CHAPTER 1 PREPARING YOUR ENVIRONMENT

Before beginning your training, you need a learning space.

A learning space consists of Linux systems (virtual or physical), where you can actively try out, practice, and explore various Linux commands and utilities.

You may already have experience working with Linux in your enterprise environment. However, most likely you are using only one Linux distribution. Training with more than one distribution is needed.

In addition, your employer may frown upon any risky behaviour on its systems.
You need to feel free to try out Linux commands that may cause a system to crash.
Your own learning space, containing various Linux distributions and their assorted tools, is a key factor in successfully. This chapter begins by looking at a few items concerning the setup of your learning space environment. We will also explore various Linux distributions for your learning space. At the chapter's end, we'll cover a method for accessing the Linux command line.

SETTING UP A LEARNING SPACE

Your learning space needs to be an environment where you can freely explore Linux and its various distributions (called *distros* for short) and utilities. Whereas some companies may have a spare Linux server available for you to fully use, many of us are not so lucky. Even if you are a student, with a nice lab environment already set up and available for your use, you may want your own space, where you can explore without restrictions.
Though there are many different ways to set up your personal learning space, we will focus on only a few, such as setting up Linux on an old laptop, implementing a virtualized environment, and using the cloud. Hopefully the ideas here will spur you on to setting up a helpful exploration and study environment.

USING THAT OLD LAPTOP OR PC

If you've got a spare or old laptop sitting around, repurposing it as your Linux learning space may work well for you. This is especially useful if you like to move your study environment, such as, for example, moving to a different and quieter location in your home when things get a little loud and crazy. An old desktop will also work, but you will be less mobile. Whatever system you choose, you need to ensure that it has enough capacity to handle the minimum hardware requirements for a learning space.

CREATING A VIRTUALIZED ENVIRONMENT

Creating a virtualized environment for your Linux learning space is ideal.
This setting will allow you to boot multiple Linux distributions at the same time, enable you to move quickly between them, and provide compare and contrast experiences. In addition, you can explore networking utilities more thoroughly in such an environment. There are several excellent and free virtualization products (called *hypervisors* or *virtual machine managers*), which you can install. They include the following.
Oracle VirtualBox This actively developed open-source software is available at www.virtualbox.org . It can run on Linux, Windows, Macintosh, and even Solaris. You can use VirtualBox to run multiple Linux distributions at the same time, assuming your hardware has enough resources. The website is loaded with helpful documentation and has community forums to help you create your Linux learning space.

VMware Workstation Player VMware Workstation Pro is a proprietary closed-source virtualization product. VMware offers a free version called Workstation Player, which is available at https://www.vmware.com/products/workstation-player.html . This free version does have its limits. Workstation Player will only allow you to run a single virtual machine at time. Also, if you want to install it at your company's site, you must pay a fee to do so.

Microsoft Hyper-V This closed source virtualization product is available on many current Windows 64-bit versions, such as Windows 10 Professional and Enterprise. However, Windows 10 Home edition does not support it. You can use Hyper-V to run multiple Linux distributions at the same time, assuming your hardware has enough resources.

Please don't feel limited by this list. It includes only a few suggested hypervisors to investigate. If you have found a virtualization product that works better for your environment, use it for your learning space.
Prior to selecting and installing a particular hypervisor, determine if your laptop or chosen system has enough capacity to handle the entire learning space's minimum hardware requirements.

JUMPING TO THE CLOUD

If you do not own a laptop or desktop with enough resources to provide a multiple Linux distribution learning space consider the cloud.

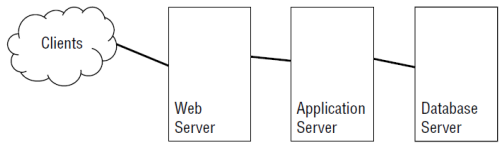
There are many cloud service providers where you can start up various Linux distribution virtual machines, such as Amazon Web Services (AWS), Microsoft Azure, and DigitalOcean.

1.1 UNDERSTANDING VIRTUALIZATION

The downside to cloud computing environments is that they're very computing intensive. A lot of computer power is required to run a cloud computing environment, and that can become costly.
The technology that made cloud computing possible is virtualization, and this is also what has made Linux a popular choice for cloud computing vendors. The following sections describe what virtualization is, the different types of virtualization available, and how to implement virtualization in a Linux environment.

HYPERVISORS

For organizations that run applications that support lots of clients, a standard performance model dictates that you should separate the different functions of an application onto separate servers, as shown in Figure 1.1.
FIGURE 1.1 SEPARATING APPLICATION RESOURCES

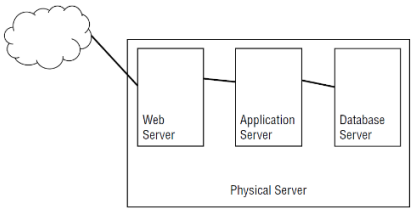


As shown in Figure 1.1, the application server, the web server, and the database server are located on separate servers.
Customers only communicate with the front-end web server. The web server passes the connections to the application, which in turn communicates with the database server.

From a performance standpoint, this model makes sense as you dedicate separate computing resources to each element. Also, from a security standpoint this helps compartmentalize access, making the job of any potential attackers a little more difficult.

However, with the increased capacity of servers, this model becomes somewhat inefficient. Dedicating an entire physical server to just running a web server, another physical server to just running the database server, and yet a third physical server to just running the application software doesn't utilize the full power of the servers and becomes costly.
This is where virtualization comes in. With virtualization, you can run multiple virtual smaller server environments on a single physical server. Figure 1.2 demonstrates this concept.

Figure 1.2 Server virtualization concept



Each virtual server operates as a stand-alone server running on the physical server hardware. This is called a *virtual machine*, or VM. None of the virtual servers interacts with each other, so they act just as if they were located on separate physical servers.
However, there needs to be a way for each virtual server to share the physical resources on the server fairly, so they don't conflict with one another. This is where the *hypervisor* comes into play.

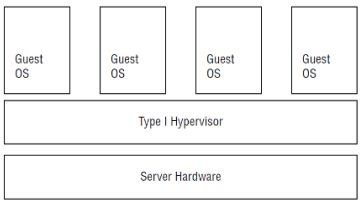
The hypervisor, also called a *virtual machine monitor (vmm)*, acts as the traffic cop for the physical server resources shared between the virtual machines. It provides a virtual environment of CPU time, memory space, and storage space to each virtual machine running on the server.

As far as each virtual machine is concerned, it has direct access to the server resources, and it has no idea that the hypervisor is in the middle controlling access to resources.
Since each virtual machine is a separate entity on the server, you can run different operating systems within the different virtual machines. This allows you to easily experiment with running applications in different operating systems, or just different versions of the same operating system. This is all without having to purchase additional servers.

TYPES OF HYPERVISORS
There are two different methods for implementing hypervisors. The following sections discuss what they are and how they differ.
TYPE I HYPERVISORS

Type I hypervisors are commonly called bare-metal hypervisors. The hypervisor system runs directly on the server hardware, with no middleman. The hypervisor software interacts directly with the CPU, memory, and storage on the system, allocating them to each virtual machine as needed. Figure 1.3 illustrates this setup.

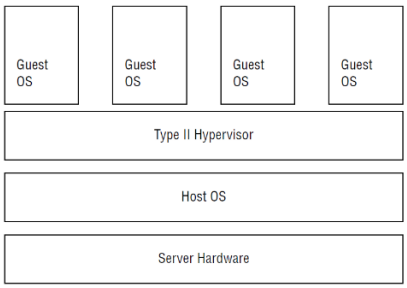
Figure 1.3 Type I hypervisors



In the Linux world, there are two popular Type I hypervisor packages used:
■ KVM: The Linux Kernel-based Virtual Machine (KVM) utilizes a standard Linux kernel along with a special hypervisor module, depending on the CPU used (Intel or AMD). Once installed, it can host any type of guest operating systems.
■ XEN: The XEN Project is an open-source standard for hardware virtualization. Not only does it support Intel and AMD CPUs, but there's also a version for Arm CPUs. The XEN Project includes additional software besides the hypervisor software, including an API stack for managing the hypervisor from a guest operating system.

TYPE II HYPERVISORS

Type II hypervisors are commonly called *hosted hypervisors* because they run on top of an existing operating system install. The hypervisor software runs like any other application on the host operating system. Figure 1.4 shows how a Type II hypervisor works.
Figure 1.4 Type II hypervisors



The Type II hypervisor software runs guest virtual machines as separate processes on the host operating system. The guest virtual machines support guest operating systems, which are completely separated from the host operating system. Thus, you can use a Linux host operating system and still run Windows or macOS guest operating systems.
The attraction of using a Type II hypervisor is that you can run it on an already installed operating system. You don't need to create a new server environment to run virtual machines. With the Type I hypervisors, you must dedicate a server to hosting virtual machines, while with a Type II hypervisor, your server can perform some (although not a lot) of other functions while it hosts virtual machines. There are many different popular Windows and macOS Type II hypervisors, such as VMware Workstation and QEMU, but for Linux the one commonly used is Oracle VirtualBox.

HYPERVISOR TEMPLATES
The virtual machines that you create to run in the hypervisor must be configured to determine the resources they need and how they interact with the hardware. These configuration settings can be saved to template files so that you can easily duplicate a virtual machine environment either on the same hypervisor or on a separate hypervisor server.
The open-source standard for virtual machine configurations is called the *Open Virtualization Format (OVF)*. The OVF format creates a distribution package consisting of multiple files. The package uses a single XML configuration file to define the virtual machine hardware environment requirements. Along with that file are additional files that define the virtual machine requirements for network access, virtual drive requirements, and any operating system requirements.
The downside to OVF templates is that they are cumbersome to distribute. The solution to that is the *Open Virtualization Appliance (OVA)* format. The OVA template bundles all of the OVF files into a single tar archive file for easy distribution.

EXPLORING CONTAINERS

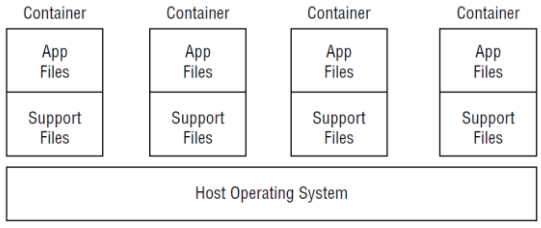
While utilizing virtual machines is a great way to spin up multiple servers in a server environment, they're still somewhat clunky for working with and distributing applications. There's no need to duplicate an entire operating system environment to distribute an application. The solution to this problem is containers. The following sections explore what containers are and how they are changing the way developers manage and distribute applications in the cloud environment.

WHAT ARE CONTAINERS?

Developing applications requires lots of files. The application runtime files are usually collocated in a single directory, but often additional library files are required for interfacing the application to databases, desktop management software, or built-in operating system functions. These files are usually located in various hard-to-find places scattered around the Linux virtual directory.
Because of all the ancillary files required to run an application, all too often an application will work just fine in development and then come crashing down when deployed to a production environment that doesn't accurately reproduce the development environment. In the Windows world, this is commonly referred to as DLL hell, as different applications overwrite common DLL library files, breaking other applications. However, this isn't limited to just the Windows world; it can also apply to the Linux world.

Containers are designed to solve this problem. A container gathers all of the files necessary to run an application, the runtime files, library files, database files, and any operating system, specific files. The container becomes self-sufficient for the application to run; everything the application needs is stored within the container. If you run multiple applications on a server, you can install multiple containers. Each container is still a self-contained environment for each particular application, as shown in Figure 1.5.
The application containers are portable. You can run the same container in any host environment and expect the same behavior for the application. This is ideal for application developers. The developer can develop the application container in one environment, copy it to a test environment, and then deploy the application container to a production environment, all without worrying about missing files.

FIGURE 1. 5 RUNNING AN APPLICATION IN A CONTAINER



By packaging and distributing an application as a container, the developer is ensured that the application will work for customers the same way it worked in the development environment.
Since containers don't contain the entire operating system, they're more lightweight than a full virtual machine, making them easier to distribute. The following sections describe two of the most common container packaging systems used in Linux.

CONTAINER SOFTWARE

Linux has been in the forefront of container development, making it a popular choice for developers. Two main container packages are commonly used in Linux:

■ *LXC* : The LXC package was developed as an open-source standard for creating containers. Each container in LXC is a little more involved than just a standard lightweight application container but not quite as heavy as a full virtual machine, placing it somewhere in the middle. LXC containers include their own bare-bones operating system that interfaces with the host system hardware directly, without requiring a host operating system. Because the LXC containers contain their own mini–operating system, they are sometimes referred to as *virtual machines,* although that term isn't quite correct as the LXC containers still require a host operating system to operate.

■ *Docker* : The Docker package was developed by Docker Incorporated and released as an open-source project. Docker is extremely lightweight, allowing several containers to run on the same host Linux system. Docker uses a separate daemon that runs on the host Linux system that manages the Docker images installed. The daemon listens for requests from the individual containers as well as from a Docker command-line interface that allows you to control the container environments.

CONTAINER TEMPLATES

Just like virtual machines, containers allow you to create templates to easily duplicate container environments. The different types of Linux containers utilize different methods for distributing templates. The LXC package uses a separate utility called LXD to manage containers. In recent versions, LXD has become so popular that it is now packaged itself as container software, although it still uses the LXC system images of the container. Docker uses Docker *container image* files to store container configurations. The container image file is a read-only container image that can store and distribute application containers.

NETWORK CONFIGURATIONS

Applications on physical systems are able to reach the outside world via a network interface card (NIC) and an attached network.

With virtualized systems and virtualized networks, the landscape is a little different. Virtualized machines can have any number of virtualized NICs, the hypervisor may provide virtualized internal switches, and the NIC configuration choices are plentiful. The right configuration results in higher network and application performance as well as increased security.

VIRTUALIZING THE NETWORK

Network virtualization has been evolving over the last few years. While it used to simply mean the virtualization of switches and routers running at OSI level 2 and 3, it can now incorporate firewalls, server load balancing, and more at higher OSI levels. Some cloud providers are even offering Network as a Service (NaaS).

Two basic network virtualization concepts are virtualized local area networks (VLANs) and overlay networks.

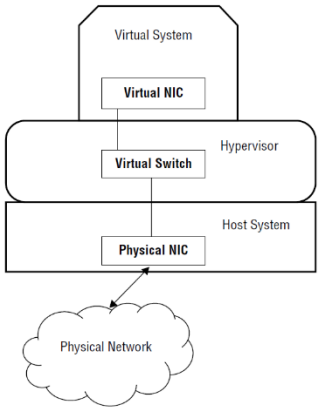
VLAN To understand a VLAN, it is best to start with a local area network (LAN) description. Systems and various devices on a LAN are typically located in a small area, such as an office or building. They share a common communications line or wireless link, are often broken up into different network segments, and their network traffic travels at relatively high speeds.

A VLAN consists of systems and various devices on a LAN, too. However, this group of systems and various devices can be physically located across various LAN subnets. Instead

of physical locations and connections, VLANs are based on logical and virtualized connections and use layer 2 to broadcast messages. Routers, which operate at layer 3, are used to implement this LAN virtualization.

CONFIGURING VIRTUALIZED NICS

Virtual NICs (adapters) are sometimes directly connected to the host system's physical NIC. Other times they are connected to a virtualized switch, depending on the configuration and the employed hypervisor. An example of a VM's adapter using a virtual switch is shown in Figure 1.6. Figure 1.6. Virtual machine using a virtual switch



When configuring a virtual machine's NIC, you have lots of choices. It is critical to understand your options in order to make the correct selections.

Host-Only A *host-only adapter* (sometimes called a *local adapter*) connects to a virtual network contained within the virtual machine's host system.

There is no connection to the external physical (or virtual) network to which the host system is attached.

The result is speed. If the host system has two or more virtual machines, the network speed between the VMs is rather fast. This is because VMs' network traffic does not travel along wires or through the air but instead takes place in the host system's RAM.

This configuration also provides enhanced security. A virtual proxy server is a good example. Typically, a proxy server is located between a local system and the Internet. Any web requests sent to the Internet by the local system are intercepted by the proxy server that then forwards them. It can cache data to enhance performance, act as a firewall, and provide privacy and security. One virtual machine on the host can act as a proxy server utilizing a different NIC configuration and have the ability to access the external network. The other virtual machine employing the host-only adapter sends/receives its web requests through the VM proxy server, increasing its protection.

Bridged A *bridged NIC* makes the virtual machine like a node on the LAN or VLAN to which the host system is attached. The VM gets its own IP address and can be seen on the network.

In this configuration, the virtual NIC is connected to a host machine's physical NIC. It transmits its own traffic to/from the external physical (or virtual) network.

This configuration is employed in the earlier virtual proxy server example. The proxy server's NIC is configured as a bridged host, so it can reach the external network and operate as a regular system node on the network.

NAT A *network address translation (NAT)* adapter configuration operates in a way that's similar to how NAT operates in the physical world. NAT in the physical networking realm uses a network device, such as a router, to "hide" a LAN computer system's IP address when that computer sends traffic out onto another network segment. All the other LAN systems' IP addresses are translated into a single IP address to other network segments. The router tracks each LAN computer's external traffic, so when traffic is sent back to that system, it is routed to the appropriate computer.

With virtualization, the NAT table is maintained by the hypervisor instead of a network device. Also, the IP address of the host system is employed as the single IP address that is sent out onto the external network. Each virtual machine has its own IP address within the host system's virtual network.
Physical and virtual system NAT has the same benefits. One of those is enhanced security by keeping internal IP addresses private from the external network.

1.2 EXPLORING LINUX DISTRIBUTIONS

in simple terms Linux Distribution is a collection of applications, packages, management, and features that run on top of the Linux kernel.
The kernel is what all distributions have in common, but at their core they all run Linux.

Distributions differ in several ways, and three of the most important are :
• Purpose
• Configuration and packaging
• Support model

First, different distributions are often designed for different purposes and provide different user experiences. Some distributions are designed as servers and others as desktops, and some are designed to perform particular functions.
The majority of Linux installations still tend to be servers. While more Linux desktops are appearing, the numbers do not yet challenge Windows and Apple OS X dominance of the desktop market.

The second major difference between distributions is in their configuration. While some distributions keep all their configuration settings and files in the same locations, others vary their locations.
Many distributions use different application installation and management tools (generally called *package management tools*). This can be confusing and can make administration difficult if you have an environment with differing distributions.

It is tempting to think that Linux distributions are all the same and few differences exist between them.

Unfortunately, this is a fallacy. We like to compare the Linux kernel to a car's engine and a distribution to a car's features.

If you have ever rented a car, the car's features are often rather different than the features of the car you normally drive.

When you get into the rented car, you have to take a few minutes to adjust the seat, view the various car controls, and figure out how to use them prior to taking off onto the roadway.
This is also true with learning new distributions. The good news is that if you have lots of previous experience with Linux, learning a new distribution is not that difficult.

RED HAT ENTERPRISE LINUX

Red Hat Enterprise Linux (www.redhat.com/rhel/) is a popular commercially supported Linux platform.
It comes in a number of versions, the two most common being Red Hat Enterprise Linux (also known as RHEL) and Red Hat Enterprise Linux Advanced Platform (RHELAP). The major difference between the versions is the number of CPUs (central processing units) supported, with RHEL supporting up to two CPUs and RHELAP supporting an unlimited number.
Red Hat, and most distributions based on it, make use of the Red Hat Package Management (RPM) packaging system.

CENTOS
Community Enterprise Operating System (CentOS) has been around since 2004.
CentOS (www.centos.org/) is a derivation of the Red Hat Enterprise Linux platform. Based on the same source code, it is available at no charge (and without Red Hat's support). People who wish to make use of the Red Hat platform and its stability without paying for additional support commonly use it. It employs the same packaging system, RPM, and many of the same administration tools as the Red Hat product.
Be aware that this distribution, like many others, comes in multiple flavors.

After you install your CentOS, you should update the software packages. Do this by logging in to the root account using the password you set up during installation and issuing the commands
sudo yum update

ROCKY

On December 8, 2020, Red Hat announced that they would discontinue development of CentOS, which had been a production-ready downstream version of Red Hat Enterprise Linux, in favor of a newer upstream development variant of that operating system known as "CentOS Stream".

That is ROCKY Linux.

THE FEDORA PROJECT
The Fedora Project (http://fedoraproject.org/) is a distribution jointly run by the community and Red Hat. It is a derivative of Red Hat Enterprise Linux and provides a forward development platform for the product. Sponsored by Red Hat, Fedora is a testing ground for many of Red Hat's new features. As a result, it is occasionally considered by some to be too edgy for commercial use. Many of the features introduced in Fedora often make their way into the new RHEL releases. Fedora also makes use of RPM packages and many of the same administration tools used by RHEL.

DEBIAN LINUX
The Debian Linux distribution (www.debian.org) is a free community-developed and community-managed distribution with a diverse and active group of developers and users. It was started in 1993 and built around a social contract (www.debian.org/social_contract). The Debian distribution strives toward freedom, openness, and maintaining a focus on delivering what users want.
The Debian distribution is well known for the *dpkg* packaging system and the availability of nearly 23,000 applications and tools for the distribution.

UBUNTU
Initiated by South African technologist and entrepreneur Mark Shuttleworth, the Ubuntu operating system (www.ubuntu.com/) is free and based on the Debian Linux platform. It is community developed, and upgrades are released on a six-month cycle. Commercial support is also available

from its coordinating organization, Canonical, as well as third-party support providers. It comes in different flavors to be used as desktops or servers.

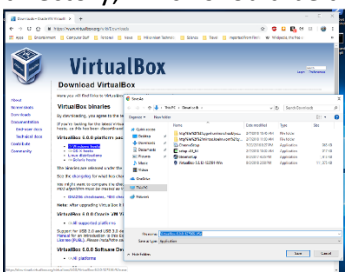
Many people consider Ubuntu one of the easiest Linux platforms to use and understand, and much of its development is aimed at ease of use and good user experience. Ubuntu makes use of Debian's packaging system and a number of its administration tools.

SO WHICH DISTRIBUTION SHOULD YOU CHOOSE?

Selecting a particular distribution should be based on your organization's budget, skills, and requirements.
Our broad recommendation, though, is that you choose either a Red Hat–derived distribution or Ubuntu (a Debian-based distribution) or Debian. All of these are well supported by the organizations and communities that maintain them.
Specifically, this course covers the material needed to implement applications and tools on
• Red Hat Enterprise Linux or a Red Hat–based distribution like CentOS or Fedora
• Ubuntu or other Debian-based distributions.

1.3 INSTALL VIRTUALBOX ON A WINDOWS HOST

1. Log in to your Windows host as an administrative user.
2. Install all current updates.
3. Open your browser.
4. Enter the following URL in the browser: https://www.virtualbox.org
5. Click the big large "Download VirtualBox" button in the middle of the screen to continue to the download page.
6. Locate the section heading **VirtualBox X.X.XX platform packages** where X.X.XX is the most current version of VirtualBox.
7. Locate the **Windows hosts** link and click that.
8. When the **Save as** window opens, as in Figure, ensure that the download target is the Downloads directory, which should be the default.



9. Click the **Save** button.
10. When the file has finished downloading, open the **File Explorer** and click **Downloads**.
11. Locate the VirtualBox installer and double-click to launch it.
12. When the setup wizard Welcome dialog shown in Figure
appears, click the **Next** button. This will take you to the Custom Setup dialog.

13. Do *not* make any changes to the **Custom Setup** dialog, and press **Next** to continue.

14. Again, do *not* make any changes on the second **Custom Setup** dialog, and press **Next** to continue.
15. If a dialog appears with a warning about resetting the network interfaces, just click **Yes** to continue.
16. When the **Ready to install** window is displayed, click **Install**.
17. You will see a dialog asking whether you want to allow this app to make changes to your device. Click **Yes** to continue.
18. When the completion dialog is displayed, remove the check from the box to start VirtualBox after the installation.
19. Click the **Finish** button to complete the basic installation. You should now have a shortcut on your desktop to launch VirtualBox. However we still need to install the "Extension Pack," which helps to integrate VMs more closely into the Windows desktop.
20. Use your browser to navigate to the URL: www.virtualbox.org/wiki/Downloads
21. Locate the section, **VirtualBox X.X.X Oracle VM VirtualBox Extension Pack**, and the link **All Platforms** under that.
22. When the file has finished downloading, open the **File Explorer** and click **Downloads**.
23. Locate the Oracle Extension Pack file, and double-click it to launch VirtualBox and install the Extension Pack.
24. When the dialog window titled **VirtualBox Question** is displayed, click **Install** to continue.
25. The license will be displayed in a dialog window. Scroll down to the bottom, and when the **I Agree** button is no longer grayed out, click it.
26. Once again, click **Yes** when the message asking if you want to allow this app to make changes. You will receive a verification dialog window when the Extension Pack software has been installed.
27. Click **OK** to close that dialog, and this leaves the VirtualBox Manager welcome window displayed on the screen.

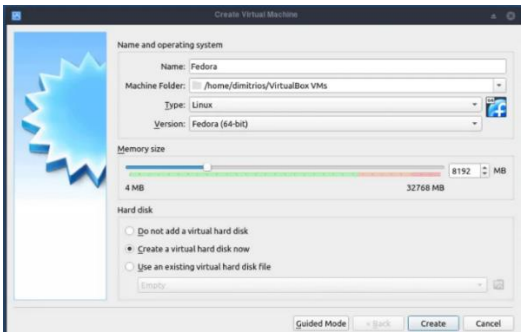
1.4 INSTALL FEDORA

I chose Fedora 34 with the GNOME desktop environment. (getfedora.org) : Fedora desktop.

Step 1: Create an empty virtual machine and configure it.
Fedora requires a minimum of 20GB disk space & 2GB RAM, to install and run successfully. Although double those amounts is recommended for a smoother user experience. Based on that I will create and configure the virtual machine.

Start Virtual Box and click on New.

The most important option to pay attention, is the type to be set to Linux and the version to Fedora (64-bit). If you start typing Fedora at the name prompt, VirtualBox will automatically pick the correct settings for you. Although the name doesn't have to be Fedora, it can be anything you like.Once you have similar settings with me, click on the create button.

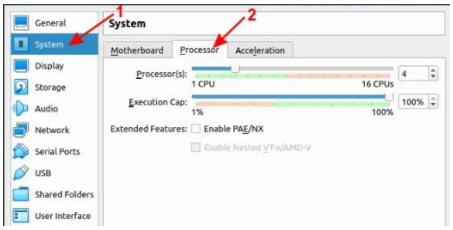


A word about RAM consumption, the RAM will only be consumed by the virtual machine when you are running it. Otherwise, it will be available for regular usage.

Before you click the start button of your virtual machine, you need to load the ISO as shown below [Optical Drive].

As your virtual hard drive is empty, the virtual machine will boot from this ISO. Think of it as using a live USB or disk for installing Linux.

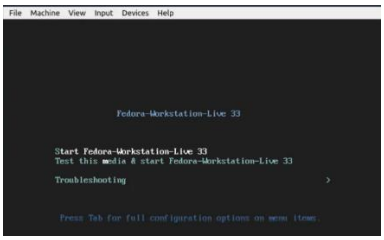
If you have a multi-core CPU it is recommended to assign 2 or more cores for your virtual machine. You may find the CPU cores under the system tab. When you configure your system click ok and start the virtual machine.



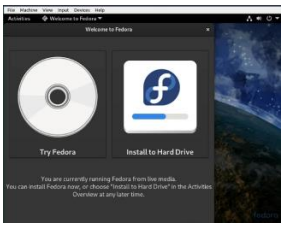
Once you have configured everything, click on the start button to begin the installation.

STEP 2: INSTALL FEDORA IN VIRTUALBOX

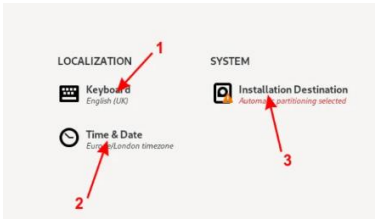
If you have followed the process correctly, when you start the virtual machine you will boot directly from the ISO file. When you see a similar to below screen select Start Fedora and press the enter key.



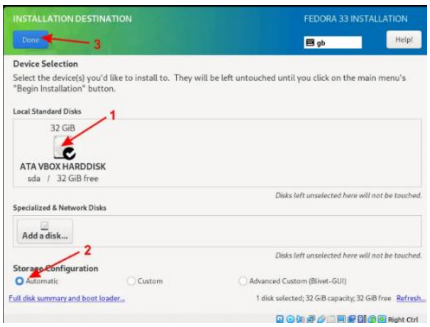
To start the installation dialog box, click on Install to Hard Drive.



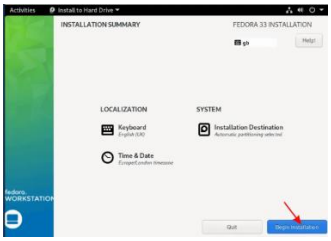
Before you proceed to the installation, it is essential to define your keyboard layout, your timezone and finally where the operating system will be installed.



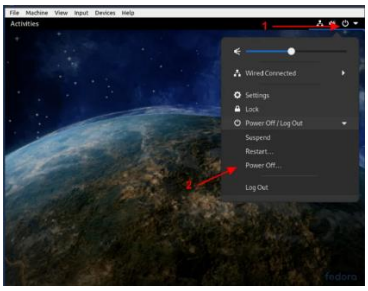
The partitioning process is straight forward. You made some free space as VDI earlier. It should be automatically recognized. Select your disk and set the storage configuration to automatic. Click on Done to go to the previous dialog box.



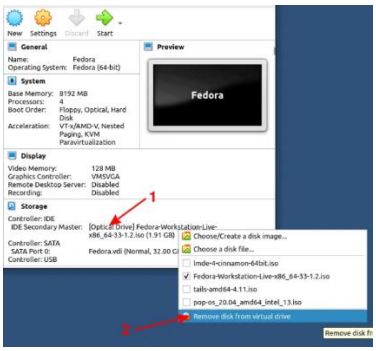
Once you have configured the above, click on “Begin Installation”.



Now you just need to wait for five-six minutes for installation completion. Click on the “Finish installation” button when installation is finished. As a last step, you need to power off your system.



You have to manually unload the ISO file that you loaded at the initial steps.



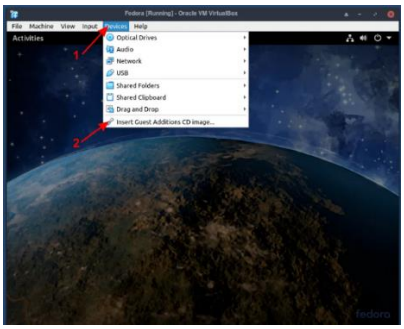
The next time you start the virtual machine with Fedora, you will be prompted to create a user account and set your password for Fedora Linux.

Use VirtualBox guest additions for additional features like clipboard sharing, folder sharing and more

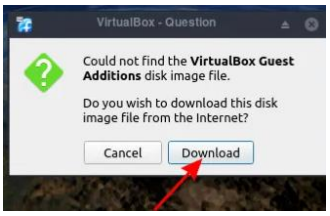
Guest Additions are designed to be installed inside a virtual machine post installation of the guest operating system. They contain device drivers and system applications that optimize the guest operating system for better performance and usability.

The Guest Additions ISO file is mounted as a virtual CD-ROM in order to be installed.

This is a straightforward process. Simply click on the devices tab and then click on “Insert Guest Additions CD image”



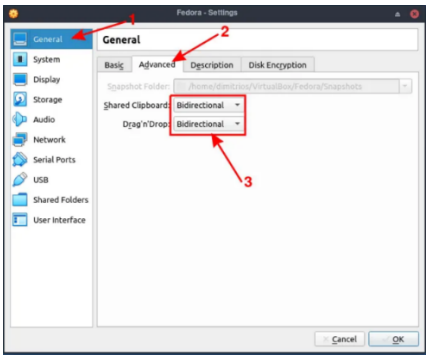
You will be prompted to download the guest additions image when prompt click on Download.



SHARED CLIPBOARD

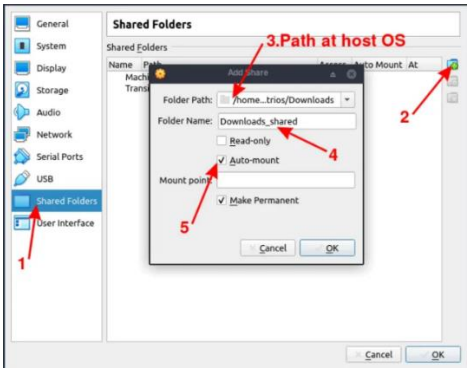
At some point you'll need to move some content between your virtual machine and the host operating system. The shared clipboard/drag and drop support will allow you to copy items on one platform and paste them on the other.

To enable this feature, choose Settings on the VirtualBox home page and follow the instructions as below. I find the Bidirectional option the most convenient.



SHARED FOLDERS

There are two types of shares:
-Permanent shares, that are saved with the Virtual Machine settings.
-Transient shares, that are disappear when the Virtual Machine is powered off. These can be created using a checkbox in the VirtualBox Manager.
Let make a permanent shared folder. In the VM settings add the host system folder you want to share and choose the name that you want to appear at your VM.



The next time you start the virtual machine, the folder should appear as a network drive.

USB AND NETWORK DEVICES SHARING

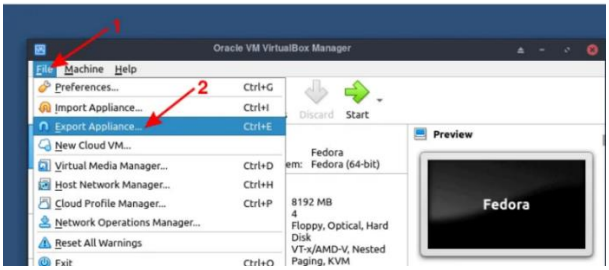
With VirtualBox, users have the ability to use a fully functional operating system, without having to do the setup on different hardware. However, sharing USB and network devices between the host and guest machine is not as straightforward as it should be.

To access USB devices, you will need to install the VirtualBox extension pack.

SAVE AND EXPORT THE VIRTUAL MACHINE SO THAT YOU CAN USE IT LATER ON ANY SYSTEM

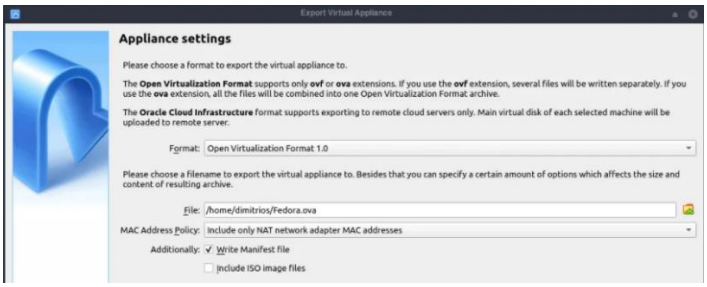
You may want to use your virtual machine to another computer, or it is time to build a new machine and you need to keep your virtual machine as it is. You can easily export your current setup and import it to another machine at a few easy steps.

At the VirtualBox home panel, click on file and the export appliance.



CHOOSE THE VIRTUAL MACHINE YOU WANT TO EXPORT AND CLICK ON NEXT.

The Format option needs some attention. There are three different options of Open Virtualization Format 0.9, 1.0 and 2.0, which can be either ovf or ova extensions.With ovf extension, several files will be written separately, compared to ova extension, which combines all the files into one Open Virtualization Format archive.The default format, Open Virtualization Format 1.0, should be fine.



1.5 LINUX BASICS

LOGGING IN

After your Linux host boots, you will be presented with a login prompt: either a command-line or GUI (graphical user interface) login prompt.

At login prompt you need to supply your username and a password. If you just installed a Linux host, you'll have been prompted to create a user, and you can make use of that user to log in now.

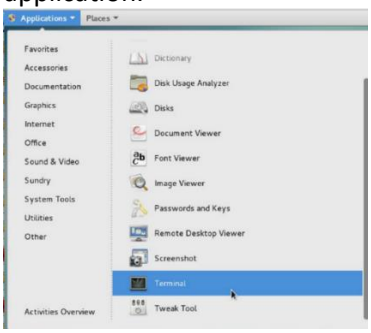
THE COMMAND LINE

In the Linux world, the command line is one of the most powerful tools available to you.

The command line can be referred to as the "console," "terminal," or "shell"; each has a slightly different context, but in the end, it is the place where you type Linux commands in.

In this course, a lot of focus is going to be on the command line. This is where at least some of your administration tasks are going to occur, and it's important to be able to understand and make use of the command line.

For example, you click the Applications menu, open the Accessories tab, and select the Terminal application.



SHELLS

What command line is presented to you depends on what shell is running for your user.
Shells are interfaces to the operating system and kernel of your host.
Each shell contains a collection of built-in commands that allow you to interact with your host.
A variety of shells are available, with the most common being the Bash (or Bourne-again) shell that is used by default on many distributions, including the popular Red Hat, Ubuntu, and Debian distributions. To find the shell you have on the default environment you can check the value of the SHELL environment variable: echo $SHELL. We're going to use the Bash shell for all our examples in this chapter, because it is most likely the shell you'll find by default. There exist other shells as well, and you can use them for your command line.
LIST OF SHELL :

1. Bash Shell
2. Tcsh/Csh Shell
3. Ksh Shell
4. Zsh Shell

These are not all the shells available in Unix/GNU Linux but they are the top most used apart from those that are already installed on different Linux distributions.

COMMAND-LINE PROMPT

After you have logged in to your Linux host, you should see a prompt that looks something like the following:
jm@centos-1 ~$

On most Linux distributions, the basic prompt is constructed from the username you're logged in as, the name of the host, the current directory, and the $ symbol, which indicates what sort of user you are logged in as.
In our case, jm is the name of the user we are logged in as; the @ symbol comes next and is followed by the name of the host we are logged into.
Next you see a ~ symbol, which is an abbreviated method of referring to your home directory. On a Linux host, most users have a special directory, called a *home directory*, which is created when the user is created.

Finally, you see the $ symbol. This symbol tells you what type of user you are; by default, all normal users on the host will have $ as their prompt.
There is a special user, called root, whose prompt uses the # symbol:
root@centos-1 ~#

THE ROOT USER IS THE SUPERUSER.

The root user can control and configure everything. So, if you see the # symbol, you know you are logged in as the root user.

In some distributions, you can log in as the root user, and you'll usually be prompted to specify a password for the root user during installation. Other distributions, most notably the Ubuntu distribution, disable the root user's password.

On Ubuntu, you are assumed to never use the root user, but rather a special command called **sudo.**

The sudo command allows you to run commands with the privileges of the root user without logging in as that user. To use the sudo command, you simply type **sudo** and the command you wish to run. You will usually be prompted for your password, and if you enter the correct password, the command will be executed.
$ sudo passwd root
This command would change the password of the root user, which is one method of enabling the root user on Ubuntu.

TYPING YOUR FIRST COMMAND

Now it's time to try entering a command.
Let's type a command called whoami and execute it by pressing the Enter key:
$ whoami
The whoami command returns the name of the user you are logged in as.
Each shell contains a series of built-in commands and functions to help you make use of the command line.
This time, though, we make a spelling error and type the wrong command name:
$ whoamii
We then press the Enter key to run the command and find that Bash has returned the following response:
-bash: whoamii: command not found

Bash is telling us that no command called whoami exists on the host. We can fix that. Let's start by correcting the command. We can bring back a previously typed command by using the up arrow key. Do that now, and you can see that the previous command has returned to the command line:
$ whoamii
Bash usefully has what is called *command history*, which keeps track of a number of the previous commands typed. Bash allows you to navigate through these commands by using the up and down arrow keys.
You can also move the cursor along the command line to edit commands using the left and right arrow keys. Move to the end of the command using the arrow keys and delete the extra i, leaving you with.
$ whoami.
Now press the Enter key, and you will see the result on the command line.
Another useful Bash feature is autocompletion. Start typing a command and then press the Tab key, and Bash will search your path to try to find the command you're trying to issue. Type more characters, and the Tab key will further narrow the search.

SERVICES AND PROCESSES

On a host, a lot of background activities and server applications run as services. Services can be started and stopped and often must be restarted when an application is reconfigured.
Services, also called daemons, run many of the key functions on your host.
Each service or daemon is one or more processes running on your host.

These processes have names. Some of these processes may be running by default on your host together with several other processes that perform a variety of system and application functions.

Most daemon processes usually have a name ending in "d" for daemon. We have the ps command with the -A flag (for all) to list all the processes currently running on our host.
$ ps -A

This list was generated using the ps command with the -A (or list all processes) option.

Each process running on the host is listed in order of its Process ID (PID), represented by the left-hand column. PIDs are used to control processes. The most important process on your host is called **systemd.**

The systemd process is the base process on Linux hosts that spawns all other processes on a host. This master process always uses PID 1 and must be running for your host to be functional.

Depending on the operating system you have chosen you might notice that you don't have a systemd process but an init process instead at PID 1. Systemd is a recent addition to mainstream Linux OS's, having been accepted in OSs like Fedora and Debian.
On systems that are older you will see init.

There is another useful command that can tell you which processes are running on your host and which are consuming the most CPU and memory. This command is called top.

$ top
The top command starts an interactive monitoring tool that updates every few seconds with the top running processes on your host.

Top refreshes every three minutes by default and processes will drift in and out of the listing as they consume resources. Processes that are heavy consumers of resources, like CPU, will always be listed at the top of the list, but you can also list by other resources like memory usage.

FILES AND FILESYSTEMS

In Unix, there is a phrase that says, "Everything in UNIX is either a file or a process." Linux also adheres to this statement. There are several types of files, but we are going to start with files and directories. In Linux directories are just files containing the names of other files. Let's take a closer look at Linux files and the filesystem. We're going to start by using a command called pwd, or print working directory.

$ pwd

The pwd command allows you to orient yourself in the filesystem by identifying our working or current directory. From here you can navigate the filesystem; start by changing the directory to the root directory using the cd, or change directory

user@host:~$ cd /

If you can ignore the user@host for a moment, you can see we've moved from our current directory to /, which is called the root directory.

The root directory is the base of the directory tree. The Linux filesystem is a single directory. This means that, unlike Windows, Linux has a single hierarchal directory structure. Instead of multiple drives, for example, C:\ and D:\, with separate directory trees beneath them, all drives, partitions, and storage are located off the root, or /, directory.

Linux drives and devices are mounted (this can occur automatically when you boot, or you can do it manually). These mounted drives appear in the filesystem as subdirectories under the '/'.
With the cd command, you can traverse to other directories and subdirectories.

Linux calls the steps you take to traverse the filesystem a path. There are two types of paths: **absolute and relative.**

The absolute path always starts with a slash symbol (/) representing the root directory and specifies the definitive location of the place you are describing; for example, /home/ntwali/ is an absolute path.
Relative paths allow you to specify a location relative to your current location or starting point.
For example, the command:
$ cd test
attempts to change from the current directory to a directory called test. If no such directory is present, the cd command fails.

There are also a couple of symbols that are often used with relative paths:

$ cd ..

The .. indicates that we wish to traverse up one level on the directory tree (if we're already at the top, we won't go anywhere at all).
We can also traverse in other ways through the directory tree using this mechanism, as you can see on the following line:
$ cd ../training/test

In this instance we have
1. Traversed up one directory level as indicated by the .. notation
2. Changed into a directory called traing in the next level up

3. Then changed into a directory called bar under the test directory

We can also refer to relative objects in a directory using the following construct:
$ ./make
The addition of the ./ in front of the command executes the make command in our current directory.

Which directories you can traverse to depends on their permissions. Many directories only allow access to specific users and groups (the root user can go anywhere). If you try to change to a directory to which you don't have suitable permissions, you will get an error message:
$ cd /root

Most Linux distributions adhere to a very similar directory structure. This is not to say all distributions are identical, but generally speaking, files and directories are located in a logical and consistent model.

Directory	Description
/bin/	User commands and binaries.
/boot/	Files used by the boot loader. (We talk about boot loaders in Chapter 6.)
/dev/	Device files.
/etc/	System configuration files.
/home/	User's home directories.
/lib/	Shared libraries and kernel modules.
/media/	Removable media is usually mounted here (see Chapter 8).
/mnt/	Temporary mounted filesystems are usually mounted here (see Chapter 8).
/opt/	Add-on application software packages.
/proc/	Kernel and process status data is stored in here in text-file format.
/root/	The root user's home directory.
/run/	A directory where applications can store data they require to operate.
/sbin/	System binaries.
/srv/	Data for services provided by this host.
/sys	Virtual filesystem that contains information and access to the Linux kernel subsystems.
/tmp/	Directory for temporary files.
/usr/	User utilities, libraries, and applications.
/var/	Variable or transient files and data, for example, logs, mail queues, and print jobs.

Let's look at some of the key directories under the root (/) directory.
The first, and one of the most important, is /etc/.

The /etc/ directory, is where most of the important configuration files on your host are located.

You'll be frequently working with files located in this directory as you add applications and services to your hosts.
Next, the /home/ directory contains all the home directories for users.
The /tmp directory is where you'll commonly find temporary files. In a similar vein is the /var directory, in which transitory data such as logs are stored.
You'll often look at log files contained in the /var/log/ directory that have been created by applications or via the host's syslog (or system logger) daemon. These log files contain a wide variety of information about the status of your applications, daemons, and services.

Let's take a closer look at files and directories and how to work with them.
Start by changing to the root, or /, directory:
$ cd /

Now you're at the root directory, and you want to see what is contained in that directory. To do this, you use the ls, or list directory, command.

$ ls

You can see the ls command has returned a list of files and directories that are in the root directory. By default, ls lists all files in a directory, but you can limit it to displaying a single file name or several file names by listing that file on the command line as follows:

$ ls test

This command would display any file or directory called test. We could also use the wildcard or asterisk symbol to select files.

$ ls te*

This would return any file called te plus any files that started with te, such as test, as well as the contents of any directories whose name starts with te. Specifying the asterisk symbol alone lists all files and all directories and their contents.

$ ls /usr/local/bin

This would list all the files in the /usr/local/bin directory.

You don't see a lot of details about these files and directories. It only shows a list of names.

To find out some more information about this list, you can add switches to the ls command.

$ ls -la

The l and a switch have been added to the ls command. The l switch, which is an abbreviation of long, uses a long listing format, which as you can see shows a lot more information.

The a switch tells ls to list all files and directories, even "hidden" ones, otherwise known as "dot" files.

So what does the long listing format tell you about your files and directories?

Each item has a small collection of information returned about it. You can see a subset of that listing showing one file and one directory, which we're going to examine in more detail.

Each line of the listing contains seven pieces of information about each object:

- Unix file type
- Permissions
- Number of hard links
- User and group ownership
- Size
- Date and time
- Name

Some of the information contained in the listing also introduces some key Linux concepts, such as permissions and users, groups, and ownership. We're going to take advantage of this introduction to not only explain each item but also explore some of the broader concepts they represent.

FILE TYPES AND PERMISSIONS

The file type and permissions are contained in the first ten characters, the section resembling -rw-r--r--.

This potentially intimidating collection of characters is actually quite simple to decipher: the first character describes the type of file, and the next nine characters describe the permissions of the file.

FILE TYPES

Almost everything on the Linux file system can be generally described as a file. The first character of the listing tells us exactly what sort of file. A dash (-) here indicates a regular file that might contain

data or text, or be a binary executable. A d indicates a directory, which is essentially a file that lists other files. An l indicates a symbolic link. Symbolic links allow you to make files and directories visible in multiple locations in the filesystem. They are much like the shortcuts used in Microsoft Windows. Table lists the file types available.

Type	Description
-	File
d	Directory
l	Link
c	Character devices
b	Block devices
s	Socket
p	Named pipe

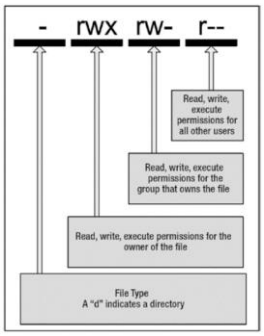
PERMISSIONS

The next nine characters detail the access permissions assigned to the file or directory. On Linux, permissions are used to determine what access users and groups have to a file. Controlling your permissions and access to files and applications is critical for security on your Linux host, and frequently in this book we'll use permissions to provide the appropriate access to files.
Thus, it is important that you understand how permissions work and how to change them.
There are three commonly assigned types of permissions for files:
• Read, indicated by the letter r
• Write, indicated by the letter w
• Execute, indicated by the letter x

Read permissions allow a file to be read or viewed but not edited. If it is set on a directory, the names of the files in the directory can be read, but other details, like their permissions and size, are not visible. Write permissions allow you to make changes or write to a file. If the write permission is set on a directory, you are able to create, delete, and rename files in that directory. Execute permissions allow you to run a file; for example, all binary files and commands (binary files are similar to Windows executables) must be marked executable to allow you to run them. If this permission is set on a directory, you are able to traverse the directory, for example, by using the cd command to access a subdirectory. The combination of the read and execute permissions set on a directory thus allows you to both traverse the directory and view the details of its contents.
Each file on your host has three classes of permissions:
• User
• Group
• Other (everyone else)

Each class represents a different category of access to the file. The User class describes the permissions of the user who owns the file. These are the first three characters in our listing. The Group class describes the permissions of the group that owns the file. These are the second set of three characters in our listing.

Finally, the Other class describes the permissions that all others have to the file. These are the final set of three characters in the listing.



-rw-r--r-- 1 root root 0 2021-06-15 20:47 myfile

We have a file, as indicated by the dash (-) at the beginning of the listing. The file is owned by the root user and root group. The first three permissions are rw-, which indicates the root user can read and write the file, but the dash means execute permissions are not set, and the file can't be executed by the user. The next three permissions, r--, indicate that anyone who belongs to the root group can read the file but can do nothing else to it. Finally, we have r-- for the last three permissions, which tell us what permissions the Other class has. In this case, others can read the file but cannot write to it or execute it.

Now you've seen what permissions look like, but how do you go about changing them? Permissions are changed using the chmod (change file mode bits) command. The key to changing permissions is that only the user who owns the file or the root user can change a file's permissions. So, only the root user could change the permissions of the myfile file.

The chmod command has a simple syntax. You can see some permissions being changed.

```
# chmod u+x myfile
# chmod u-x,og+w myfile
# chmod 654 myfile
```

We've changed the myfile file's permissions three times. Permission changes are performed by specifying the class, the action you want to perform, the permission itself, and then the file you want to change. In our first example, you can see u+x. This translates to adding the execute permission to the User class.

You can see the addition of the x to the User class. So how did chmod know to do that? Well, the u in our change represents the User class. With chmod, each class is abbreviated to a single letter:

- u: User
- g: Group
- o: Other or everyone
- a: All

After the class, you specify the action you'd like to take on the class. In the first line in Listing 4-11, the + sign represents adding a permission. You can specify the - sign to remove permissions from a class or the = sign to set absolute permissions on the class. Finally, you specify the permission to the action, in this case x.

You can also specify multiple permission changes in a single command, as you can see in the second line. In this second line, we have the change u-x,go+w. This would remove the x, or execute, permission from the User class and add the w, or write, permission to both the Group and Other classes. You can see we've separated each permission change with a comma and that we can list

multiple classes to act upon. (You can also list multiple permissions; for example, u+rw would add the read and write permissions to the User class.)
Thus the second line would leave our file permissions as -rw-rw-rw- 1 root root 0 2021-06-15 20:47 myfile.
With chmod, you can also use the a class abbreviation, which indicates an action should be applied to all classes; for example, a+r would add read permissions to all classes: User, Group, and Other.
We can also apply the permissions of one class to another class by using the = symbol.
chmod u=g myfile
On the previous line, we've set the User class permissions to be the same as the Group class permissions. You can also set permissions for multiple files by listing each file separated by space as follows:
chmod u+r file1 file2 file3
As with the ls command, you can also reference files in other locations as follows:
chmod u+x /usr/local/bin/foobar
The previous line adds the execute permission to the User class for the foobar file located in the /usr/local/bin directory. You can also use the asterisk symbol to specify all files and add the -R switch to recurse into lower
directories as follows:
chmod -R u+x /usr/local/bin/*
The chmod command on the previous line would add the execute permission to the User class to every file in the /usr/local/bin directory.
The last line is a little different. Instead of classes and permissions, we've specified a number, 654. This number is called *octal notation*. Each digit represents one of the three classes: User, Group, and Other. Additionally, each digit is the sum of the permissions assigned to that class.
You can see the values assigned to each permission type.

OCTAL PERMISSION VALUES

Permission	Value	Description
r	4	Read
W	2	Write
x	1	Execute

Each permission value is added together, resulting in a number ranging from 1 and 7 for each class. So the value of 654 would represent the following permissions:
The first value, 6, equates to assigning the User class the read permission with a value of 4 plus the write permission with a value of 2. The second value, 5, assigns the Group class the read permission with a value of 4 and the execute permission with a value of 1.The last value, 4, assigns only the read permission to the Other class. To make this clearer, you can see a list of the possible values from 0 to 7
THE OCTAL VALUES

Octal	permissions	Description
0	---	None
1	--x	Execute
2	-w-	Write
3	-wx	Write and execute
4	r--	Read
5	r-x	Read and execute
6	rw-	Read and write
7	rwx	Read, write, and execute

1.6 DEPLOYING BASH SCRIPTS

Linux system administrators often need to perform the same tasks over and over, such as checking available disk space on the system or creating user accounts. Instead of entering multiple commands every time, you can write scripts that run in the shell to do these tasks automatically for you. This chapter explores how bash shell scripts work and demonstrates how you can write your own scripts to automate everyday activities on your Linux system.

THE BASICS OF SHELL SCRIPTING

Shell scripting allows you to write small programs that automate activities on your Linux system. Shell scripts can save you time by giving you the flexibility to quickly process data and generate reports that would be cumbersome to do by manually entering multiple commands at the command prompt. You can automate just about anything you do at the command prompt using shell scripts.
The following sections walk through the basics of what shell scripts are and how to get started writing them.

RUNNING MULTIPLE COMMANDS

One exciting feature of the Linux command line is that you can enter multiple commands on the same command line and Linux will process them all. Just place a semicolon between each command you enter:

```
$ date ; who
```

The Linux bash shell runs the first command (date) and displays the output; then it runs the second command (who) and displays the output from that command immediately.

REDIRECTING OUTPUT

Another building block of shell scripting is the ability to store command output. Often, when you run a command, you'd like to save the output for future reference. To help with this, the bash shell provides output redirection.
Output redirection allows us to redirect the output of a command from the monitor to another device, such as a file. This feature comes in handy when you need to log data from a shell script that runs after business hours, so you can see what the shell script did when it ran.
To redirect the output from a command, you use the greater-than symbol (>) after the command and then specify the name of the file that you want to use to capture the redirected output. This is demonstrated in Listing 25.1.
Listing 25.1: Redirecting output to a file

```
$ date > today.txt
$ cat today.txt
```

The example shown in Listing 25.1 redirects the output of the date command to the file named today.txt. Notice that when you redirect the output of a command, nothing displays on the monitor output. All of the text from the output is now in the file, as shown by using the cat command to display the file contents.

The greater-than output redirection operator automatically creates a new file for the output, even if the file already exists. If you prefer, you can append the output to an existing

file by using the double greater-than symbol (>>), as shown in Listing 25.2.
Listing 25.2: Appending command output to a file
$ **who >> today.txt**
$ **cat today.txt**

The today.txt file now contains the output from the original date command in Listing 25.1 and the output from the who command ran in Listing 25.2.

Output redirection is a crucial feature in shell scripts. With it, we can generate log fi les from our scripts, giving us a chance to keep track of things as the script runs in background mode on the Linux system.

Piping Data
While output redirection allows us to redirect command output to a file, *piping* allows us to redirect the output to another command. The second command uses the redirected output from the first command as input data. This feature comes in handy when using commands that process data, such as the sort command.
The piping symbol is the bar (|) symbol, which usually appears above the backslash key on US keyboards. Listing 25.3 shows an example of using piping.
Listing 25.3: Piping command output to another command
$ **ls | sort**

The output from the ls command is sent directly to the sort command as input, but behind the scenes. You don't see the output from the ls command displayed on the monitor; you only see the output from the last command in the pipe line, which in this case is the sort command. There's no limit on how many commands you can link together with piping.

TEXT EDITORS

Text editors were developed for an environment where the text characters and strings were the only important aspect of the resulting file.

NANO

GNU nano is an easy to use command line text editor for Unix and Linux operating systems. It includes all the basic functionality you'd expect from a regular text editor.

To check if it is installed on your system type:

nano –version
Install Nano on Ubuntu and Debian

sudo apt install nano

Install Nano on CentOS and Fedora

sudo yum install nano

Opening and Creating Files

To open an existing file or to create a new file, type nano followed by the file name:

$nano filename

This opens a new editor window, and you can start editing the file.

At the bottom of the window, there is a list of the most basic command shortcuts to use with the nano editor.

All commands are prefixed with either ^ or M character. The caret symbol (^) represents the Ctrl key. For example, the ^J commands mean to press the Ctrl and J keys at the same time. The letter M represents the Alt key.

To open a file you must have read permissions to the file.

Saving and Exiting

To save the changes you've made to the file, press Ctrl+o. If the file doesn't already exist, it will be created once you save it.

To exit nano press Ctrl+x. If there are unsaved changes, you'll be asked whether you want to save the changes.

To save the file, you must have at write permissions to the file. If you are creating a new file , you need to have write permission to the directory where the file is created.

THE SHELL SCRIPT FORMAT

Placing multiple commands on a single line, by using either the semicolon or piping, is a great way to process data but can still get rather tedious. Each time you want to run the set of commands, you need to type them all at the command prompt.

However, Linux allows us to place multiple commands in a text file and then run the text file as a program from the command line. This is called a *shell script* because we're scripting out commands for the Linux shell to run.

Shell script files are plain-text files. To create a shell script file, you just need to use any text editor that you're comfortable with. If you're working from a KDE-based graphical desktop, you can use the KWrite program, or if you're working from a GNOME-based graphical desktop, you can use the GEdit program.

If you're working directly in a command-line environment, you still have some options. Many Linux distributions include either the pico or nano editor to provide a graphical editor environment by using ASCII control characters to create a full-screen editing window.

If your Linux distribution doesn't include either the pico or nano editor, there is still one last resort: the vi editor.
The vi editor is a text-based editor that uses simple single-letter commands. It's the oldest text editor in the Linux environment, dating back to the early days of Unix, which may be one reason it's not overly elegant or user-friendly.

Once you've chosen your text editor, you're ready to create your shell scripts. First, for your shell script to work you'll need to follow a specific format for the shell script file.
The first line in the file must specify the Linux shell required to run the script. This is written in somewhat of an odd format:
#!/bin/bash

The Linux world calls the combination of the pound sign and the exclamation symbol (#!) the *shebang* .
It signals to the operating system which shell to use to run the shell script.
Most Linux distributions support multiple Linux shells, but the most common is the bash shell. You can run shell scripts written for other shells as long as those shells are installed on the Linux distribution.

After you specify the shell, you're ready to start listing the commands in your script.
You don't need to enter all of the commands on a single line, Linux allows you to place them on separate lines. Also, the Linux shell assumes each line is a new command in the shell script, so you don't need to use semicolons to separate the commands. Listing 25.4 shows an example of a simple shell script file.
Listing 25.4: A simple shell script file

$ **cat test1.sh**
#!/bin/bash
This script displays the date and who's logged in
date
who

The test1.sh script file shown in Listing 25.4 starts out with the shebang line identifying the bash shell, the standard shell in Linux. The second line in the code shown in Listing 25.4 demonstrates another feature in shell scripts. Lines that start with a pound sign are called *comment lines* . They allow you to embed comments into the shell script program to help you remember what the code is doing. The shell skips comment lines when processing the shell script. You can place comment lines anywhere in your shell script file after the opening shebang line.

Running the Shell Script
If you just enter a shell script fi le at the command prompt to run it, you may be a bit disappointed:
$ **test1.sh**
test1.sh: command not found

Unfortunately, the shell doesn't know where to find the test1.sh command in the virtual directory.
The reason for this is the shell uses a special environment variable called PATH to list directories where it looks for commands. If your local HOME folder is not included in the PATH environment variable list of directories, you can't run the shell script fi le directly.

Instead, you need to use either a relative or an absolute path name to point to the shell script file. The easiest way to do that is by adding the ./ relative path shortcut to the file:
$ **./test1.sh**
bash: ./test1.sh: Permission denied

Now the shell can find the program file, but there's still an error message. This time the error is telling us that we don't have permissions to run the shell script file. A quick look at the shell script file using the ls command with the -l option shows the permissions set for the file:
$ **ls -l test1.sh**

By default, the Linux system didn't give anyone execute permissions to run the file. You can use the chmod command to add that permission for the file owner:
$ **chmod u+x test1.sh**

$ **ls -l test1.sh**

The u+x option adds execute privileges to the owner of the file. You should now be able to run the shell script file and see the output:
$ **./test1.sh**

Now that you've seen the basics for creating and running shell scripts, the next sections dive into some more advanced features you can add to make fancier shell scripts.