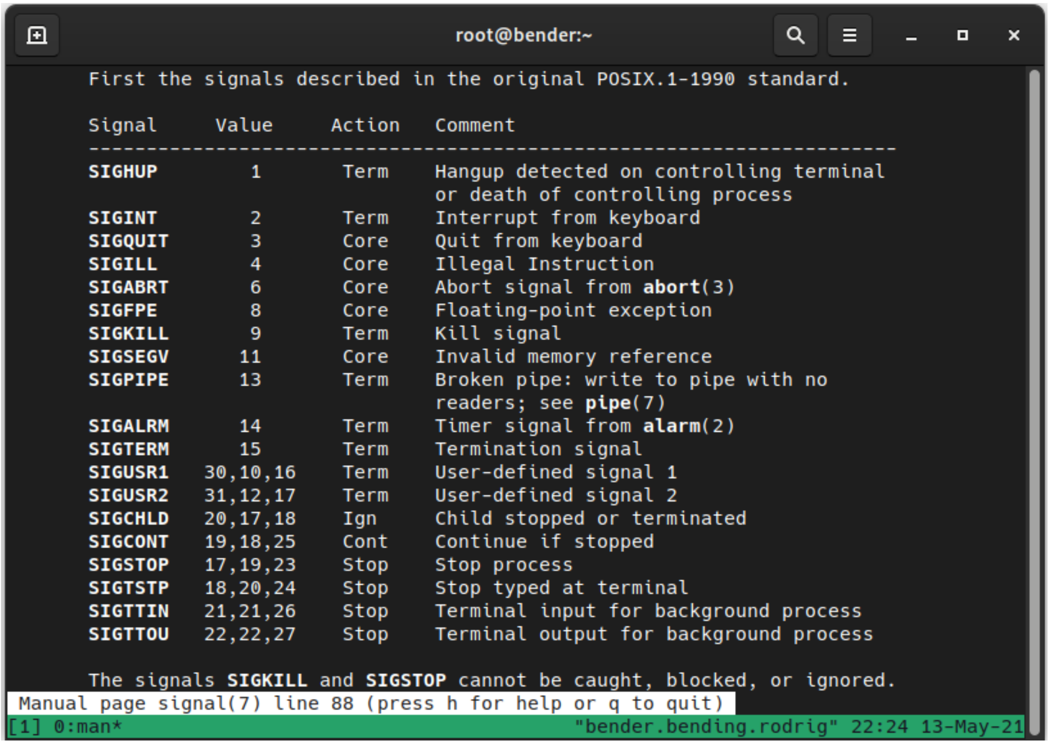
Figure 16.1 – Available signals to send to processes

Note that each signal contains a number and a name—both can be used to send the signal to the process via its **process identifier** (**PID**).

Let's review the most common ones, as follows:

Signal	ID	Usage
`KILL`	`9`	This is the most common signal, used to abruptly terminate a running process.
`TERM`	`15`	Notifies the process that it must exit so that the process can perform some cleanup operations before exiting.
`USR1`	`10`	Some programs use this signal to get notified about a configuration reload request without exiting the process itself.
`HUP`	`1`	Signal sent to processes when the controlling terminal or process ends.

From the list shown in *Figure 16.1*, it's important to know that each signal has a **disposition**—that is, once a signal is sent, the process must, according to the signal received, perform one of the following actions: terminate, ignore the signal, perform a core dump, stop the process, or continue the process if it was stopped. The exact details about each signal can be checked at `man 7 signal`, as illustrated in the following screenshot:



```
root@bender:~
       First the signals described in the original POSIX.1-1990 standard.

       Signal     Value     Action   Comment
       ──────────────────────────────────────────────────────────────────────
       SIGHUP        1       Term    Hangup detected on controlling terminal
                                     or death of controlling process
       SIGINT        2       Term    Interrupt from keyboard
       SIGQUIT       3       Core    Quit from keyboard
       SIGILL        4       Core    Illegal Instruction
       SIGABRT       6       Core    Abort signal from abort(3)
       SIGFPE        8       Core    Floating-point exception
       SIGKILL       9       Term    Kill signal
       SIGSEGV      11       Core    Invalid memory reference
       SIGPIPE      13       Term    Broken pipe: write to pipe with no
                                     readers; see pipe(7)
       SIGALRM      14       Term    Timer signal from alarm(2)
       SIGTERM      15       Term    Termination signal
       SIGUSR1   30,10,16    Term    User-defined signal 1
       SIGUSR2   31,12,17    Term    User-defined signal 2
       SIGCHLD   20,17,18    Ign     Child stopped or terminated
       SIGCONT   19,18,25    Cont    Continue if stopped
       SIGSTOP   17,19,23    Stop    Stop process
       SIGTSTP   18,20,24    Stop    Stop typed at terminal
       SIGTTIN   21,21,26    Stop    Terminal input for background process
       SIGTTOU   22,22,27    Stop    Terminal output for background process

       The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.
 Manual page signal(7) line 88 (press h for help or q to quit)
[1] 0:man*                                 "bender.bending.rodrig" 22:24 13-May-21
```

Figure 16.2 – Listing of signals, number equivalent, disposition (action), and behavior (man 7 signal)

One of the most typical usages when arriving at this point is to terminate processes that are misbehaving, so a combination of locating the process, obtaining the PID, and sending a signal to it is a very common task... so common that there are even tools that allow you to combine these stages in one command.

For example, we can compare `ps aux|grep -i chrome|grep -v grep|awk '{print $2}'|xargs kill -9` with `pkill -9 -f chrome`: both will perform the same action, search processes named `chrome`, and send signal `9` (kill) to them.

Of course, even a user logging in is a process in the system (running SSH or the shell, and more); we can find the processes started by our target user via a similar construction (with `ps`, `grep`, and others) or with `pgrep` options such as `pgrep -l -u user`.

Bear in mind that, as the signals indicate, it's better to send a `TERM` signal to allow the process to run its internal cleanup steps before exiting, as directly killing them might result in leftovers in our system.

One interesting command that was widely used before terminal multiplexers such as `tmux` or `screen` became commonplace was `nohup`, which was prepended to commands that would last longer—for example, downloading a big file. This command captured the terminal hangout signal, allowing the process executed to continue execution, storing the output in a `nohup.out` file that could later be checked.

For example, to download the latest **Red Hat Enterprise Linux** (**RHEL**) **Image Standard Optical** (**ISO**) file from the Customer Portal, select one release—for example, 8.4—and once logged in at `https://access.redhat.com/downloads/content/479/ver=/rhel---8/8.4/x86_64/product-software`, we will select the binary ISO and right-click to copy the **Uniform Resource Locator** (**URL**) for the download.

> **Tip**
>
> The URLs obtained when copying from the **Customer Portal** are timebound, meaning they are only valid for a short period of time, and afterward, the download link is no longer valid and a new one should be obtained after refreshing the URL.

In a terminal, we will then execute the following command with the copied URL:

```
nohup wget URL_OBTAINED_FROM_CUSTOMER_PORTAL &
```

With the preceding command, `nohup` will not close the processes on terminal hang-up (disconnection), so `wget` will continue downloading the URL, and the ending ampersand symbol (`&`) detaches the execution from the active terminal, leaving it as a background job we can check with the `jobs` command until it has finished.

If we forgot to add the ampersand, the program will be blocking our input, but we can press *Ctrl* + *Z* on the keyboard and the process will be stopped. However, as we really want it to be continuing execution but in the background, we will execute `bg`, which will continue the execution of it.

In case we want to bring back the program to receive our input and interact with it, we can move it to the foreground with the `fg` command.

If we press *Ctrl* + *C* instead, while the program has our input, it will receive a petition to interrupt and stop execution.

You can see that workflow in the following screenshot:

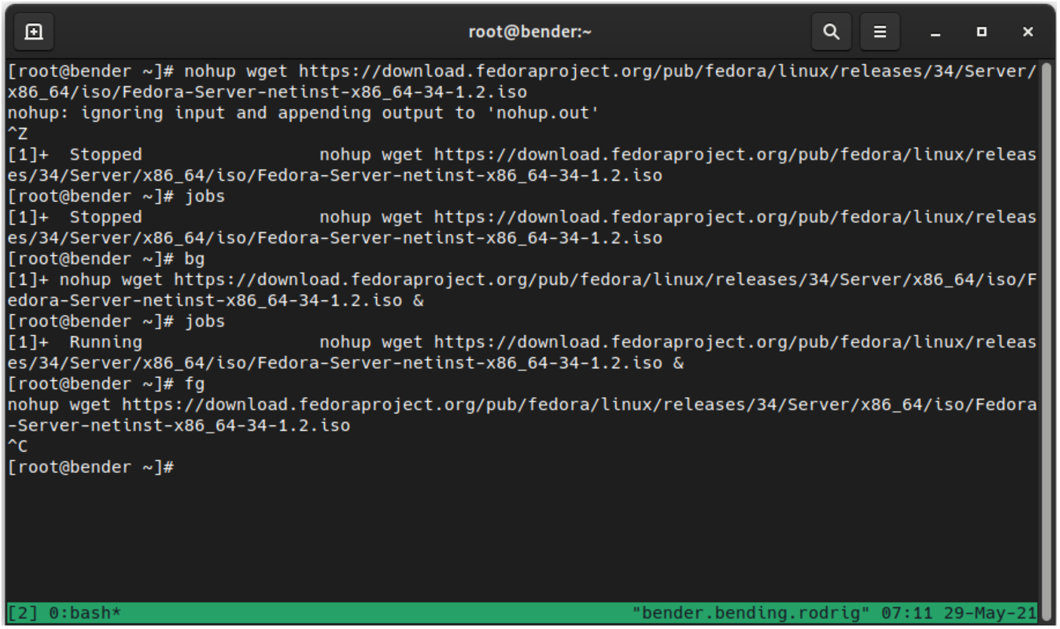


Figure 16.3 – Suspending the process, resuming to the background, bringing to the foreground, and aborting

In this case, we're downloading the Fedora 34 installation ISO (8 **gigabytes** (**GB**)) using `nohup` and `wget`; as we forgot to add the ampersand, we executed *Ctrl* + *Z* (appearing on screen as `^Z`).

The job was reported as job `[1]` with a status of `Stopped` (also reported when executing `jobs`).

Then, we bring the job to the background execution with `bg`, and now, `jobs` reports it as `Running`.

Afterward, we bring the job back to the foreground with `fg` and execute *Ctrl* + *C*, represented as `^C` on the screen, to finalize it.

This feature enables us to run multiple background commands—for example, we can copy a file in parallel to several hosts, as illustrated in the following screenshot:

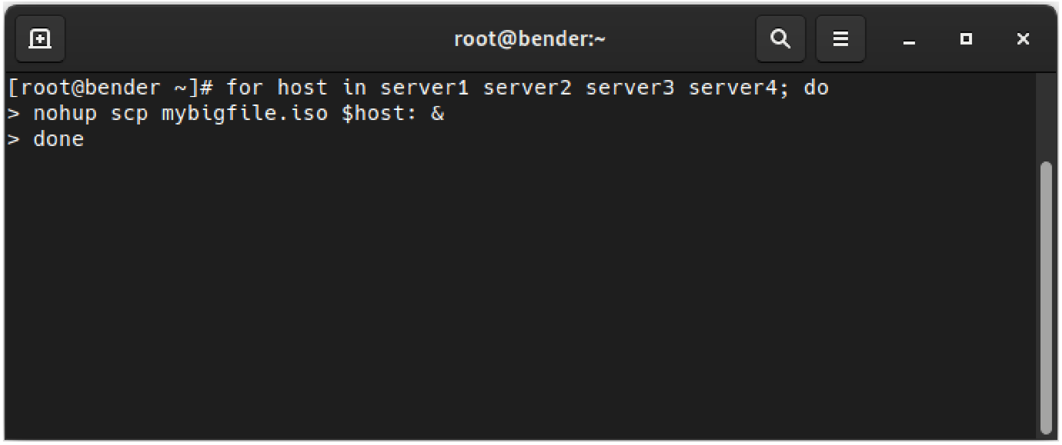


Figure 16.4 – Sample for loop to copy a file to several servers with nohup

In this example, the copy operation performed over `scp` will be happening in parallel, and, in the event of disconnection from our terminal, the job will continue execution and the output will be stored on `nohup.out` files in the folder we were executing it from.

> **Important note**
>
> Processes launched with `nohup` will not be getting any additional input, so in case the program asks for input, it will just stop execution. If the program asks for input, it's recommended to use `tmux` instead as it will still protect from terminal disconnection but also allow interaction with the launched program.

Not always will we be willing to kill processes or to stop or resume them; we may just want to deprioritize or prioritize them—for example, for long-running tasks that might not be critical.

Let's learn about this feature in the next section.

Adjusting kernel scheduling parameters to better manage processes

The Linux kernel is a highly configurable piece of software, so there's a whole world of tunables that can be used for adjusting its behavior: for processes, for network cards, for disk, for memory, and more.

The most common tunables are the `nice` process value and the I/O priority, which regulate respectively the prioritization versus other processes of the CPU and I/O time.

For interacting with processes we're about to start, we can use `nice` or `ionice` commands, prepending the command we want to execute with some parameters (remember to check the `man` contents for each one to get the full available range of options). Just remember that for `nice`, processes can go from –20 to +19, with 0 being the standard one, -20 the highest priority, and 19 the lowest priority (the higher the value, the nicer the process is).

Each process has a likelihood of getting kernel attention to run; by changing the priority via `nice` before execution or via `renice` once it's running, we can alter it a bit.

Let's think about a long-running process such as performing a backup—we want the task to succeed, so we will not be stopping or killing the process, but at the same time, we don't want it to alter the production or level of service of our server. If we define the process with a `nice` value of 19, this means that any process in the system will get more priority—that is, our process will keep running but will not make our system busier.

This gets us into an interesting topic—many new users arriving in the Linux world, or administrators of other platforms, get a shock when they see that the system, with plenty of memory (**random-access memory**, or **RAM**), is using swap space, or system load is high. It is clear that some slight usage of swap and having lots of free RAM just means that the kernel has optimized the usage by swapping out unused memory to disk. As long as the system doesn't feel sluggish, having a high load just means that the system has a long queue of processes to be executed, but—for example—if the processes are *niced* to 19, they are in the queue, but as mentioned, any other process will get ahead of it.

When we're checking the system status with `top` or `ps`, we can also check for how long a process has been running, and that is also accounted for by the kernel. A new process just created that starts eating CPU and RAM has a higher chance of being killed by the kernel to ensure system operability (remember the **out-of-memory** (**OOM**) killer mentioned in *Chapter 4*, *Tools for Regular Operations*?).

For example, let's `renice` the process running our backup (containing the backup pattern in the process name to the lowest priority) with the following code:

```
pgrep -f backup | xargs renice -n 19
143405 (process ID) old priority 0, new priority 19
144389 (process ID) old priority 0, new priority 19
2924457 (process ID) old priority 0, new priority 19
3228039 (process ID) old priority 0, new priority 19
```

As we can see, `pgrep` has collected a list of PIDs, and that list has been piped as arguments for `renice` with a priority adjustment of 19, making processes nicer to others actually running in the system.

Let's repeat the preceding example in our system by running a pi (π) calculation using `bc`, as illustrated in the man page for `bc`. First, we will time how long it takes for your system, and then, we will execute it via `renice`. So, let's get hands-on—first, let's time it, as follows:

```
time echo "scale=10000; 4*a(1)" | bc -l
```

In my system, this was the result:

```
real 3m8,336s
user 3m6,875s
sys  0m0,032s
```

Now, let's run it with `renice`, as follows:

```
time echo "scale=10000; 4*a(1)" | bc -l &
pgrep -f bc |xargs renice -n 19 ; fg
```

In my system again, this was the result:

```
real 3m9,013s
user 3m7,273s
sys  0m0,043s
```

There's a slight difference of 1 second, but you can try running more processes to generate system activity in your environment to make it more visible and add more zeros to the scale to increase the time of execution. Similarly, `ionice` can adjust the priority of I/O operations that a process is causing (reads, writes)—for example, repeating the action over the processes for our backup, we could run the following command:

```
pgrep -f  backup|xargs ionice -c 3 -p
```

By default, it will not output information, but we can check the value via execution of the following command:

```
pgrep -f backup|xargs ionice -p
idle
idle
idle
idle
```

In this case, we've moved our backup processes so that I/O requests are handled when the system is idle.

The class, which we specified with the `-c` argument, can be one of the following:

- `0`: None
- `1`: Real-time
- `2`: Best-effort
- `3`: Idle

With `-p`, we specify the processes to act on.

Most of the settings that we can apply to our system came from specific ones, applied to each PID via the `/proc/` virtual filesystem, such as—for example—adjusting the `oom_adj` file to reduce the value shown on the `oom_score` file, which in the end determines if the process should be higher in the list when OOM has to kill some process to try saving the system from catastrophe.

Of course, there are system-level settings such as `/proc/sys/vm/panic_on_oom` that can tune how the system has to react (panic or not) in case the OOM has to be invoked.

The disks also have a setting to define the scheduler being used—for example, for a disk named `sda`, it can be checked via `cat /sys/block/sda/queue/scheduler`.

The scheduler used for a disk has different approaches and depends on the kernel version—for example, it used to be `noop`, `deadline`, or `cfq` in RHEL 7, but in RHEL 8 those were removed, and we have `md-deadline`, `bfq`, `kyber`, and `none`.

This is such a big and complex topic that there is even a specific manual for it at `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/8/html-single/tuning_guide/index`, so if you're interested in going deeper, have a look at it.

I hope to have achieved two things here, as follows:

- Making clear that the system has a lot of options for tuning and that it has its own documentation for it, and even a Red Hat Certified Architect exam for it at `https://www.redhat.com/en/services/training/rh442-red-hat-enterprise-performance-tuning`.
- It's not an easy task—several times in this book, one idea has been reinforced: test everything using your system's workload, as results might vary from one system to another.

Fortunately, there's no need to feel afraid about system tuning—it's something we can become more proficient in with experience at all levels (knowledge, hardware, workloads, and so on), but on the other hand, systems also include some easier ways to perform quick adjustments that will fit many scenarios, as we will see in the next section.

Installing tuned and managing tuning profiles

Hopefully, after a bit of scaremongering happening in the previous section, you already have a mindset prepared for an easier path.

Just in case, ensure the `tuned` package is installed, or install it with `dnf -y install tuned`. The package provides a *tuned* service that must be enabled and started for operation; as a refresher, we achieve this by running the following command:

```
systemctl enable tuned
systemctl start tuned
```

We're now ready to interact and get more information about this service, which announces itself at `dnf info tuned` as a daemon that tunes the system dynamically based on some observation and is currently acting on an Ethernet network and hard disks.

Interaction with the daemon is performed via the `tuned-adm` command. For illustration, we're showing in the following screenshot the command-line options available and a list of profiles:

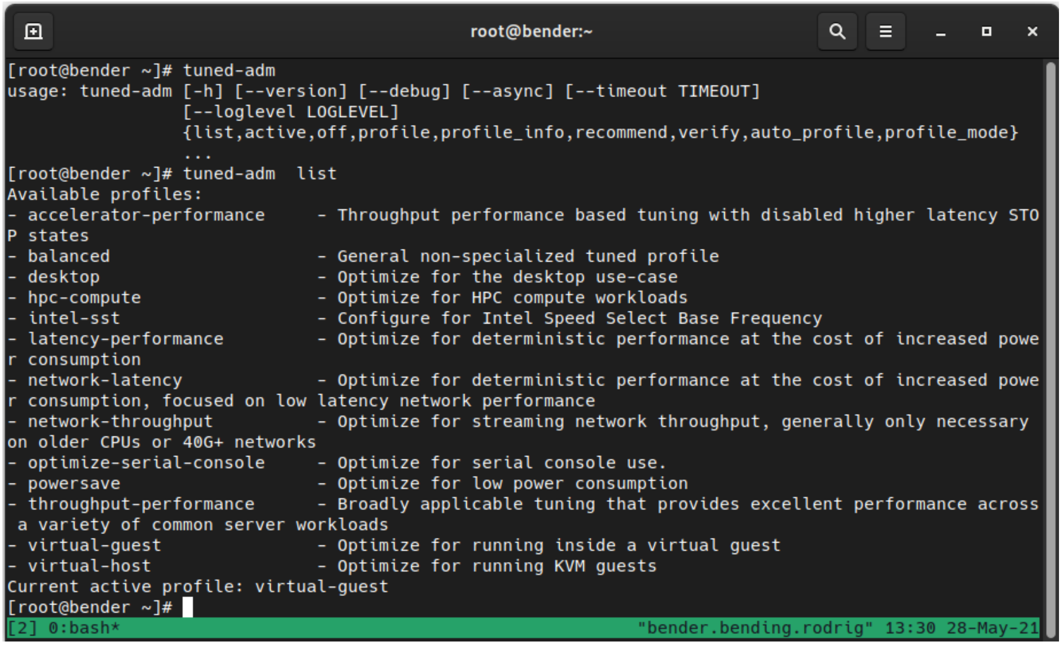


Figure 16.5 – The tuned-adm command-line options and profiles

As we can see, there are some options for listing, disabling, and grabbing information about a profile, getting recommendations on which profile to use, verifying that settings have not been altered, automatically selecting a profile, and so on.

One thing to bear in mind is that newer versions of the `tuned` package might bring additional profiles or configurations (stored in the `/usr/lib/tuned/` folder hierarchy), so the output might differ in your system.

Let's review some of the most common ones in the following table:

Profile	**Use case**
`virtual-host`	Optimizes server usage as host for VMs
`virtual-guest`	Optimizes the server as a VM running on top of `virtual-host`
`powersaving`	Tries to reduce power consumption as much as possible
`balanced`	Balances between power-saving and performance
`desktop`	Based on balanced, better responsiveness for applications

Profile	**Use case**
`throughput-performance`	Maximum throughput
`latency-performance`	Reduces latency and provides performance
`network-throughput`	Based on throughput performance, but with additional network tuning

As mentioned, each configuration is always a trade-off: more power consumption is required when increasing performance, or improving throughput might also increase latency.

Let's enable the `latency-performance` profile for our system. To do so, we will execute the following command:

```
tuned-adm profile latency-performance
```

We can verify that it has been activated with `tuned-adm active`, where we can see it shows `latency-performance`, as seen in the following screenshot:

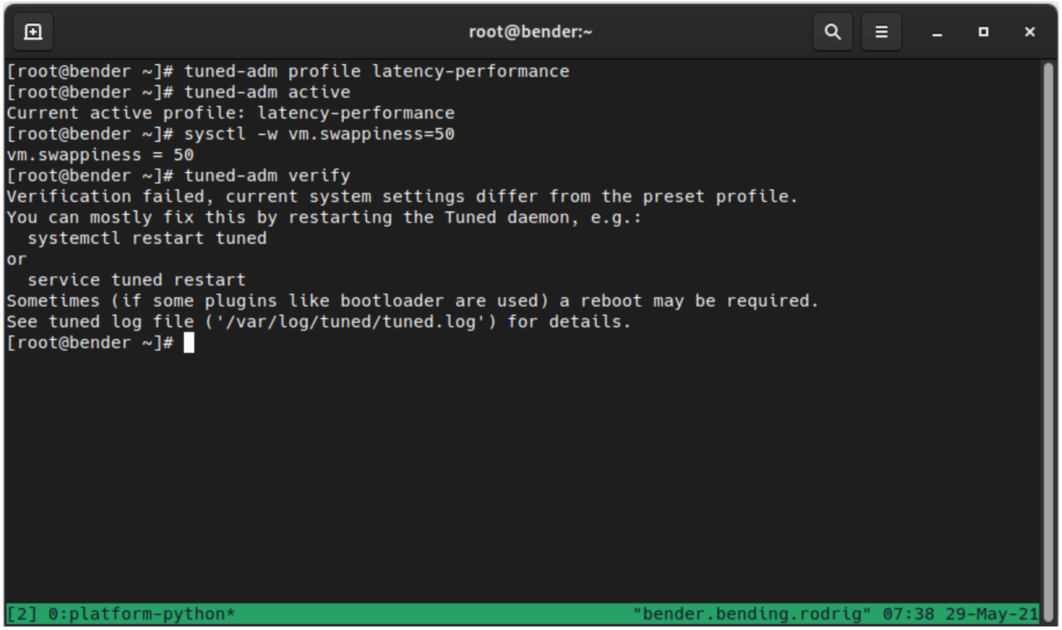


Figure 16.6 – The tuned-adm profile activation and verification

We additionally modified the system with `sysctl -w vm.swappiness=69` (on purpose) to demonstrate the `tuned-adm verify` operation, as it reported that some settings changed from the ones defined in the profile.

> **Important note**
> Dynamic tuning is by default disabled as of this writing—to enable or to check the current status, check that `dynamic_tuning=1` appears in the `/etc/tuned/tuned-main.conf` file. It is disabled in the performance profiles as it by default tries to balance between power consumption and system performance, which is the opposite of what performance profiles try to do.

Additionally, bear in mind that the **Cockpit** interface introduced in this book also features a way to change the performance profile—as seen in the following screenshot—once you have clicked on the **Performance profile** link in the main Cockpit page, opening up this dialog:

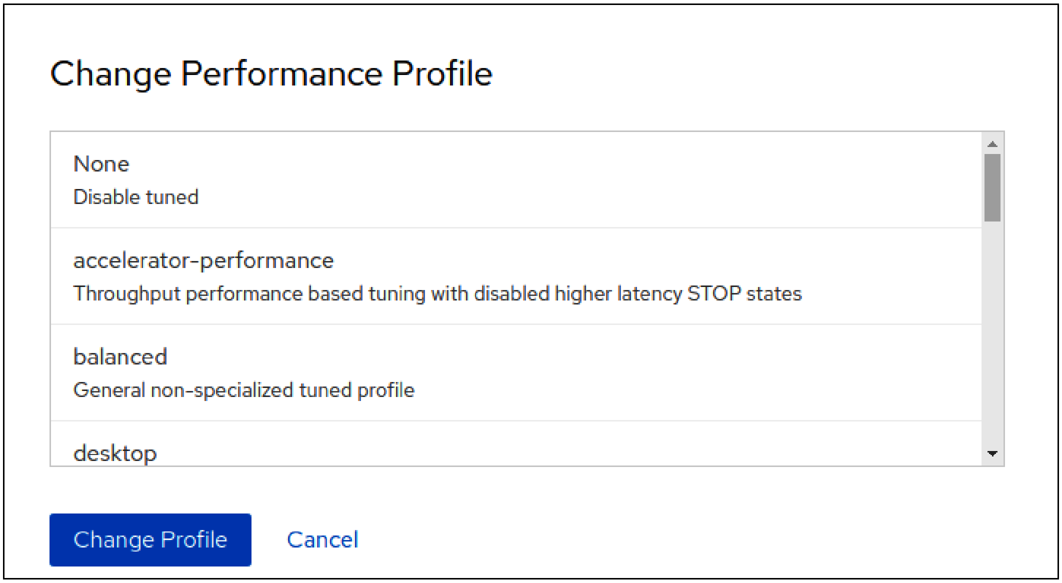


Figure 16.7 – Changing tuned profile within Cockpit web interface

In the next section, we will examine how tuned profiles work under the hood and how to create a custom one.

Creating a custom tuned profile

Once we've commented on the different tuned profiles... *How do they work? How to create one?*

Let's examine, for example, `latency-performance` in the next lines of code, by checking the `/usr/lib/tuned/latency-performance/tuned.conf` file.

In general, the syntax of the file is described in the `man tuned.conf` page, but the file, as you will be able to examine, is an *initialization (ini)-file*—that is, a file organized in categories, expressed between brackets and pairs of keys and values assigned by the equals (=) sign.

The main section defines a summary of the profile if it inherits from another profile via `include`, and the additional sections depend on the plugins installed.

To learn about the available plugins, the documentation included in the man page (`man tuned.conf`) instructs us to execute `rpm -ql tuned | grep 'plugins/plugin_.*.py$'`, which provides an output similar to this:

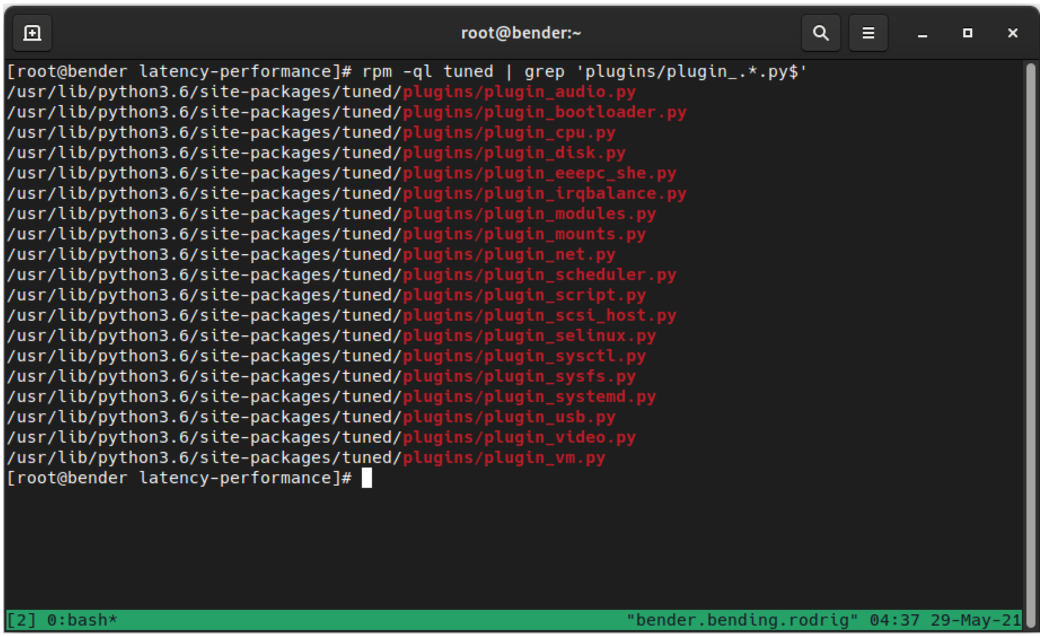


Figure 16.8 – Available tuned plugins in our system

> **Important note**
>
> If two or more plugins try to act over the same devices, the `replace=1` setting will mark the difference between running all of them or only the latest one.

Coming back to the `latency-performance` profile, this has three sections: `main`, `cpu`, and `sysctl`.

For the CPU, it sets the performance governor, which we can check—if supported via `cat /sys/devices/system/cpu/*/cpufreq/scaling_governor`—for each CPU available in our system. Bear in mind that in some systems, the path might differ or may even not exist, and we can check the available ones via execution of `cpupower frequency-info -governors`, with `powersave` and `performance` being the most common ones.

The name of the section for each plugin might be arbitrary if we specify the `type` keyword to indicate which plugin to use, and we can use some devices to act on via the `devices` keyword, allowing—for example—the definition of several disk sections with different settings based on the disk being configured. For example, we might want some settings for the system disk—let's say `sda`—and for the disk we use for data backups at `sdb`, as illustrated here:

```
[main_disk]
type=disk
devices=sda
readahead=>4096
[data_disk]
type=disk
devices=!sda
spindown=1
```

In the preceding example, the disk named `sda` gets configured with `readahead` (which reads sectors ahead of current utilization to have the data cached before actually being requested to access it), and we're telling the system to `spindown` data disks that might be used only at backup time, thus reducing noise and power consumption when not in use.

Another interesting plugin is `sysctl`, used by several of the profiles, which defines settings in the same way we might do with the `sysctl` command, and because of this, the possibilities are huge: defining **Transmission Control Protocol** (**TCP**) window sizes for tuning networking, virtual memory management, transparent huge pages, and so on.

> **Tip**
>
> It is hard to start from scratch with any performance tuning, and as `tuned` allows us to inherit settings from a parent, it makes sense to find which one of the available profiles is the closest to what we want to achieve, check what is being configured in it, and—of course—compare it with the others (as we can see, there are also examples for other plugins) and apply it to our custom profile.

To get an idea about how the defined system profiles touch a system, my RHEL 8 system shows the following output for `cat /usr/lib/tuned/*/tuned.conf|grep -v ^#|grep '^\['|sort -u`:

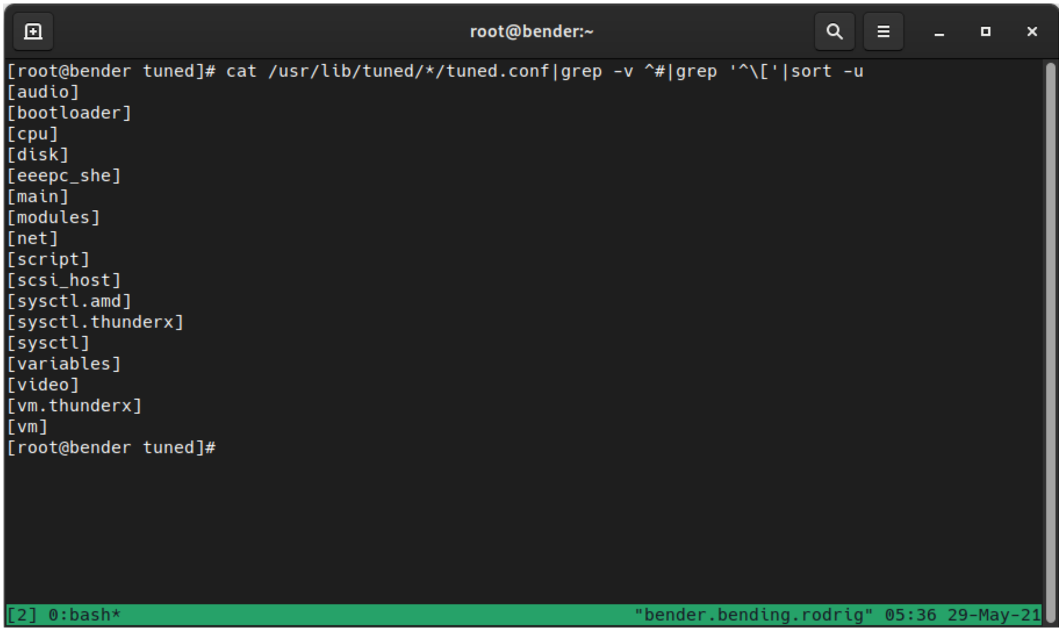


Figure 16.9 – Sections in system-supplied profiles

So, as we can see, they touch a lot of areas, and I would like to highlight the `script` section, which defines a shell script to execute used by the `powersave` profile, and the `variables` section, used by `throughput-performance` to define regular expressions for later matching and applying settings based on the CPU.

Once we're ready, we will create a new folder at `/etc/tuned/newprofile`. A `tuned.conf` file must be created, containing the main section with the summary and the additional sections for the plugins we want to use.

When creating a new profile, it might be easier if we copy the profile we're interested in from `/usr/lib/tuned/$profilename/` into our `/etc/tuned/newprofile/` folder and start the customization from there.

Once it's ready, we can enable the profile with `tuned-adm profile newprofile`, as we introduced earlier in this chapter.

You can find more information about the profiles available in the official documentation at `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/monitoring_and_managing_system_status_and_performance/index`.

With this, we've set up our own custom profile for tuning our performance settings.

Summary

In this chapter, we learned about identifying the processes, checking their resource consumption, and how to send signals to them.

About the signals, we learned that some of them have some additional behavior, such as terminating processes nicely or abruptly, or just sending a notification that some programs understand as reload configuration without restarting, and so on.

Also, related to processes, we learned about how to adjust their priority compared to other processes in terms of CPU and I/O so that we can adjust long-running processes or disk-intensive ones to not affect other services running.

Finally, we introduced the `tuned` daemon, which includes several general use-case profiles that we can use directly in our system, allowing `tuned` to apply some dynamic tuning, or we can fine-tune the profiles by creating one of our own to increase system performance or optimize power usage.

In the next chapter, we will learn about how we can work with containers, registries, and other components so that applications can run as provided by the vendor while being isolated from the server running them.

17 Managing Containers with Podman, Buildah, and Skopeo

In this chapter, we'll learn to use **Podman** and **Red Hat Universal Base Image**, also called **UBI**. Together, Podman and UBI provide users with the software they need to run, build, and share enterprise-quality containers on **Red Hat Enterprise Linux** (**RHEL**).

In recent years, understanding and using containers has become a key requirement for Red Hat systems administrators. In this chapter, we'll review the basics of containers, how containers work, and the standard tasks for managing containers.

You will learn how to run containers with simple commands, build enterprise-quality container images, and deploy them on a production system. You will also learn when to use more advanced tools such as **Buildah** and **Skopeo**.

These are the topics that will be covered in this chapter:

- Introduction to containers
- Running a container using Podman and UBI
- When to use Buildah and Skopeo

Technical requirements

In this chapter, we will review the basic usage of Podman, Buildah, and Skopeo, as well as how to build and run containers using Red Hat UBI.

We will create and run containers on the local RHEL 8 system, as we deployed in *Chapter 1*, *Installing RHEL8*. You will need to have the `container-tools:rhel8` **Application Stream** installed.

Introduction to containers

Containers provide users with a new way to run software on Linux systems. Containers provide all the dependencies related to a given piece of software, in a consistent redistributable manner. While containers were first made popular by Docker, Google, Red Hat, and many others joined Docker to create a set of open standards called the **Open Container Initiative** (**OCI**). The popularity of the OCI standards has facilitated a large ecosystem of tools where users don't have to worry about compatibility between popular container images, registries, and tools. Containers have become standardized in recent years and most major tools follow three standards governed by the OCI, outlined here:

- **Image specification**: Governs how container images are saved on disk
- **Runtime specification**: Specifies how containers are started by communicating with the operating system (in particular, the Linux kernel)
- **Distribution specification**: Governs how images are pushed and pulled from registry servers

You can learn more about this at `https://opencontainers.org/`.

All container tools (Docker, Podman, Kubernetes, and so on) need an operating system to run the container, and each operating system can choose different sets of technology to secure containers, as long as they comply with the OCI standards. RHEL uses the following operating system capabilities to securely store and run the containers:

- **Namespaces**: These are a technology in the Linux kernel that helps isolate processes from one another. Namespaces prevent containerized processes from having visibility of other processes on the host operating system (including other containers). Namespaces are what make a container feel like a **virtual machine** (**VM**).
- **Control groups (Cgroups)**: These limit the amount of **central processing unit** (**CPU**), memory, disk **input/output** (**I/O**), and/or network I/O available to a given process/container. This prevents the *noisy neighbor* problem.
- **Security-Enhanced Linux (SELinux)**: As described in *Chapter 10*, *Keeping Your System Hardened with SELinux*, this provides an extra layer of operating system security that can limit the damage caused by security exploits. SELinux is nearly transparent when used in conjunction with containers and provides mitigation of security breakouts, even when there are vulnerabilities in tools such as Podman, Docker, or Runc.

Many systems administrators use VMs to isolate applications and their dependencies (libraries, and so on). Containers provide the same level of isolation but reduce the overhead of virtualization. Since containers are simple processes, they do not need a **virtual CPU** (**vCPU**) with all of the overhead of translation. Containers are also smaller than VMs, which simplifies management and automation. This is particularly useful for **continuous integration/continuous delivery** (**CI/CD**).

RHEL provides users with container tools and images that are compatible with all OCI standards. This means they work in a way that is very familiar to anyone who has used Docker. For those unfamiliar with these tools and images, the following concepts are important:

- **Layers**: Container images are constructed as a set of layers. New containers are created by adding new layers (even to delete things) that reuse existing lower layers. The ability to use existing prepackaged containers is convenient for developers who simply want to make changes to their applications and test them in a reproducible way.
- **Distribution and deployments**: Since containers provide all the dependencies coupled with an application, they are easy to deploy and redistribute. Combining them with container registries makes it easy to share container images, and collaboration, deployments, and rollbacks are much quicker and easier.

The container tools provided with RHEL make it easy to deploy containers at a small scale, even for production workloads. But to manage containers at scale and with reliability, container orchestration such as Kubernetes is a much better fit. Red Hat, following the lessons learned from building Linux distributions, has created a Kubernetes distribution called **OpenShift**. If you need to deploy containers at scale, we recommend you take a look at this platform. The container tools and images provided in RHEL, and introduced in this chapter, will provide a solid foundation for later deploying to Kubernetes/OpenShift if and when you are ready for it. The tools introduced in this chapter are built in a way that will prepare your applications to be deployed in Kubernetes when you are ready for it.

Installing container tools

There are two Application Streams for **container tools** provided in RHEL 8. The first is a fast-moving stream that is updated every 12 weeks. The second is a stable stream that is released once a year and supported for 24 months.

Before we install the container tools, let's take a look at which ones are available, as follows:

```
[root@rhel8 ~]# yum module list | grep container-tools
```

```
container-tools      rhel8 [d][e]     common [d]
Most recent (rolling) versions of podman, buildah, skopeo,
runc, conmon, runc, conmon, CRIU, Udica, etc as well as
dependencies such as container-selinux built and tested
together, and updated as frequently as every 12 weeks.
```

```
container-tools      1.0              common [d]
Stable versions of podman 1.0, buildah 1.5, skopeo 0.1, runc,
conmon, CRIU, Udica, etc as well as dependencies such as
container-selinux built and tested together, and supported for
24 months.
```

```
container-tools      2.0              common [d]
Stable versions of podman 1.6, buildah 1.11, skopeo 0.1, runc,
conmon, etc as well as dependencies such as container-selinux
built and tested together, and supported as documented on the
Application Stream lifecycle page.
```

```
container-tools      3.0              common [d]
Stable versions of podman 3.0, buildah 1.19, skopeo 1.2, runc,
conmon, etc as well as dependencies such as container-selinux
built and tested together, and supported as documented on the
Application Stream lifecycle page.
```

Let's take a look at the main tools that we have listed, as follows:

- `podman`: This is the command to run containers. You may use it in every case in which you find the use of the `docker` command in the examples you will discover on the internet. It is the command we will use in this chapter to run our own containers.
- `buildah`: This is a tool specific to create container images. It uses the same Dockerfile definitions as Docker but without the need for a daemon.
- `skopeo`: A tool to introspect containers and check the different layers so that we can review whether they contain any non-compliant issues.

We will install the fast-moving stream to get access to the latest versions of Podman, Skopeo, and Buildah, as follows:

```
[root@rhel8 ~]# yum module install container-tools:rhel8
... [output omitted] ...
```

You now have a machine installed with all of the tools you need to build, run, and manage containers on an RHEL 8 system.

Running a container using Podman and UBI

Now you have the container tools' Application Stream installed, let's run a simple container based on Red Hat UBI that is a set of official container images and extra software based on RHEL. To run a UBI image, it only takes a single command, as illustrated in the following code snippet:

```
[root@rhel8 ~]# podman run -it registry.access.redhat.com/ubi8/
ubi bash
[root@407ca121cbbb /]#
```

> **Tip**
>
> These tutorials run commands as root, but one of the benefits of Podman is that it can run containers as a regular user without special permissions or a running daemon in the system.

You now have a fully isolated environment to execute whatever you want. You can run any commands you'd like in this container. It's isolated from the host and from other containers that might be running, and you can even install software on it.

> **Note**
>
> Red Hat UBI is based on software and packages from RHEL. This is the official image for use with RHEL and provides a rock solid, enterprise-ready base for your containers. UBI is used throughout this chapter.

Running a one-off container such as this is useful for testing new configuration changes and new pieces of software without interfering with software directly on the host.

Let's take a look at the processes running in the container, as follows:

```
[root@ef3e08e4eac2 /]# ps -efa
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 13:50 pts/0    00:00:00 bash
root          12       1  0 13:52 pts/0    00:00:00 ps -efa
```

As you can see, the only processes running are the shell we are using and the command we have just run. It is a completely isolated environment.

Now, exit the container by running the following command:

```
[root@407ca121cbbb /]# exit
[root@rhel8 ~]#
```

Now we have a working set of container tools and a UBI container image cached locally, we're going to move on to some more basic commands.

Basic container management – pull, run, stop, and remove

In this section, we'll run some basic commands to get familiar with using containers. First, let's pull some more images, as follows:

```
[root@rhel8 ~]# podman pull registry.access.redhat.com/ubi8/
ubi-minimal
...
[root@rhel8 ~]# podman pull registry.access.redhat.com/ubi8/
ubi-micro
...
[root@rhel8 ~]# podman pull registry.access.redhat.com/ubi8/
ubi-init
...
```

We now have several different images cached locally. Let's take a look at these here:

```
[root@rhel8 ~]#  podman images
REPOSITORY                                     TAG      IMAGE ID
CREATED      SIZE
registry.access.redhat.com/ubi8/ubi            latest
613e5da7a934  2 weeks ago  213 MB
registry.access.redhat.com/ubi8/ubi-minimal    latest
332744c1854d  2 weeks ago  105 MB
registry.access.redhat.com/ubi8/ubi-micro      latest
75d0ed7e8b6b  5 weeks ago  38.9 MB
registry.access.redhat.com/ubi8/ubi-init       latest
e13482c4e694  2 weeks ago  233 MB
```

Notice that we have four images cached locally. The Red Hat UBI actually comes in multiple flavors, as outlined here:

- **UBI Standard** (`ubi8/ubi`): An RHEL-based container base image with **YellowDog Updater Modified** (**YUM**)/**Dandified YUM** (**DNF**) in the image. It can be used in a similar way to any other Linux base image. This image is targeted at 80% of people's use cases and can easily be consumed from within a Dockerfile or Containerfile. The trade-off with this image is that it is larger than some of the other images.
- **UBI Minimal** (`ubi8/ubi-minimal`): This base image minimizes the size by using a small package manager called `microdnf` that is written in C instead of Python, like for standard YUM/DNF. This C implementation makes it smaller and pulls fewer dependencies into a container image. This base image can be used in any Dockerfile or Containerfile simply by using the `microdnf` command instead of `yum`. This image saves about 80 **megabytes** (**MB**) in memory.
- **UBI Micro** (`ubi8/ubi-micro`): This base image is built without a package manager. It cannot be used with a standard Dockerfile or Containerfile. Users instead add software to this image using the Buildah tool on the container host. This image is the smallest base image provided in RHEL.
- **UBI Init** (`ubi8/ubi-init`): Based on the RHEL standard image, this image also supports the use of `systemd` in the container. This makes it easy to install a few pieces of software, start them with `systemd`, and treat the container in a similar way to a VM. This image is best for users who don't mind slightly larger images and just want ease of use.

Now that you understand the basics of the four types of base images, let's start a container in the background so that we can inspect it while it's running. Start it in the background with the following command:

```
[root@rhel8 ~]# podman run -itd --name background ubi8 bash
262fa3beb8348333d77381095983233bf11b6584ec1f
22090604083c0d94bc50
```

Notice that when we start the container, the shell returns to normal and we can't type commands in the container. Our terminal doesn't enter a shell in the container. The `-d` option specified that the container should run in the background. This is how most server-based software such as web servers runs on a Linux system.

We can still connect our shell to a container running in the background if we need to troubleshoot one, but we have to determine which container we want to connect to. To do this, list all of the containers that are running with the following command:

```
[root@rhel8 ~]# podman ps
CONTAINER ID  IMAGE                                  COMMAND
CREATED             STATUS                   PORTS   NAMES
262fa3beb834  registry.access.redhat.com/ubi8:latest  bash
About a minute ago  Up About a minute ago            background
```

We could reference the container using the CONTAINER ID value, but we have started the container with the name background to make it easier to reference. We can enter the container and see what is going on inside it with the exec subcommand, as follows:

```
[root@rhel8 ~]# podman exec –it background bash
[root@262fa3beb834 /]#
```

After you type a few commands, exit the container by running the following command:

```
[root@262fa3beb834 /]# exit
```

Now, let's stop the containerized process by running the following command:

```
[root@262fa3beb834 /]# podman stop background
262fa3beb8348333d77381095983233bf11b6584ec1f
22090604083c0d94bc50
```

Check to make sure it's really stopped by running the following command:

```
[root@rhel8 ~]# podman ps -a
CONTAINER ID  IMAGE                                  COMMAND
```

```
CREATED                STATUS                       PORTS   NAMES
262fa3beb834  registry.access.redhat.com/ubi8:latest
bash     7 minutes ago  Exited (0) About a minute ago
background
```

Notice the state is `Exited`. This means the process has been stopped and is no longer in memory, but the storage is still available on disk. The container could be restarted, or we can delete it permanently with the following command:

```
[root@rhel8 ~]# podman rm background
262fa3beb8348333d77381095983233bf11b6584ec1f
22090604083c0d94bc50
```

This deleted the storage and the container has now gone forever. Verify this by running the following command:

```
[root@rhel8 ~]# podman ps -a
CONTAINER ID  IMAGE                                      COMMAND
CREATED                STATUS                       PORTS   NAMES
```

This section taught you some basic commands, but let's now move on to attaching storage.

Attaching persistent storage to a container

Remember that the storage in a container is ephemeral. Once the `podman rm` command is executed, the storage is deleted. If you have data that you need to save after the container is removed, you need to use a volume. To run a container with a volume, execute the following command:

```
[root@rhel8 ~]# podman run –it --rm -v /mnt:/mnt:Z --name data
ubi8 bash
[root@12ad2c1fcdc2 /]#
```

The preceding command has mounted `/mnt` into the container, and the `Z` option has told it to appropriately change the SELinux labels so that data can be written to it. The `--rm` option ensures that the container is removed as soon as you exit the shell. You can now save data on this volume, and it won't be removed when you exit the container. Add some data by running the following command:

```
[root@12ad2c1fcdc2 /]# touch /mnt/test.txt
[root@12ad2c1fcdc2 /]# exit
```

```
exit
[root@rhel8 ~]#
```

Now, inspect the test file you created by running the following command:

```
[root@rhel8 ~]# ls /mnt/data
test.txt
```

Notice the file is still on the system although the container has been removed, and its internal storage has been deleted.

Deploying a container on a production system with systemd

Since Podman is not a daemon, it relies on `systemd` to start a container when the system boots. Podman makes it easy to start a container with `systemd` by creating a `systemd` **unit file** for you. The process of running a container with `systemd` looks like this:

1. Run a container with Podman exactly how you want it to run in production.
2. Export a `systemd` unit file.
3. Configure `systemd` to use this unit file.

First, let's run an example container, as follows:

```
[root@rhel8 ~]# podman run -itd --name systemd-test ubi8 bash
D8a96d6a51a143853aa17b7dd4a827efa2755820c9967bee52
fccfeab2148e98
```

Now, let's export the `systemd` unit file that we'll use to start this container, as follows:

```
[root@rhel8 ~]# podman generate systemd --name --new systemd-
test > /usr/lib/systemd/system/podman-test.service
```

Enable and start the service by running the following command:

```
systemctl enable --now podman-test
Created symlink /etc/systemd/system/multi-user.target.wants/
podman-test.service → /usr/lib/systemd/system/podman-test.
service.
Created symlink /etc/systemd/system/default.target.wants/
podman-test.service → /usr/lib/systemd/system/podman-test.
service
```

Test that the container is running by executing the following command:

```
[root@rhel8 ~]# systemctl status podman-test
● podman-test.service - Podman container-systemd-test.service
Loaded: loaded (/usr/lib/systemd/system/podman-test.service;
enabled; vendor preset: disabled)
Active: active (running) since Thu 2021-04-29 20:29:30 EDT;
13min ago
[output omitted]
...
```

Now, check that the container is running by using the `podman` command, as follows:

```
[root@rhel8 ~]# podman ps
CONTAINER ID  IMAGE                                  COMMAND
CREATED             STATUS                     PORTS   NAMES
7cb55cc98e81  registry.access.redhat.com/ubi8:latest  bash
About a minute ago  Up About a minute ago              systemd-test
```

This container will now start every time the system boots; even if you kill the container with Podman, `systemd` will always make sure this container is running. Podman and `systemd` make it easy to run containers in production. Now, let's stop the container with `systemctl` and disable it, as follows:

```
systemctl stop podman-test
systemctl disable podman-test
```

Building a container image using a Dockerfile or Containerfile

Now we know how to run containers, let's learn how to build our own container images. Container images are commonly built with a file that serves as a blueprint for how to build it the same way every time. A **Dockerfile** or **Containerfile** has all of the information necessary to build container images. It makes it easy to script how a container will get built. A Containerfile is just like a Dockerfile, but the name attempts to make it more agnostic and not tied to the Docker tooling. Either type of file can be used with the container tools that come with RHEL. First, start by creating a file called `Containerfile` with the following content:

```
FROM registry.access.redhat.com/ubi8/ubi
RUN yum update -y
```

This simple Containerfile pulls the UBI standard base image and applies all of the latest updates to it. Now, let's build a container image by running the following command:

```
[root@rhel8 ~]# podman build -t test-build ./Containerfile
STEP 1: FROM registry.access.redhat.com/ubi8/ubi
STEP 2: RUN yum update -y
... [output omitted] ...
```

You now have a new image called `test-build` that has a new layer with all of the updated packages from the Red Hat UBI repositories, as illustrated in the following code snippet:

```
[root@rhel8 ~]# podman images
REPOSITORY                                        TAG      IMAGE ID
CREATED          SIZE
localhost/test-build                              latest
6550a939d3ef  9 minutes ago  335 MB
... [output omitted] ...
```

The workflow for building images from a Dockerfile or Containerfile is nearly identical to how Docker was in RHEL 7 or any other operating system. This makes it easy for system administrators and developers to move to Podman.

Configuring Podman to search registry servers

Container registries are like file servers for container images. They allow users to build and share container images, resulting in better collaboration. Often, it's useful to pull container images from public registry servers that are located on the internet, but in many instances, corporations have private registries that are not public. Podman makes it easy to search multiple registries, including private registries, on your company's network.

Podman comes with a configuration file that allows users and administrators to select which registries are searched by default. This makes it easy for users to find the container images that administrators want them to find.

A set of default registries to search are defined in `/etc/containers/registries.conf`. Let's take a quick look at this file by filtering all the comments in it, as follows:

```
[root@rhel8 ~]# cat /etc/containers/registries.conf | grep -v
^#
[registries.search]
registries = ['registry.access.redhat.com', 'registry.redhat.
```

```
io', 'docker.io']

[registries.insecure]
registries = []

[registries.block]
registries = []

unqualified-search-registries = ["registry.fedoraproject.org",
"registry.access.redhat.com", "registry.centos.org", "docker.
io"]

```

As you can see, we have the `registries.search` section for secure registries that includes the two main Red Hat registries, `registry.access.redhat.com` and `registry.redhat.io`, as well as the `docker.io` Docker registry. All of these registries are secured with **Transport Layer Security** (**TLS**) certificates, but Podman can also be configured to pull images without encryption using the `registries.insecure` section.

Separately from TLS, all images provided by Red Hat are signed and provide a signature store that can be used to verify them. This is not configured by default and is beyond the scope of this chapter.

To verify that Podman is using and searching the proper registries, run the following command:

```
[root@rhel8 ~]# podman info | grep registries -A 4
registries:
  search:
  - registry.access.redhat.com
  - registry.redhat.io
  - docker.io
```

> **Tip**
>
> If you want to publish your own images, you can do this in the service that Red Hat offers to do so: `https://quay.io`. You can also configure `registries.conf` to search `quay.io` for images you store there.

Summary of Podman options

Let's review the options used with Podman in this chapter, as follows:

Command	Usage
`podman run`	Runs a container with a specified image name
`podman run -it ubi8 bash`	Interactively runs the `bash` command in a new container created using a `ubi8` image
`podman pull`	Retrieves an image from the registry to be used at a later time
`podman ps`	Lists running containers
`podman stop`	Stops a running container
`podman rm`	Removes the instance of a container (not its image)
`podman exec`	Executes a command in a running container
`podman generate`	Helps generate configuration files such as `systemd` units
`podman info`	Shows information on Podman configuration
`podman build`	Builds a container image using the specifications coming in a Containerfile or Dockerfile
`podman images`	Lists the images available in the local system

As you can see while reviewing the table, Podman includes options to manage the full container life cycle. Most Podman commands are compatible with `docker`. Podman even provides a package (`podman-docker`) that supplies an alias from `podman` to `docker` so that users can continue to type a command they are familiar with. While Podman and Docker feel quite similar to use, Podman can be run as a regular user and does not require a daemon to be continuously running. Let's move on to the next section to explore some advanced use cases.

When to use Buildah and Skopeo

Podman is a general-purpose container tool and should solve 95% of a user's needs. Podman leverages Buildah and Skopeo as libraries and pulls these tools together under one interface. That said, there are edge cases when a user may want to leverage Buildah or Skopeo separately. We will explore two such use cases.

Building container images with Buildah

Building from a Dockerfile or Containerfile is quite easy, but it does come with some trade-offs. For example, Buildah is good in the following situations:

- When you need granular control over committing image layers. This can be necessary when you want two or three commands to run, and then commit a single layer.
- When you have difficult-to-install software—for example, some third-party software comes with standardized installers that don't understand they are being run in a Dockerfile. Many of these `install.sh` installers assume they have access to the entire filesystem.
- When a container image does not provide a package manager. UBI Micro builds very small images because it does not have a Linux package manager installed, nor any of the dependencies of a package manager.

For this example, let's build on top of UBI Micro to demonstrate why Buildah is such a great tool. First, create a new container to work with, as follows:

```
[root@rhel8 ~]# buildah from registry.access.redhat.com/ubi8/
ubi-micro
ubi-micro-working-container
```

The preceding command created a reference to a new container called `ubi-micro-working-container`. Once Buildah creates this reference, you can build upon it. To make it easier, let's start over and save the reference in a shell variable, as follows:

```
microcontainer=$(buildah from registry.access.redhat.com/ubi8/
ubi-micro)
```

You can then mount the new container as a volume. This lets you modify the container image by changing files in a directory. Run the following command to do this:

```
micromount=$(buildah mount $microcontainer)
```

Once the container storage is mounted, you can modify it in any way you would like. These changes will eventually be saved as a new layer in the container image. This is where you could run an installer (`install.sh`), but in the following example, we will use the package manager on the host to install packages in UBI Micro:

```
yum install \
    --installroot $micromount \  --releasever 8 \  --setopt
install_weak_deps=false \  --nodocs -y \    httpd
```

```
... [output omitted] ...
[root@rhel8 ~]# yum clean all \
    --installroot $micromount
... [output omitted] ...
```

When the package installation completes, we will unmount the storage and commit the new image layer as a new container image called `ubi-micro-httpd`, as illustrated in the following code snippet:

```
[root@rhel8 ~]# buildah umount $microcontainer
467403b1633fbcb42535e818929fd49a5e381b86733c99d
65cd8b141e9d64fff
[root@rhel8 ~]# buildah commit $microcontainer ubi-micro-httpd
Getting image source signatures
Copying blob 5f70bf18a086 skipped: already exists
Copying blob 8e7500796dee skipped: already exists
Copying blob 881a7504d0b5 skipped: already exists
Copying blob 771043083e15 done
Copying config 9579d04234 done
Writing manifest to image destination
Storing signatures
9579d0423482e766d72e3909f34e8c10d4258128d5cae394
c1f0816ac637eda0
```

You now have a new container image with `httpd` installed, built on UBI Micro. Only a minimal set of dependencies have been pulled in. Look here at how small the image is:

```
[root@rhel8 ~]# podman images
localhost/ubi-micro-
httpd                                     latest
9579d0423482  About a minute ago  152 MB
```

Buildah is a wonderful tool that gives you a lot of control over how builds are done. Now, we will move on to Skopeo.

Inspecting a remote container with Skopeo

Skopeo is specifically designed and built to work on remote container repositories. With the following command, you can easily remotely inspect the available tags for an image:

```
[root@rhel8 ~]# skopeo inspect docker://registry.access.redhat.
com/ubi8/ubi
{
    "Name": "registry.access.redhat.com/ubi8/ubi",
    "Digest":
"sha256:37e09c34bcf8dd28d2eb7ace19d3cf634f8a073058ed63ec6e
199e3e2ad33c33",
    "RepoTags": [
        "8.2-343-source",
        "8.1-328",
        "8.2-265-source",
... [output omitted] ...
```

Remote inspection is useful to determine if you want to pull an image, and if so, with which tag. Skopeo can also be used to copy between two remote registry servers without caching a copy in the local storage. For more information, see the `skopeo` man pages.

Summary

In this chapter, we have reviewed the basics of how to run, build, and share containers on RHEL 8. You are prepared to create your own containers, run them, manage them, and even use `systemd` to ensure they're always running in a production environment.

You are now ready to leverage the functionality and ease of deployment that containers provide. While a deep dive into all of the intricacies of migrating software into containers is outside the scope of this book, containers simplify packaging and delivery of applications ready to be executed with all of their dependencies.

Containers are now a strong focus within the **Information Technology** (**IT**) industry. Containers alone simplify the packaging and delivery of applications, but orchestration platforms such as OpenShift (based on Kubernetes) make it easier to deploy, upgrade, and manage containerized applications at scale.

Congratulations—you have come to the end of this chapter! It's now time to move to the next chapter and take a self-assessment to ensure you've absorbed the material and to practice your skills. There are two more chapters to go.

Section 4: Practical Exercises

This section includes practical exercises to review what was learned in the previous sections. It includes an intermediate practice and a more advanced one, allowing you to assess your progress.

The following chapters are included in this section:

- *Chapter 18*, *Practice Exercises – 1*
- *Chapter 19*, *Practice Exercises – 2*

18 Practice Exercises – 1

In this practice exercise, we will run a set of steps to check the knowledge you have acquired throughout this book. As opposed to previous chapters, not all steps will be indicated, and it's therefore left to your discretion to perform the steps required to accomplish your desired goals. It is recommended to avoid referencing past chapters for guidance. Instead, try to use your memory, or the tools available in the system. This exercise, if performed correctly, will train you effectively for official exams.

It is strongly advised to start this exercise with a clock to keep track of time.

Technical requirements

All the practice exercise in this chapter requires the use of a virtual machine (**VM**), running Red Hat Enterprise Linux 8 installed with the base installation. Additionally, new virtual drives will be required for storage operations.

For the exercises, it is assumed you possess the following:

- Red Hat Enterprise Linux 8 installed with the base operating system **minimal install** software selection.
- Access to the Red Hat Customer Portal, with an active subscription.
- The VM must be expendable. This is because actions performed on it during the exercise might render it unusable and require it to be reinstalled.

Tips for the exercise

This is a list of general recommendations for any test, and most fall under the category of common sense, but it's always important to have them in our mind before performing any such test:

- Read all the questions before starting the official exam or any test.
- Specific words have specific meanings that give hints about the requirements or ways to accomplish the goals. This is why reading everything first might give you multiple perspectives on how to complete the test.
- Make yourself comfortable. Install your favorite editor and run `updatedb` to have a fresh database of packages and installed files ready for use. Define your keyboard layout. Install `tmux` and learn how to use it, so that you can open new tabs and name them without requiring extra windows.
- Locate dependencies between requests, as some goals depend on others for completion. Find those dependencies to see how you can locate the solution without later having to come back and redo some steps as a result of choosing the wrong path.
- Use a timer. This is important to get an idea of what exercises will take you more time to complete, in order to see the areas that you need to improve in.
- Don't remember specific command lines. Learn how to use the documentation available in the system via `man`, `/usr/share/docs` or arguments like `--help` for the commands required.
- Ensure that changes persist and are still active after a reboot. Some changes might be active while you run them, but those must be persisted. Examples may include firewall rules, services to start at boot, and so on.
- Remember to use `dnf whatprovides /COMMAND"` to find packages providing a file you might be missing.

- Check the following link: `https://www.redhat.com/en/services/training/ex200-red-hat-certified-system-administrator-rhcsa-exam?=Objectives`. This will provide you with the official EX200 exam objectives.

Practice exercise 1

> **Important note**
>
> The following exercise has, by design, been created so that there will be no highlights on commands, packages, and so on. Remember what you've learned so far in order to detect the keywords to see what needs to be done.

Don't jump into the walk-through too early. Try to remember what was covered.

Exercises

1. Configure the time zone to GMT.
2. Allow password-less login for the root user using SSH.
3. Create a user (named *user*) that can connect to the machine without a password.
4. The user `user` should change their password every week, with 2 days' warning and 1 day of usage once expired.
5. The root user must be able to SSH as *user* without a password so that nobody can connect remotely as root using a password.
6. The user *user* should be able to become root user without a password, and also execute commands without a password.
7. When a user tries to log in over SSH, display a legal message about not allowing unauthorized access to this system.
8. SSH must listen on port *22222*, instead of the default one (*22*).
9. Create a group named `devel`.
10. Make `user` a member of `devel`.
11. Store user membership in a file named `userids` in the home folder for *user.*
12. The user *user*, and *root* user, should be able to connect to localhost via SSH without specifying the port, and default to compression for the connection.
13. Find all man page names in the system, and put the names into a file named *manpages.txt*.

14. Print usernames for users without logins permitted to the system. For each username, print the user ID and groups for that user.
15. Monitor available system resources every 5 minutes. Do not use cron. Store as */root/resources.log*.
16. Add a per-minute job to report the available percentage of free disk space, and store it in */root/freespace.log* so that it shows both the filesystem and free space.
17. Configure the system to only leave 3 days of logs.
18. Configure the log rotation for */root/freespace.log* and */root/resources.log*.
19. Configure the time synchronization against *pool.ntp.org*, using fast sync.
20. Provide NTP server services for the subnet *172.22.0.1/24*.
21. Configure system stats for collection every minute.
22. Configure the password length for users in the system to be 12 characters long.
23. Create a bot user named *privacy,* which will keep its files only visible to itself by default.
24. Create a folder in *shared* that can be accessed by all users, and that defaults new files and directories to still be accessible to users of the *devel* group.
25. Configure a network connection with IPv4 and IPv6 addresses named *mynic,* using the following data:

```
Ip6: 2001:db8:0:1::c000:207/64 g
gateway 2001:db8:0:1::1
Ipv4 192.0.1.3/24
gateway 192.0.1.1
```

26. Allow the host to use a *google* hostname to reach `www.google.com`, and a *redhat* hostname to reach `www.redhat.com`.
27. Report the files modified from those that the vendor distributed, and store them in */root/altered.txt*.
28. Make our system installation media packages available via HTTP under the */mirror path* for other systems to use as a mirror, configuring the repository in our system. Remove the kernel packages from that mirror, so that other systems (even ours) can't find new kernels. Prevent the glibc packages from being installed from this repo without removing them.
29. While being *user*, make a copy of the */root* folder in */home/user/root/* folder, and keep it in sync every day, synchronizing additions and deletions.

30. Check that our system conforms to the PCI-DSS standard.
31. Add a second hard drive of 30 GB to the system. However, use only 15 GB to move the mirror to it, making it available at boot using compression and deduplication. Make it available under */mirror/mirror*.
32. As we plan to mirror custom sets of packages based on the same data, configure the filesystem to report at least 1,500 GB to be used by our mirrors.
33. Create a second copy of the mirror under */mirror/mytailormirror*, removing all packages starting with the letter *k**.
34. Create a new volume in the remaining space of the added hard drive (15 GB), and use it to extend the root filesystem.
35. Create a boot entry that allows you to boot into emergency mode, in order to change the root password.
36. Create a custom tuning profile that defines the readahead to be *4096* for the first drive and *1024* for the second drive. This profile should also crash the system should an OOM event occur.
37. Disable and remove the installed HTTP package. Then, set up the HTTP server using the *registry.redhat.io/rhel8/httpd-24* image.

For this section, we'll copy each item in the list of goals, and then provide an explanation below it, using proper syntax highlighting and explanations.

Exercise 1 resolution

1. Configuring the time zone to GMT

We can check the current system date by executing the `date` command. At the very last part of the line that is subsequently printed, the time zone will be shown. In order to configure it, we can use the `timedatectl` command, or alter the `/etc/localtime` symbolic link.

So, to achieve this goal, we can use one of the following:

- `timedatectl set-timezone GMT`
- `rm -fv /etc/localtime; ln -s /usr/share/zoneinfo/GMT /etc/localtime`

Now `date` should report the proper time zone.

2. Allowing password-less login to the root user using SSH

Doing this will require the following:

- SSH must be installed and available (that means installed and started).
- The root user should have an SSH key generated and added to the list of authorized keys.

First, let's tackle this with SSH, as seen in the following:

```
dnf -y install openssh-server; systemctl enable sshd; systemctl
start sshd
```

Now, let's generate an SSH key by pressing *Enter* to accept all defaults:

```
ssh-keygen
```

Now, let's add the generated key (`/root/.ssh/id_rsa`) to the authorized keys:

```
cd; cd .ssh; cat id_rsa.pub >> authorized_keys; chmod 600
authorized_keys
```

To validate this, we can execute `ssh localhost date`, after which we will be able to get our current system's date and time without providing a password.

3. Creating a user named 'user' that can connect to the machine without a password

This requires creating a user and an SSH key that is added in a similar fashion to the root user. The next options will also be relevant to a user, but for the purposes of this demonstration, we will tackle them as separate tasks:

```
useradd user
su - user
```

Now, let's generate an SSH key by pressing *Enter* to accept all defaults:

```
ssh-keygen
```

Now, let's add the generated key (/root/.ssh/id_rsa) to the authorized keys:

```
cd; cd .ssh; cat id_rsa.pub >> authorized_keys; chmod 600
authorized_keys
```

To validate this, we can execute `ssh localhost date`, and we will be able to get the current system date and time without providing a password.

Then, use `logout` to return to our `root` user.

4. The user 'user' should change their password every week, with 2 days' warning and 1 day of usage once expired

This requires us to tune the user restrictions, as follows:

```
chage -W 2 user
chage -I 1 user
chage -M 7 user
```

5. The root user must be able to SSH as 'user' without a password, so that nobody can connect remotely as the root user using a password

This requires two steps. The first is to enable `user` with the root's authorized key, and then tune the `sshd` daemon, as follows:

```
cat /root/id_rsa.pub >> ~user/.ssh/authorized_keys
```

Edit the `/etc/sshd/sshd_config` file and add or replace the `PermitRootLogin` line so that it looks like the following:

```
PermitRootLogin prohibit-password
```

Save and then restart the `sshd` daemon:

```
systemctl restart sshd
```

6. The user 'user' should be able to become root and execute commands without a password

This means configuring the /etc/sudoers file by adding the following line:

```
user ALL=(ALL) NOPASSWD:ALL
```

7. When a user tries to log in over SSH, display a legal message about not allowing unauthorized access to this system

Create a file, for example, /etc/ssh/banner, with the message to display. For example, "Get out of here".

Modify /etc/ssh/sshd_config and set the line banner with /etc/ssh/banner, then restart the sshd daemon with systemctl restart sshd.

8. SSH must listen on port 22222 instead of the default one

This is a tricky one. The first step is to alter /etc/ssh/sshd_config and define port 22222. Once this is done, restart sshd with the following command:

```
systemctl restart sshd
```

This, of course, will fail... why?

The firewall must be configured:

```
firewall-cmd --add-port=22222/tcp --permanent
firewall-cmd --add-port=22222/tcp
```

SELinux must then be configured:

```
semanage port -a -t ssh_port_t -p tcp 22222
```

Now, the sshd daemon can be restarted:

```
systemctl restart sshd
```

9. Creating a group named 'devel'

Use the following command:

```
groupadd devel
```

10. Making 'user' a member of 'devel'

Use the following command:

```
usermod -G devel user
```

11. Storing user membership in a file called 'userids,' in a home folder for 'user'

Use the following command:

```
id user > ~user/userids
```

12. The user 'user' and root user should be able to connect to the localhost via SSH, without specifying the port, and default to compression for the connection

We altered the default SSH port to be `22222`.

Create a file named `.ssh/config` for both `user` and root, with the following contents:

```
Host localhost
Port 22222
    Compression yes
```

13. Finding all man page names in the system, and putting the names into a file named 'manpages.txt'

Man pages are stored in `/usr/share/man`. Therefore, use the following command:

```
find  /usr/share/man/ -type f > manpages.txt
```

14. Printing usernames for users without a login, so they can be permitted access to the system, and printing the user ID and groups for each user

The following command first builds a list of users in the system with the `nologin` shell:

```
for user in $(cat /etc/passwd| grep nologin|cut -d ":" -f 1)
do
echo "$user -- $(grep $user /etc/group|cut -d ":" -f 1|xargs)"
done
```

From that list, check the membership in the `/etc/group` file, leaving only the group name, and using `xargs` to concatenate them into a string to be printed.

The above example makes use of `for` loops and inline execution of commands, via `$()`.

15. Monitoring available system resources every 5 minutes without using cron, and storing them as /root/resources.log

The ideal way to monitor something would be cron, but as we're told not to use it, this only leaves us with systemd timers. (You can check the files tested via the following link: `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/tree/main/chapter-18-exercise1`.

Create `/etc/systemd/system/monitorresources.service` with the following contents:

```
[Unit]
Description=Monitor system resources

[Service]
Type=oneshot
ExecStart=/root/myresources.sh
```

Create `/etc/systemd/system/monitorresources.timer` with the following contents:

```
[Unit]
Description=Monitor system resources
```

```

[Timer]
OnCalendar=*-*-* *:0,5,10,15,20,25,30,35,40,45,50,55:00
Persistent=true

[Install]
WantedBy=timers.target
```

Create `/root/myresources.sh` with the following contents:

```
#!/bin/bash
df > /root/resources.log
```

Enable the new timer, as follows:

```
systemctl daemon-reload
systemctl enable  monitorresources.timer
```

Does it work? If not, `journalctl -f` will give some details. SELinux prevents us from executing a root file, so let's convert it into a binary type and mark it as executable, as shown in the following:

```
chcon -t bin_t /root/myresources.sh
chmod +x /root/myresources.sh
```

16. Adding a per-minute job to report the available percentage of free disk space and storing it in /root/freespace.log, so that it shows the filesystem and free space

The `df` reports used disk space and available space, so we need to do some math.

This will report the mounted location, size, used space, and available space, with `;` as a separator. Refer to the following:

```
df|awk '{print $6";"$2";"$3";"$4}'
```

Bash allows us to do some math operations, but these lack fractional parts. Luckily, we can do a trick: we will be looping over it, as follows:

```
for each in $(df|awk '{print $6";"$2";"$3";"$4}'|grep -v
"Mounted")
do
    FREE=$(echo $each|cut -d ";" -f 4)
    TOTAL=$(echo $each|cut -d ";" -f 2)
    echo "$each has $((FREE*100/TOTAL)) free"
done
```

The `for` loop will check all the available data, grab some specific fields, separate them with `;`, and then run the loop for each line, stored in the `$each` variable.

We cut the output and then get the fourth field. This is the available space.

We cut the output and then we get the second field. This is the total number of blocks.

As `bash` can do integer divisions, we can multiply by 100 and then divide to get the percentage and add a string as part of the output.

Alternatively (but not as illustrative), we could have discounted to 100 the percentage of used already given by `df` and save some steps of the calculation.

We also need to store the output in a file. To do this, we can either wrap the whole loop in a redirection or add it in the `echo` line so that it appends to a file.

And we also need to do it via cron, so the full solution will be as follows:

Create a `/root/myfreespace.sh` script with the following contents:

```
for each in $(df|awk '{print $6";"$2";"$3";"$4}'|grep -v
"Mounted")
do
    FREE=$(echo $each|cut -d ";" -f 4)
    TOTAL=$(echo $each|cut -d ";" -f 2)
    echo "$each has $((FREE*100/TOTAL)) free"
done
```

Then, use `chmod 755 /root/myfreespace.sh` to make it executable.

Run `crontab -e` to edit root's crontab, and add the following line:

```
*/1 * * * * /root/myfreespace.sh >> /root/freespace.log
```

17. Configuring the system to only leave 3 days of logs

This can be done by editing `/etc/logrorate.conf`, with the following settings:

```
daily
rotate 3
```

Remove other occurrences of weekly, monthly, and so on, to leave only the one we want.

18. Configuring log rotation for /root/freespace.log and /root/resources.log

Create a `/etc/logrotate.d/rotateroot` file, with the following contents:

```
/root/freespace.log {
    missingok
    notifempty
    sharedscripts
    copytruncate
}
/root/resources.log {
    missingok
    notifempty
    sharedscripts
    copytruncate
}
```

19. Configuring time synchronization against pool.ntp.org with fast sync

Edit `/etc/chrony.conf` and add the following line:

```
pool pool.ntp.org iburst
```

Then, run the following:

```
systemctl restart chronyd
```

20. Providing NTP server services for subnet 172.22.0.1/24

Edit /etc/chrony.conf, by adding the following line:

```
Allow 172.22.0.1/24
```

Then, run the following:

```
systemctl restart chronyd
```

21. Configuring system stats collection every minute

Run the following command:

```
dnf -y install sysstat
```

We now need to modify /usr/lib/systemd/system/sysstat-collect.timer. Let's do this by creating an override, as follows:

```
cp /usr/lib/systemd/system/sysstat-collect.timer /etc/systemd/
system/
```

Edit /etc/systemd/system/sysstat-collect.timer by replacing the OnCalendar value so that it looks like the following:

```
OnCalendar=*:00/1
```

Then, reload the units with the following command:

```
systemctl daemon-reload
```

22. Configuring the password length in the system for users to be 12 characters

Edit /etc/login.defs with the following line:

```
PASS_MIN_LEN 12
```

23. Creating a bot user called 'privacy,' which keeps its files only visible to itself by default

To do this, run the following:

```
adduser privacy
su - privacy
echo "umask 0077" >> .bashrc
```

This resolution uses `umask` to remove permissions from others on all newly created files.

24. Creating a folder named /shared that can be accessed by all users, and defaults new files and directories to still be accessible to users of the 'devel' group

To do this, run the following:

```
mkdir /shared
chown root:devel /shared
chmod 777 /shared
chmod +s /shared
```

25. Configuring a network connection with IPv4 and IPv6 addressing named 'mynic,' using the provided data Ip6, as follows: 2001:db8:0:1::c000:207/64 g gateway 2001:db8:0:1::1 IPv4 192.0.1.3/24 gateway 192.0.1.1

See the following for how to accomplish this:

```
nmcli con add con-name mynic type ethernet ifname eth0 ipv6.
address 2001:db8:0:1::c000:207/64 ipv6.gateway 2001:db8:0:1::1
ipv4.address 192.0.1.3/24 ipv4.gateway 192.0.1.1
```

26. Allowing the host to use a google hostname to reach www.google.com, and a redhat hostname to reach www.redhat.com

Run and record the IPs obtained, as shown here:

```
ping www.google.com
ping www.redhat.com
```

Note down the IPs obtained above.

Edit `/etc/hosts` by adding the following:

```
IPFORGOOGLE google
IPFORREDHAT redhat
```

Then, save and exit.

27. Reporting the files modified from those that the vendor distributed, and storing them in /root/altered.txt

See the following for how to accomplish this:

```
rpm  -Va > /root/altered.txt
```

28. Making our system installation media packages available via HTTP under the path /mirror for other systems to use it as the mirror, and configuring the repository in our system. Removing the kernel packages from that mirror so that other systems (even ours) can't find new kernels. Ignoring the glibc packages from this repo to be installed without removing them

This is a complex one, so let's examine it step by step.

Install `http` and enable it using the following:

```
dnf -y install httpd
firewall-cmd  --add-service=http --permanent
firewall-cmd  --add-service=http
systemctl start httpd
systemctl enable httpd
```

Create a folder under `/mirror`, then copy the source media packages and make them available over `http`:

```
mkdir /mirror /var/www/html/mirror
mount /dev/cdrom /mnt
rsync -avr -progress /mnt/ /mirror/
mount -o bind /mirror /var/www/html/mirror
chcon  -R -t httpd_sys_content_t /var/www/html/mirror/
```

Remove the kernel packages:

```
find /mirror -name kernel* -exec rm '{}' \;
```

Create repository file metadata by using the following commands:

```
dnf -y install createrepo
cd /mirror
createrepo .
```

Create a repository file using the repository we created, and set it up on the system, ignoring the `glibc*` packages from it.

Edit `/etc/yum.repos.d/mymirror.repo` by adding the following contents:

```
[mymirror]
name=My RHEL8 Mirror
baseurl=http://localhost/mirror/
enabled=1
gpgcheck=0
exclude=glibc*
```

29. As 'user,' make a copy of the /root folder in the /home/user/root/ folder, and keep it in sync every day, synchronizing additions and deletions

See the following for how to accomplish this:

```
su - user
crontab -e
```

Edit crontab and add the following line:

```
@daily rsync  -avr --progress --delete root@localhost:/root/ /
home/user/root/
```

30. Checking whether our system conforms to the PCI-DSS standard

```
dnf -y install openscap  scap-security-guide openscap-utils
oscap xccdf eval --report pci-dss-report.html --profile pci-dss
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

31. Adding a second hard drive of 30 GB to the system, but using only 15 GB to move the mirror to it, making it available at boot using compression and deduplication, and available under /mirror/mirror

Compression and deduplication in this sentence mean VDO. We need to move the mirror we currently have to it and make the old mirror we had go there instead.

If we have the installation media, we can choose to copy it over and repeat the kernel removal or transfer. To do so, first let's create the VDO volume in a partition in our new hard drive (`sdb`):

```
fdisk /dev/sdb
n <enter>
p <enter>
1 <enter>
<enter>
+15G <enter>
```

```
w <enter>
q <enter>
```

This will create a partition of 15 GB from the start. Let's create a VDO volume on it, by using the following command:

```
dnf -y install vdo kmod-kvdo
vdo create -n myvdo -device /dev/sdb --force
pvcreate /dev/mapper/myvdo
vgcreate myvdo /dev/mapper/myvdo
lvcreate -L 15G -n myvol myvdo
mkfs.xfs /dev/myvdo/myvol
# Let's umount cdrom if it was still mounted
umount /mnt
# Mount vdo under /mnt and copy files over
mount /dev/myvdo/myvol /mnt
rsync -avr -progress /mirror/ /mnt/mirror/
# Delete the original mirror once copy has finished
rm -Rfv /mirror
umount /mnt
mount /dev/myvdo/myvol /mirror
```

At this point, the old mirror was copied into a `mirror` folder on the VDO volume. This is mounted under `/mirror`, hence it has the original mirror under `/mirror/mirror` as requested. We might need to perform the following:

- Bind mount `/mirror` to `/var/www/html/mirror/` to make the file available.
- Restore SELinux context to allow the `httpd` daemon to access to files in `/var/www/html/mirror/`.

Adjust the repofile we created to point to the new path.

32. Configuring the filesystem to report at least 1,500 GB in size, to be used by our mirrors

See the following command:

```
vdo growLogical --name=myvdo --vdoLogicalSize=1500G
```

33. Creating a second copy of the mirror under /mirror/mytailormirror and removing all packages starting with k*

See the following for how to accomplish this:

```
rsync -avr -progress /mirror/mirror/ /mirror/mytailormirror/
find /mirror/mytailormirror/ -name "k*" -type f -exec rm '{}'
\;
cd /mirror/mytailormirror/
createrepo .
```

34. Creating a new volume in the remaining space (15 GB) of the hard drive and using it to extend the root filesystem

See the following for how to accomplish this:

```
fdisk /dev/sdb
n <enter>
p <enter>
<enter>
<enter>
w <enter>
q <enter>
pvcreate /dev/sdb2
# run vgscan to find out the volume name to use (avoid myvdo as
is the VDO from above)
vgextend $MYROOTVG /dev/sdb2
# run lvscan to find out the LV storing the root filesystem and
pvscan to find the maximum available space
lvresize -L +15G /dev/rhel/root
```

35. Creating a boot entry that allows us to boot into emergency mode in order to change the root password

See the following for how to accomplish this:

```
grubby --args="systemd.unit=emergency.target" --update-kernel=/
boot/vmlinuz-$(uname -r)
```

36. Creating a custom tuning profile that defines the readahead to be 4096 for the first drive and 1024 for the second drive – this profile should also crash the system should an OOM event occur

Refer to the following command:

```
dnf -y install tuned
mkdir -p /etc/tuned/myprofile
```

Edit the `/etc/tuned/myprofile/tuned.conf` file by adding the following contents:

```
[main]
summary=My custom tuned profile

[sysctl]
vm.panic_on_oom=1

[main_disk]
type=disk
devices=sda
readahead=>4096
[data_disk]
type=disk
devices=!sda
readahead=>1024
```

37. Disabling and removing the installed httpd package, and setting up the httpd server using the registry.redhat.io/rhel8/httpd-24 image

See the following for how to accomplish this:

```
rpm –e httpd
```

```
dnf –y install podman
```

```
podman login registry.redhat.io # provide RHN credentials
```

```
podman pull registry.redhat.io/rhel8/httpd-24
```

```
podman run -d --name httpd –p 80:8080 -v /var/www:/var/www:Z
registry.redhat.io/rhel8/httpd-24
```

19
Practice Exercise – 2

In this second practice exercise chapter, we will run a set of exercises to check the knowledge you've acquired throughout this book. In contrast with this book's chapters, not all the steps will be specified; it's left up to your discretion to perform the steps required to accomplish the necessary goals. It is recommended that you avoid checking back on the chapters for guidance and instead try to use your memory or the tools available in the system. This experience will be a key factor when you take on the official exams.

It is strongly advised that you start this exercise with a clock so that you know how long it took for you to complete.

Technical requirements

All the practical exercises in this chapter require a **virtual machine** (**VM**) running Red Hat Enterprise Linux 8 to be installed with the base installation. Additionally, new virtual drives will be required for storage operations.

The exercises assume that you have the following:

- Red Hat Enterprise Linux 8 installed with a base operating system **Minimal Install** software selection.
- Access to the Red Hat Customer Portal with an active subscription.
- The VM must be expendable; that is, actions that you perform on it might render it unusable, so it will have to be reinstalled.

Tips for the exercise

This is a list of general recommendations for any test, most of which are common sense, but it's always interesting to keep them in mind:

- Read the questions in their entirety before starting the exam.
- Specific words have specific meanings that give hints about the requirements or ways to accomplish the exercise. That's why, again, reading everything first might add or remove possibilities.
- Make yourself comfortable: install your favorite editor, run `updatedb` so that you have a fresh database of packages and files ready for you, and define your keyboard layout. Install and learn the basics of how to use `tmux` so that you can open new tabs and name them without requiring extra windows.
- Locate dependencies between requests. Some goals depend on others for completion, so find those dependencies to see how you can build up the solution without having to go back and redo some steps because of taking the wrong path.
- Use a timer. It's important to get an idea of what exercises took more time to complete so that you can find areas to improve upon.
- Don't remember specific commands. Instead, learn how to use the documentation available in the system by using `man`, `/usr/share/docs` arguments such as `--help` for the commands, and so on.
- Ensure that changes persist and are still active after a reboot. Some changes might be active while you run them, but those must be persisted: firewall rules, services to start at boot, and so on.
- Remember that you can use `dnf whatprovides "*/COMMAND"` to find packages regarding that file you might be missing.
- Check `https://www.redhat.com/en/services/training/ex200-red-hat-certified-system-administrator-rhcsa-exam?=Objectives` for the official EX200 exam objectives.

Practice exercise – 2

> **Important Note**
> By design, in the following exercise, commands, packages, and so on will not be highlighted. Remember what you've learned so far to detect the keywords to see what needs to be done.

Don't jump into the solution too early; try to think and remember what was covered.

Exercises

1. Download the necessary file from this book's GitHub repository at `https://raw.githubusercontent.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/main/chapter-19-exercise2/users.txt`.
2. Use the `users.txt` file to generate users in the system in an automated way using the values provided, in the following order: `username`, `placeholder`, `uid`, `gid`, `name`, `home`, `shell`.
3. Create a group named users and add that group as the primary group to all users, leaving their own groups, named after each user, as secondary groups.
4. Change the home folders for the users so that they are group-owned.
5. Set up an HTTP server and enable a web page for each user with a small introduction for each that is different between users.
6. Allow all users in the `users` group to become root without a password.
7. Create SSH keys for each user and add each key to root and the other users so that each user can SSH like the other users; that is, without a password.
8. Disable password access to the system with SSH.
9. Set each user with a different password using `/dev/random` and store the password in the `users.txt` file in the second field of the file.
10. If the number of letters in the username is a multiple of 2, add that fact to each user description web page.
11. Create a container that runs the `yq` python package as the entry point.
12. Configure password aging for users that are not a multiple of 2 to so that they're expiring.
13. Configure a daily compressed log rotation for a month of logs using date-named files.
14. Save all the logs that have been generated in the day in `/root/errors.log`.
15. Install all the available updates for the system libraries.
16. Repair the broken rpm binary using a previously downloaded package available in the `/root` folder.
17. Make all the processes that are executed by the user doe to run with a low priority and the ones from john run with a higher priority (+/- 5).
18. Make the system run with the highest throughput and performance.

19. Change the system network interface so that it uses an IP address that's higher than the one it was using. Add another IPv6 address to the same interface.
20. Create and add `/opt/mysystem/bin/` to the system PATH for all users.
21. Create a firewall zone, assign it to an interface, and make it the default zone.
22. Add a repository hosted at `https://myserver.com/repo/` with a GPG key from `https://myserver.com/mygpg.key` to the system since our server might be down. Configure it so that it can be skipped if it's unavailable.

Answers to practice exercise 2

In this section, we'll copy each item from the list of goals, and explain them while using proper syntax highlighting.

1. Download the necessary file from this book's GitHub repository at https://raw.githubusercontent.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/main/chapter-19-exercise2/users.txt

```
wget https://raw.githubusercontent.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/main/chapter-19-exercise2/users.txt
```

2. Use the users.txt file to generate users in the system in an automated way using the values provided, in the following order: username, placeholder, uid, gid, name, home, shell

First, let's examine the `users.txt` file with the following code:

```
cat users.txt
user;x;1000;1000;myuser1;/home/user1; /bin/false
john ;x ;1001 ;1001; John; /home/john ;/bin/false
doe ;x ;1001 ;1001; Doe; /home/doe ; /bin/csh
athena ;x ;1011 ;1011; Athena Jones; /home/ajones ; /bin/rsh
```

```
pilgrim ;x ;2011 ;2011; Scott Pilgrim; /home/spilgrim ; /bin/rsh
laverne; x ; 2020;2020; LaVerne;/home/LaVerne;/bin/bash
```

As described in the request, the fields in that file are `username`, `placeholder`, `uid`, `gid`, `name`, `home`, `shell`. The placeholder is not asked to create a user as it's usually the password so that we can work with the other data while ignoring that.

As we can also see, each field is separated by at least a `;` symbol, but some have extra spaces before or after it. Since we also have surnames, we can't just remove all spaces; we need to do this before and after the actual text we want.

We need to use cut with the `;` field separator, but first, we need to read the file line by line.

We can achieve this with Bash's built-in `read` function:

```
cat users.txt|while read -r line; do echo ${line};done
```

Using this as a base, we can start building up everything we're going to need to create the users. Let's start by working on the individual steps and then build up the full command line.

We have lots of lines, so for each one, we need to define the fields and remove the end/start spaces:

```
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1)
NEWUID=$(echo ${line}|cut -d ";" -f 3)
NEWGID=$(echo ${line}|cut -d ";" -f 4)
NEWNAME=$(echo ${line}|cut -d ";" -f 5)
NEWSHELL=$(echo ${line}|cut -d ";" -f 6)
```

In the preceding examples, we're echoing each line and cutting the field specified with `-f` using the `;` field delimiter. This allows us to select exactly the field containing the data we're looking for. To make this easier, we can store each in a variable so that we can reuse that snippet of code and still have a clear understanding of what each script will be doing.

The preceding code will work, *but* it will fail with the spaces, so we need to extend them to just capture the actual text without the spaces. Let's use `xargs` for this:

```
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
NEWUID=$(echo ${line}|cut -d ";" -f 3|xargs)
NEWGID=$(echo ${line}|cut -d ";" -f 4|xargs)
NEWNAME=$(echo ${line}|cut -d ";" -f 5|xargs)
```

```
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
NEWSHELL=$(echo ${line}|cut -d ";" -f 7|xargs)
```

The next step is to build the command line for adding a user:

```
useradd --d "${NEWHOME}" --m --s "${NEWSHELL}" --u "${NEWUID}"
--g "${NEWGID}" --c "${NEWNAME}" "${NEWUSERNAME}"
```

Now that everything's ready, let's build the solution:

```
cat users.txt| while read -r line ; do
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
NEWUID=$(echo ${line}|cut -d ";" -f 3|xargs)
NEWGID=$(echo ${line}|cut -d ";" -f 4|xargs)
NEWNAME=$(echo ${line}|cut -d ";" -f 5|xargs)
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
NEWSHELL=$(echo ${line}|cut -d ";" -f 7|xargs)
useradd -d "${NEWHOME}" -m -s "${NEWSHELL}" -u "${NEWUID}" -g
"${NEWGID}" -c "${NEWNAME}" "${NEWUSERNAME}"
done
```

3. Create a group named users and add that group as the primary group to all users, leaving their own groups, named after each user, as secondary groups

In this case, we need to create the groups that we didn't create in the previous step. So, we will loop over the users once the new group has been created, create new groups for each, and then modify the user to get the `users` group and add their own as secondary groups:

```
groupadd users
cat users.txt| while read -r line ; do
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
groupadd -g ${NEWGID} ${NEWUSERNAME}
usermod -g users -G ${NEWUSERNAME} ${NEWUSERNAME}
done
```

4. Change the home folders for the users so that are group owned

```
cat users.txt| while read -r line ; do
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
chown -R ${NEWUSERNAME}:users ${NEWHOME}/
done
```

5. Set up an HTTP server and enable a web page for each user, with a small introduction for each that is different between users

```
dnf -y install httpd
firewall-cmd --add-service=http --permanent  f
firewall-cmd --reload
-
cat users.txt| while read -r line ; do
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
NEWUID=$(echo ${line}|cut -d ";" -f 3|xargs)
NEWGID=$(echo ${line}|cut -d ";" -f 4|xargs)
NEWNAME=$(echo ${line}|cut -d ";" -f 5|xargs)
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
NEWSHELL=$(echo ${line}|cut -d ";" -f 7|xargs)
mkdir -p ${NEWHOME}/public_html/
echo "Hello, my name is ${NEWNAME} and I'm a user of this
system" > ${NEWHOME}/public_html/index.htm
Done
```

Finally, we'll need to enable `homedirs` by editing `/etc/httpd/conf.d/userdir.conf` and disabling `UserDir` so that it becomes `Userdir public_html`:

```
service httpd start
```

6. Allow all the users in the users group to become root without a password

This can be done in several ways, but since all the users are in the `users` group, we can add that group:

```
echo "%users ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
```

7. Create SSH keys for each user and add each key to root and the other users so that each user can SSH like the other users; that is, without a password

First, let's create the keys for each user and add the keys to root:

```
cat users.txt| while read -r line ; do
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
mkdir -p ${NEWHOME}/.ssh/
ssh-keygen -N '' -f ${NEWHOME}/.ssh/id_dsa
cat ${NEWHOME}/.ssh/id_dsa.pub >> /root/.ssh/authorized_keys
done
```

Now, let's copy the authorized keys for each user:

```
cat users.txt| while read -r line ; do
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
cp /root/.ssh/authorized_keys  ${NEWHOME}/.ssh/ >>
chown -R ${NEWUSERNAME}:users ${NEWHOME}/.ssh/
Done
```

Validate that users can `ssh` just like any other user:

```
USERS=$(cat users.txt|cut -d ";" -f1|xargs)
for user in ${USERS};
do
for userloop in ${USERS};
do
su -c "ssh ${user}@localhost" ${userloop}
```

```
done
done
```

The preceding command should work for all the users because we copied `authorized_keys`, right? This isn't the case as some users have their shell disabled.

8. Disable password access to the system with SSH

Edit `/etc/ssh/sshd_config` and replace any value of `PasswordAuthentication` with `no`.

Then, restart `sshd`:

```
systemctl restart sshd
```

9. Set each user with a different password using /dev/random and store the password in the users.txt file in the second field of the file

From `/dev/random`, we can get random data, but it's binary, so it's probably not valid if we want to use it for logging in later. We can use a hash function over the data we've received and use that as the password:

```
MYPASS=$(dd if=/dev/urandom count=1024 2>&1|md5sum|awk '{print
$1}')
```

This will be the password, without the need for it to be encrypted.

With `usermod`, we can define a password from its encrypted seed, so we will be combining both.

Additionally, we're told to store the generated password in `users.text`, so we will need to edit the file.

But there's a problem: editing a specific field in the `.txt` file might not be easy, but we can just rewrite it completely:

```
cat users.txt| while read -r line ; do
MYPASS=$(dd if=/dev/random count=12>&1|md5sum|awk '{print $1}')
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
NEWUID=$(echo ${line}|cut -d ";" -f 3|xargs)
NEWGID=$(echo ${line}|cut -d ";" -f 4|xargs)
```

```
NEWNAME=$(echo ${line}|cut -d ";" -f 5|xargs)
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
NEWSHELL=$(echo ${line}|cut -d ";" -f 7|xargs)
echo  "${NEWUSERNAME};${MYPASS};${NEWUID};${NEWGID};${NEWNAME};
${NEWHOME};${NEWSHELL}" >> newusers.txt
echo ${MYPASS} | passwd ${NEWUSERNAME} --stdin
done
cp newusers.txt users.txt
```

In this way, we've rewritten the `users.txt` file to a new file by adding all the fields we had and overwritten `users.txt` with our new copy.

The last command in the loop reads the password from the variable and feeds it to the `passwd` file, which will encrypt and store it while reading it from `stdin`.

10. If the number of letters in the username is a multiple of 2, add that fact to each users description web page

```
cat users.txt| while read -r line ; do
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
LETTERSINNAME=$(( $(echo ${NEWUSERNAME}|wc -m) - 1 ))
if [ "$((${LETTERSINNAME} % 2 ))" == "0" ]; then
echo "My name is multiple of 2" >> ${NEWHOME}/public_html/
index.htm
done
done
```

In this example, we repeat the same field calculation, but we add the `wc` command to get the number of characters and remove one to adjust it to the number of letters.

In the comparison, we evaluate the remainder when dividing by 2, so that when there's no remainder, this means that our number of letters is a multiple of 2.

11. Create a container that runs the yq Python package

When we read "Python package," we should think about PIP. PIP is not recommended to be used on systems directly as it might alter the system-provided Python libraries, and it's better to use a virtual environment for it. Alternatively, you can use a container that will keep it isolated.

As described in the *Chapter 17, Managing Containers with Podman, Buildah, and Skopeo*, the easiest way to do this is by creating a file that defines the container creation steps.

For containers, it will be also required to install the `podman` package and the `container-tools` modules if you don't have them in your system.

As this file is a Python package, we require a container that already has Python in it; for example, `https://catalog.redhat.com/software/containers/rhel8/python-38/5dde9cb15a13461646f7e6a2`.

So, let's create a `Containerfile` with the following contents (available at `https://github.com/PacktPublishing/Red-Hat-Enterprise-Linux-8-Administration/blob/main/chapter-19-exercise2/ContainerFile`):

```
FROM registry.access.redhat.com/ubi8/python-38
MAINTAINER RHEL8 Student <student@redhat.com>
LABEL name="yq image" \
maintainer="student _AT_ redhat.com" \
vendor="Risu" \
version="1.0.0" \
release="1" \
summary="yq execution container" \
description="Runs yq"
ENV USER_NAME=risu \
USER_UID=10001 \
LC_ALL=en_US.utf8
RUN pip3 install --upgrade pip --no-cache-dir && \
pip3 install --upgrade yq --no-cache-dir
USER 10001
VOLUME /data
ENTRYPOINT ["/opt/app-root/bin/yq"]
CMD ["-h"]
```

When combined with `podman build -t yq -f ContainerFile`, it will pull the `ubi8` image with Python so that we can just run the `pip3 install` command to install `yq`, which will be then assigned as our `entrypoint`.

For example, if we define an invalid `entrypoint` (because we might not know where the program is installed), we can use `podman run -it --entrypoint /bin/bash <podmanid>`. We can get the podman ID by running `podman images` and checking the generation date for each of the available pods in our system.

The created container can be tested with `podman run –it <podmanid>`, where it will output the information about what the `yq` command does.

Note that `yq`, as expressed in their repository at `https://github.com/kislyuk/yq`, requires that we have installed the `jq` command, but we left it out on purpose to demonstrate how to create the container.

12. Configure password aging for users that are not a multiple of 2 so that they're expiring

```
cat users.txt| while read -r line ; do
NEWUSERNAME=$(echo ${line}|cut -d ";" -f 1|xargs)
NEWHOME=$(echo ${line}|cut -d ";" -f 6|xargs)
LETTERSINNAME=$(( $(echo ${NEWUSERNAME}|wc -m) - 1 ))
if [ "$((${LETTERSINNAME} % 2 ))" != "0" ]; then
chage -M 30 ${NEWUSERNAME}
done
done
```

Here, we've reused the loop from question 10, but inverted the conditional. Since there's no requirements regarding the kind of password aging we can use, we just need to define the maximum number of days before a password change is required to be `30 days`.

13. Configure the daily compressed log rotation for a month of logs using date-named files

First, we need to make sure that `logrotate` is installed:

```
dnf -y install logrotate
```

Once installed, edit the `/etc/logrotate.conf` file so that it contains the following:

```
rotate 30
daily
compress
dateext
```

We need to ensure that no other period is defined (monthly, weekly, and so on).

14. Save all the logs generated in the day in /root/errors.log

This has a trick to it: some programs will log to the journal, while some of them will log to `*.log` files.

The date for today can be obtained with `+%Y-%m-%d`. This format, which uses the year-month-day format, is commonly used in program logs:

```
grep "$(date '+%Y-%m-%d')" -Ri /var/log/*.log|grep -i error >
/root/errors.log
journalctl --since "$(date '+%Y-%m-%d')" >> /root/errors.log
```

By doing this, we combine both outputs. We could, of course, try to sort the entries by date so that they correlate, but bear in mind that the first `grep` does a recursive search, so the filename is being prepended, making it harder to sort.

15. Install all the available updates for system libraries

Usually, the system libraries contain the `lib` substring in them, so the update should be a matter of running the following command:

```
dnf upgrade *lib*
```

As it will ask for confirmation, review the listed packages to make sure that no errors occurred.

16. Repair the broken rpm binary using a previously downloaded package available in the /root folder

This is a tricky but useful knowledge check.

First, let's make sure that the `rpm` package is available:

```
yumdownloader rpm
```

Verify that the file exists with the following command:

```
ls -l rpm*.rpm
```

Check the file to make sure we have a way to go back in case we breack it beyond repair:

```
rpm -qip rpm*.rpm
```

Now, let's look at the destructive action that will help us validate we are solving the issue:

```
rm -fv /usr/bin/rpm
```

From here, it's like *look ma, no hands*... no RPM is available to install the `rpm*.rpm` package, but we still need to install it to fix the issue.

`rpm` packages are compressed `cpio` archives, so what we can do is use the following command:

```
rpm2cpio rpm*.rpm |cpio -idv
```

This will extract the compressed `rpm` contents (without the need to run a script).

Move the uncompressed `rpm` file back into `/usr/bin`:

```
mv usr/bin/rpm /usr/bin/rpm
```

Verify the installation and operation of `rpm`:

```
rpm -V rpm
```

It will complain. saying that at least the date has changed. However, it may have also updated the sizes and md5sum if the downloaded file was newer.

Move the system to a sane state by reinstalling the `rpm` package:

```
rpm -i rpm*.rpm
```

This will make the system complain because the package was already installed (it will state that it will overwrite `rpm`, `rpm2archive`, `rpm2cpio`, `rpmdb`, `rpmkeys`, and more).

If the `rpm` version differs, we can just upgrade it with the following command:

```
rpm -Uvh rpm*.rpm
```

Then, we can verify this with the following command:

```
rpm -V rpm
```

Nothing should be reported as changed regarding what the database contains. If we cannot upgrade, we can run the installation with the `--force` argument to tell `rpm` that it's OK to continue and overwrite the files.

Alternatively, once the `rpm` binary has been restored with `cpio`, we can use the following command:

```
dnf -y reinstall rpm
```

Another approach for this could have been to `scp` the `rpm` binary from a similar system or to use rescue media.

17. Make all the processes that are executed by the user doe run with a low priority and the ones from john run with a higher priority (+/- 5)

We have no way of making this a default, but we can combine a cron job to do so.

Execute crontab `-e` as root to edit the root's crontab and set up a job that runs every minute:

```
*/1 * * * *  pgrep -u doe |xargs renice +5
*/1 * * * * pgrep -u john|xargs renice -5
```

This will use `pgrep` for all the PIDs for john and doe and feed them via `xargs` to the `renice` process.

Alternatively, we could use something like the following:

```
renice +5 $(pgrep -u doe)
```

This can be used as an alternative to the `xargs` command.

18. Make the system run with the highest throughput and performance

`tuned` is a system daemon we can install to automatically apply some well-known parameters to our system, which will become the base for our specific optimizations later:

```
dnf -y install tuned
systemctl enable tuned
systemctl start tuned
tuned-adm profile throughput-performance
```

19. Change the system network interface so that it uses an IP address that's higher than the one it was using. Add another IPv6 address to the same interface

Using `nmcli`, check the current system IP's address:

```
nmcli con show
```

The output should be as follows:

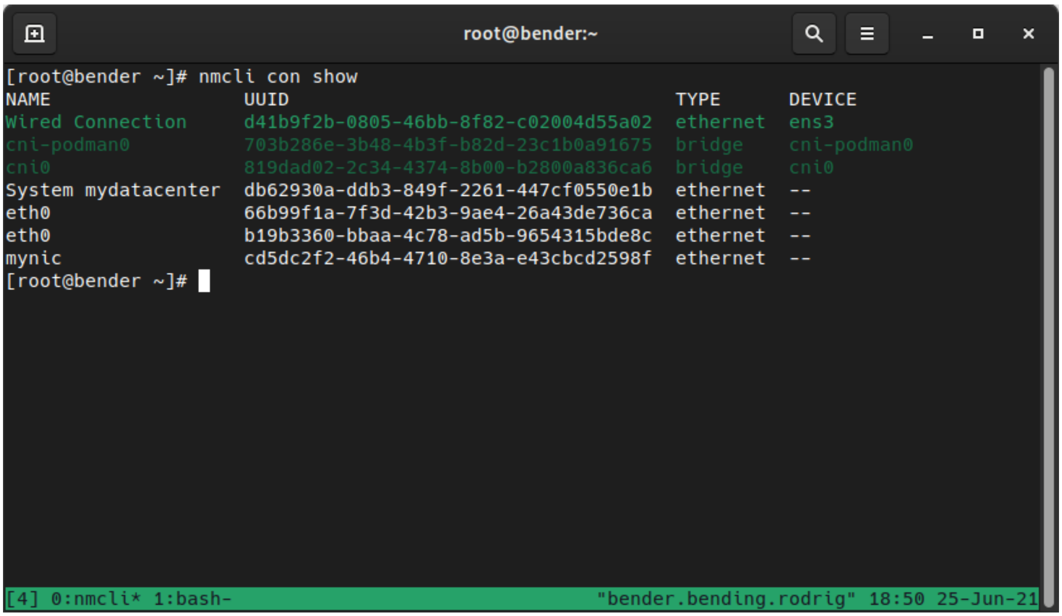


Figure 19.1 – Output of nmcli con show

With this, we can find what system interface is being used and connected. Let's say it's `ens3`, which is connected on the connection named `Wired Connection`.

Let's use `nmcli con show "Wired Connection"|grep address` to find the current addresses.

If our address is, for example, `10.0.0.6`, we can use the following code:

```
nmcli con mod "Wired Connection" ipv4.addresses 10.0.0.7
nmcli con mod "Wired Connection" ipv6.addresses
2001:db8:0:1::c000:207
```

Verify this with the following command:

```
nmcli con show "Wired Connection"|grep address
```

20. Create and add /opt/mysystem/bin/ to the system PATH for all users

Edit the `/etc/profile.d/mysystempath.sh` file and place the following contents:

```
export PATH=${PATH}:/opt/mysystem/bin
```

To validate this, add the `+x` attribute to the file and create the folder with the following commands:

```
chmod +x /etc/profile.d/mysystempath.sh
mkdir -p /opt/mysystem/bin
```

Relogging with the user should show the new path when executing the following command:

```
echo ${PATH}
```

21. Create a firewall zone, assign it to an interface, and make it the default zone

This is a tricky question. In this book, we've explained how to query zones and how to change the default one, and even shown screenshots of `cockpit` for managing the firewall, so now that you're an experienced user, this shouldn't be hard.

The first thing you need to do when you don't know how to do something is check the manual page:

```
man firewall-cmd
```

This doesn't show a lot of interesting information. However , toward the end of the man pages, there's a section called **SEE ALSO**, where we can find out about `firewalld.zones(5)`. This means that we can check section 5 of the manual for `firewalld.zones`.

We don't usually specify the section as there might not be a lot of duplicates, so we can just run the following command:

```
man firewalld.zones
```

This instructs us to check the default ones in `/usr/lib/firewalld/zones` and `/etc/firewalld/zones`, so let's do that:

```
cp /usr/lib/firewalld/zones/public.xml /etc/firewalld/zones/
dazone.xml
```

Now, let's edit the new copied file, called `/etc/firewalld/zones/dazone.xml`, and change its name from `Public` to `dazone`. Then, we need to reload the firewall:

```
firewall-cmd -reload
```

Let's validate that the new zone is there with the following command:

```
firewall-cmd --get-zones
```

Let's make it the default zone:

```
firewall-cmd --set-default-zone=dazone
```

Now, add the default interface (`ens3`):

```
firewall-cmd --add-interface=ens3 --zone=dazone
```

It will fail. This is expected since `ens3` has already been assigned to a zone (`public`). So, let's use the following commands:

```
firewall-cmd -remove-interface=ens3 --zone=public
firewall-cmd -add-interface=ens3 --zone=dazone
```

As you can see, even without prior knowledge about creating new zones, we've been able to use our system knowledge about finding information to accomplish this goal.

22. Add a repository hosted at https://myserver.com/repo/ with GPG key from https://myserver.com/mygpg.key to the system since our server might be down. Configure it so that it can be skipped if it's unavailable

If we don't remember the syntax for a repository, we can use one of the examples available on the system. To do this, go to `/etc/yum.repos.d/`, list the available files, and pick one to create a `myserver.repo` file with the following contents:

```
[myserver]
name=My server repository
baseurl=https://myserver.com/repo/
enabled=1
gpgcheck=1
gpgkey=https://myserver.com/mygpg.key
```

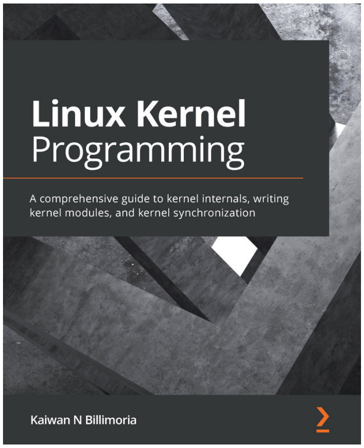
How do we skip it if it's unavailable? Let's check the man page for `yum`. Again, not much information is provided here, but in the **SEE ALSO** section, `man dnf.conf` is specified. This lists a Boolean that might help us, so let's add this to our `repofile`:

```
skip_if_unavailable=1
```

With that, we've completed our objectives.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



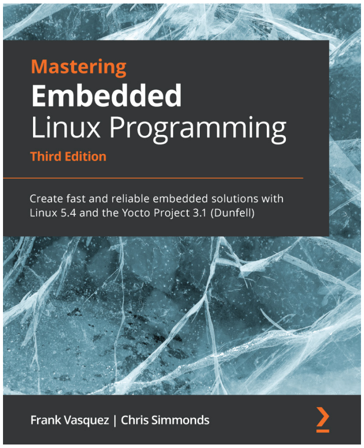
Linux Kernel Programming

Kaiwan N Billimoria

ISBN: 9781789953435

- Write high-quality modular kernel code (LKM framework) for 5.x kernels
- Configure and build a kernel from source
- Explore the Linux kernel architecture
- Get to grips with key internals regarding memory management within the kernel
- Understand and work with various dynamic kernel memory alloc/dealloc APIs

- Discover key internals aspects regarding CPU scheduling within the kernel
- Gain an understanding of kernel concurrency issues
- Find out how to work with key kernel synchronization primitives



Mastering Embedded Linux Programming - Third Edition

Frank Vasquez, Chris Simmonds

ISBN: 9781789530384

- Use Buildroot and the Yocto Project to create embedded Linux systems
- Troubleshoot BitBake build failures and streamline your Yocto development workflow
- Update IoT devices securely in the field using Mender or balena
- Prototype peripheral additions by reading schematics, modifying device trees, soldering breakout boards, and probing pins with a logic analyzer
- Interact with hardware without having to write kernel device drivers
- Divide your system up into services supervised by BusyBox runit
- Debug devices remotely using GDB and measure the performance of systems using tools such as perf, ftrace, eBPF, and Callgrind

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share Your Thoughts

Now you've finished *Red Hat Enterprise Linux 8 Administration*, we'd love to hear your thoughts! If you purchased the book from Amazon, please click here to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Index

D

E

F

G

H

I

J

K

L

O

P

S

T

Y