

Ubuntu 22.04 Essentials

Ubuntu 22.04 Essentials

© 2023 Neil Smyth / Payload Media, Inc. All Rights Reserved.

This book is provided for personal use only. Unauthorized use, reproduction and/or distribution strictly prohibited. All rights reserved.

The content of this book is provided for informational purposes only. Neither the publisher nor the author offers any warranties or representation, express or implied, with regard to the accuracy of information contained in this book, nor do they accept any liability for any loss or damage arising from any errors or omissions.

This book contains trademarked terms that are used solely for editorial purposes and to the benefit of the respective trademark owner. The terms used within this book are not intended as infringement of any trademarks.

Rev: 1.0a

Table of Contents

1. Introduction	1
1.1 Superuser Conventions.....	1
1.2 Opening a Terminal Window	2
1.3 Editing Files	3
1.4 Feedback.....	4
1.5 Errata.....	5
2. A Brief History of Ubuntu Linux	7
2.1 What exactly is Linux?.....	7
2.2 UNIX Origins	7
2.3 Who Created Linux?	7
2.4 The History of Ubuntu	8
2.5 What does the word “Ubuntu” Mean?	8
2.6 Summary	8
3. Installing Ubuntu on a Clean Disk Drive	9
3.1 Ubuntu Installation Options.....	9
3.2 Server vs. Desktop Editions	10
3.3 Obtaining the Ubuntu Installation Media	10
3.4 Writing the ISO Installation Image to a USB Drive.....	11
3.4.1 Linux.....	11
3.4.2 macOS	12
3.4.3 Windows/macOS.....	13
3.5 Booting from the Ubuntu USB Image.....	14
3.6 Installing Ubuntu	15
3.7 Accessing the Ubuntu Desktop	20
3.8 Installing Updates.....	21
3.9 Displaying Boot Messages.....	22
3.10 Summary	23
4. Dual Booting Ubuntu with Windows	25
4.1 Beginning the Ubuntu Installation	25
4.2 Booting Ubuntu for the First Time.....	31
4.3 Changing the Default Boot Option.....	31
4.4 Accessing the Windows Partition from the Command-line	32
4.5 Accessing the Windows Partition from the Desktop.....	33
4.6 Summary	36
5. Allocating Windows Disk Partitions to Ubuntu	37

Table of Contents

5.1 Unmounting the Windows Partition.....	37
5.2 Deleting the Windows Partitions from the Disk	37
5.3 Formatting the Unallocated Disk Partition	40
5.4 Mounting the New Partition	40
5.5 Editing the Boot Menu	41
5.6 Using GNOME Disks Utility	41
5.7 Summary	46
6. A Guided Tour of the GNOME 42 Desktop	47
6.1 Installing the GNOME Desktop	47
6.2 An Overview of the GNOME 42 Desktop	47
6.3 Launching Activities	49
6.4 Managing Windows	51
6.5 Using Workspaces	52
6.6 Calendar and Notifications	54
6.7 Desktop Settings	55
6.8 Customizing the Dock	56
6.9 Installing Ubuntu Software	56
6.10 Beyond Basic Customization	57
6.11 Summary	58
7. An Overview of the Cockpit Web Interface	59
7.1 An Overview of Cockpit.....	59
7.2 Installing and Enabling Cockpit.....	60
7.3 Accessing Cockpit	60
7.4 Overview	61
7.5 Logs	62
7.6 Storage.....	63
7.7 Networking.....	63
7.8 Accounts	64
7.9 Services	64
7.10 Applications	65
7.11 Virtual Machines	65
7.12 Software Updates.....	66
7.13 Terminal	66
7.14 Connecting to Multiple Servers	67
7.15 Enabling Stored Metrics	68
7.16 Summary	69
8. Using the Bash Shell on Ubuntu 22.04.....	71
8.1 What is a Shell?	71
8.2 Gaining Access to the Shell	71
8.3 Entering Commands at the Prompt.....	72

8.4 Getting Information about a Command	72
8.5 Bash Command-line Editing	72
8.6 Working with the Shell History	73
8.7 Filename Shorthand	74
8.8 Filename and Path Completion	74
8.9 Input and Output Redirection	74
8.10 Working with Pipes in the Bash Shell	75
8.11 Configuring Aliases	75
8.12 Environment Variables	76
8.13 Writing Shell Scripts	77
8.14 Summary	78
9. Managing Ubuntu 22.04 Users and Groups	79
9.1 User Management from the Command-line	79
9.2 User Management with Cockpit	81
9.3 User Management using the Settings App	83
9.4 Summary	85
10. Managing Ubuntu 22.04 systemd Units	87
10.1 Understanding Ubuntu systemd Targets	87
10.2 Understanding Ubuntu systemd Services	87
10.3 Ubuntu systemd Target Descriptions	87
10.4 Identifying and Configuring the Default Target	89
10.5 Understanding systemd Units and Unit Types	90
10.6 Dynamically Changing the Current Target	90
10.7 Enabling, Disabling, and Masking systemd Units	91
10.8 Working with systemd Units in Cockpit	92
10.9 Summary	94
11. Ubuntu Software Package Management and Updates	95
11.1 Repositories	95
11.2 Managing Repositories with Software & Updates	96
11.3 Managing Packages with APT	98
11.4 Performing Updates	99
11.5 Enabling Automatic Updates	100
11.6 Enabling Ubuntu Pro	102
11.7 Summary	104
12. Ubuntu Snap Package Management	105
12.1 Managing Software with Snap	105
12.2 Basic Snap Commands	106
12.3 Working with Snap Channels	108
12.4 Snap Refresh Schedule	109
12.5 Snap Services	111

Table of Contents

12.6 Summary	111
13. Ubuntu 22.04 Network Management.....	113
13.1 An Introduction to NetworkManager	113
13.2 Installing and Enabling NetworkManager.....	114
13.3 Basic nmcli Commands.....	114
13.4 Working with Connection Profiles	118
13.5 Interactive Editing.....	120
13.6 Configuring NetworkManager Permissions.....	122
13.7 Summary	122
14. Ubuntu 22.04 Firewall Basics.....	123
14.1 Understanding Ports and Services	123
14.2 Securing Ports and Services.....	123
14.3 Ubuntu Services and iptables Rules.....	124
14.4 Well-Known Ports and Services.....	125
14.5 Summary	128
15. Using gufw and ufw to Configure an Ubuntu Firewall	129
15.1 An Overview of gufw and ufw	129
15.2 Installing gufw on Ubuntu	129
15.3 Running and Enabling gufw	129
15.4 Creating a New Profile.....	130
15.5 Adding Preconfigured Firewall Rules.....	132
15.6 Adding Simple Firewall Rules.....	133
15.7 Adding Advanced Rules	134
15.8 Configuring the Firewall from the Command Line using ufw	135
15.9 Summary	137
16. Basic Ubuntu Firewall Configuration with firewalld	139
16.1 An Introduction to firewalld.....	139
16.1.1 Zones	139
16.1.2 Interfaces.....	141
16.1.3 Services.....	141
16.1.4 Ports.....	141
16.2 Checking firewalld Status	141
16.3 Configuring Firewall Rules with firewall-cmd.....	142
16.3.1 Identifying and Changing the Default Zone	142
16.3.2 Displaying Zone Information	142
16.3.3 Adding and Removing Zone Services.....	143
16.3.4 Working with Port-based Rules.....	144
16.3.5 Creating a New Zone.....	144
16.3.6 Changing Zone/Interface Assignments	144
16.3.7 Masquerading	144

16.3.8 Adding ICMP Rules	145
16.3.9 Implementing Port Forwarding.....	145
16.4 Managing firewalld using firewall-config	146
16.5 Summary	147
17. Configuring SSH Key-based Authentication on Ubuntu 22.04	149
17.1 An Overview of Secure Shell (SSH).....	149
17.2 SSH Key-based Authentication	149
17.3 Setting Up Key-based Authentication	150
17.4 Installing and Starting the SSH Service.....	150
17.5 SSH Key-based Authentication from Linux and macOS Clients.....	150
17.6 Managing Multiple Keys	152
17.7 SSH Key-based Authentication from Windows Clients	153
17.8 SSH Key-based Authentication using PuTTY	155
17.9 Generating a Private Key with PuTTYgen	156
17.10 Summary	157
18. Ubuntu 22.04 Remote Desktop Access with Vino	159
18.1 Remote Desktop Access Types	159
18.2 Secure and Insecure Remote Desktop Access	159
18.3 Enabling Remote Desktop Access on Ubuntu.....	160
18.4 Connecting to the Shared Desktop.....	162
18.5 Connecting from Windows	164
18.6 Summary	165
19. Displaying Ubuntu 22.04 Applications Remotely (X11 Forwarding).....	167
19.1 Requirements for Remotely Displaying Ubuntu Applications	167
19.2 Displaying an Ubuntu Application Remotely	168
19.3 Trusted X11 Forwarding	169
19.4 Compressed X11 Forwarding	169
19.5 Displaying Remote Ubuntu Apps on Windows.....	169
19.6 Summary	172
20. Using NFS on Ubuntu 22.04 to Share Files with Remote Systems	173
20.1 Ensuring NFS Services are running on Ubuntu	173
20.2 Configuring the Firewall to Allow NFS Traffic	173
20.3 Specifying the Folders to be Shared	174
20.4 Accessing Shared Folders	174
20.5 Mounting an NFS Filesystem on System Startup	175
20.6 Unmounting an NFS Mount Point	175
20.7 Accessing NFS Filesystems in Cockpit.....	175
20.8 Summary	177
21. Sharing Files between Ubuntu 22.04 and Windows with Samba.....	179

Table of Contents

21.1 Accessing Windows Resources from the GNOME Desktop.....	179
21.2 Samba and Samba Client.....	180
21.3 Installing Samba on Ubuntu.....	180
21.4 Configuring the Ubuntu Firewall to Enable Samba	180
21.5 Configuring the smb.conf File.....	181
21.5.1 Configuring the [global] Section.....	181
21.5.2 Configuring a Shared Resource	181
21.5.3 Removing Unnecessary Shares	182
21.6 Creating a Samba User	182
21.7 Testing the smb.conf File.....	182
21.8 Starting the Samba and NetBIOS Name Services	183
21.9 Accessing Samba Shares	184
21.10 Accessing Windows Shares from Ubuntu.....	186
21.11 Summary	188
22. An Overview of Virtualization Techniques	189
22.1 Guest Operating System Virtualization	189
22.2 Hypervisor Virtualization	190
22.2.1 Paravirtualization	191
22.2.2 Full Virtualization.....	192
22.2.3 Hardware Virtualization	192
22.3 Virtual Machine Networking.....	193
22.4 Summary	193
23. Installing KVM Virtualization on Ubuntu 22.04	195
23.1 An Overview of KVM	195
23.2 KVM Hardware Requirements.....	195
23.3 Preparing Ubuntu for KVM Virtualization	196
23.4 Verifying the KVM Installation.....	196
23.5 Summary	198
24. Creating KVM Virtual Machines on Ubuntu 22.04 using Cockpit.....	199
24.1 Installing the Cockpit Virtual Machines Module	199
24.2 Creating a Virtual Machine in Cockpit	199
24.3 Starting the Installation.....	201
24.4 Working with Storage Volumes and Storage Pools.....	203
24.5 Summary	205
25. Creating KVM Virtual Machines on Ubuntu 22.04 using virt-manager	207
25.1 Starting the Virtual Machine Manager.....	207
25.2 Configuring the KVM Virtual System	208
25.3 Starting the KVM Virtual Machine	212
25.4 Summary	213

26. Creating KVM Virtual Machines with virt-install and virsh	215
26.1 Running virt-install to build a KVM Guest System	215
26.2 An Example Ubuntu virt-install Command	215
26.3 Starting and Stopping a Virtual Machine from the Command-Line	217
26.4 Creating a Virtual Machine from a Configuration File.....	217
26.5 Summary	217
27. Creating an Ubuntu 22.04 KVM Networked Bridge Interface.....	219
27.1 Getting the Current Network Manager Settings.....	219
27.2 Creating a Network Manager Bridge from the Command-Line	221
27.3 Declaring the KVM Bridged Network	222
27.4 Using a Bridge Network in a Virtual Machine	223
27.5 Creating a Bridge Network using nm-connection-editor.....	225
27.6 Summary	228
28. Managing KVM using the virsh Command-Line Tool.....	229
28.1 The virsh Shell and Command-Line.....	229
28.2 Listing Guest System Status	230
28.3 Starting a Guest System	231
28.4 Shutting Down a Guest System	231
28.5 Suspending and Resuming a Guest System	231
28.6 Saving and Restoring Guest Systems	231
28.7 Rebooting a Guest System	232
28.8 Configuring the Memory Assigned to a Guest OS	232
28.9 Summary	232
29. An Introduction to Linux Containers.....	233
29.1 Linux Containers and Kernel Sharing.....	233
29.2 Container Uses and Advantages.....	234
29.3 Ubuntu Container Tools	235
29.4 The Ubuntu Docker Registry.....	235
29.5 Container Networking.....	236
29.6 Summary	236
30. Working with Containers on Ubuntu	237
30.1 Installing the Container Tools	237
30.2 Pulling a Container Image	237
30.3 Running the Image in a Container	239
30.4 Managing a Container	240
30.5 Saving a Container to an Image	241
30.6 Removing an Image from Local Storage	241
30.7 Removing Containers	241
30.8 Building a Container with Buildah	242

Table of Contents

30.9 Summary	242
31. Setting Up an Ubuntu 22.04 Web Server	243
31.1 Requirements for Configuring an Ubuntu Web Server	243
31.2 Installing the Apache Web Server Packages	243
31.3 Configuring the Firewall	244
31.4 Port Forwarding	244
31.5 Starting the Apache Web Server	244
31.6 Testing the Web Server	245
31.7 Configuring the Apache Web Server for Your Domain	245
31.8 The Basics of a Secure Website	247
31.9 Configuring Apache for HTTPS	248
31.10 Obtaining an SSL Certificate	248
31.11 Summary	251
32. Configuring an Ubuntu 22.04 Postfix Email Server	253
32.1 The Structure of the Email System	253
32.1.1 Mail User Agent	253
32.1.2 Mail Transfer Agent	253
32.1.3 Mail Delivery Agent	253
32.1.4 SMTP	254
32.1.5 SMTP Relay	254
32.2 Configuring an Ubuntu Email Server	254
32.3 Postfix Pre-Installation Steps	254
32.4 Firewall/Router Configuration.....	255
32.5 Installing Postfix on Ubuntu.....	255
32.6 Configuring Postfix	255
32.7 Configuring DNS MX Records	257
32.8 Starting Postfix on an Ubuntu System.....	257
32.9 Testing Postfix.....	257
32.10 Sending Mail via an SMTP Relay Server.....	258
32.11 Summary	259
33. Adding a New Disk Drive to an Ubuntu 22.04 System	261
33.1 Mounted File Systems or Logical Volumes	261
33.2 Finding the New Hard Drive	261
33.3 Creating Linux Partitions	262
33.4 Creating a File System on an Ubuntu Disk Partition	263
33.5 An Overview of Journaled File Systems.....	263
33.6 Mounting a File System	264
33.7 Configuring Ubuntu to Mount a File System Automatically	265
33.8 Adding a Disk Using Cockpit	265
33.9 Summary	267

34. Adding a New Disk to an Ubuntu 22.04 Volume Group and Logical Volume.....	269
34.1 An Overview of Logical Volume Management (LVM)	269
34.1.1 Volume Group (VG).....	269
34.1.2 Physical Volume (PV)	270
34.1.3 Logical Volume (LV)	270
34.1.4 Physical Extent (PE)	270
34.1.5 Logical Extent (LE)	270
34.2 Getting Information about Logical Volumes	270
34.3 Adding Additional Space to a Volume Group from the Command-Line	272
34.4 Summary	274
35. Adding and Managing Ubuntu Swap Space.....	275
35.1 What is Swap Space?	275
35.2 Recommended Swap Space for Ubuntu.....	275
35.3 Identifying Current Swap Space Usage	276
35.4 Adding a Swap File to an Ubuntu System.....	276
35.5 Adding Swap as a Partition	277
35.6 Adding Space to an Ubuntu LVM Swap Volume.....	277
35.7 Adding Swap Space to the Volume Group.....	279
35.8 Summary	280
36. Ubuntu 22.04 System and Process Monitoring.....	281
36.1 Managing Processes.....	281
36.2 Real-time System Monitoring with top.....	285
36.3 Command-Line Disk and Swap Space Monitoring.....	286
36.4 Summary	287
Index	289

1. Introduction

Ubuntu is arguably one of the most highly regarded and widely used Linux distributions available today. Praised both for its ease of use and reliability, Ubuntu also has a loyal following of Linux users and an active community of developers.

Ubuntu 22.04 Essentials is intended to provide detailed information on the installation, use, and administration of the Ubuntu distribution. For beginners, the book covers topics such as operating system installation, the basics of the GNOME desktop environment, configuring email and web servers, and installing packages and system updates. Additional installation topics, such as dual booting with Microsoft Windows, are also covered, together with all important security topics, such as configuring a firewall and user and group administration.

For the experienced user, topics such as remote desktop access, the Cockpit web interface, logical volume management (LVM), disk partitioning, swap management, KVM virtualization, Secure Shell (SSH), Linux Containers, and file sharing using both Samba and NFS are covered in detail to provide a thorough overview of this enterprise class operating system.

1.1 Superuser Conventions

Ubuntu, in common with Linux in general, has two types of user accounts, one being a standard user account with restricted access to many of the administrative files and features of the operating system and the other a superuser (*root*) account with elevated privileges. Typically, a user can gain root access either by logging in as the root user or using the *su* - command and entering the root password. In the following example, a user is gaining root access via the *su* - command:

```
[demo@demo-server ~]$ su -  
Password:  
[demo@demo-server ~]#
```

Note that the command prompt for a regular user ends with a \$ sign while the root user has a # character. When working with the command line, this is a useful indication of whether you are currently issuing commands as the root user.

If the *su* - command fails, the root account on the system has most likely been disabled for security reasons. In this case, the *sudo* command can be used instead, as outlined below.

Using *sudo*, a single command requiring root privileges may be executed by a non-root user. Consider the following attempt to update the operating system with the latest patches and packages:

```
$ apt update  
Reading package lists... Done  
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)
```

Introduction

Optionally, user accounts may be configured so that they have access to root-level privileges. Instead of using the `su` - command to first gain root access, user accounts with administrative privileges are able to run otherwise restricted commands using `sudo`:

```
$ sudo apt update
[sudo] password for demo:
Hit:1 http://us.archive.ubuntu.com/ubuntu bionic InRelease
.
```

To perform multiple commands without repeatedly using the `sudo` command, a command prompt with persistent super-user privileges may be accessed as follows:

```
[demo@demo-server]$ sudo su -
[demo@demo-server]#
```

The reason for raising this issue so early in the book is that many of the command-line examples outlined in this book will require root privileges. Rather than repetitively preface every command-line example with directions to run the command as root, the command prompt at the start of the line will be used to indicate whether or not the command needs to be performed as root. If the command can be run as a regular user, the command will be prefixed with a `$` command prompt as follows:

```
$ date
```

If, on the other hand, the command requires root privileges, the command will be preceded by a `#` command prompt:

```
# apt install openssh-server
```

1.2 Opening a Terminal Window

If you are using the GNOME desktop and need to access a command prompt, you will need to open a Terminal window. This can be achieved by right-clicking on the desktop background and selecting the Open in Terminal menu option as shown in Figure 1-1:

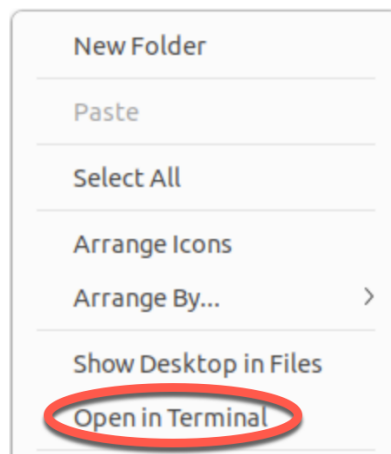


Figure 1-1

A terminal window may also be opened within the GNOME desktop using the Ctrl-Alt-T keyboard accelerator.

1.3 Editing Files

Configuring a Linux system typically involves editing files. For those new to Linux, it can be unclear which editor to use. If you are running a terminal session and do not already have a preferred editor, we recommend using the *nano* editor. To launch *nano* in a terminal window, enter the following command:

```
# nano <file>
```

Where <file> is replaced by the path to the file you wish to edit. For example:

```
# nano /etc/passwd
```

Once loaded, *nano* will appear as illustrated in Figure 1-2:

```
GNU nano 2.9.3 /etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nolog$
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr$

[ Read 44 lines ]

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify    ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell   ^_ Go To Line
```

Figure 1-2

To create a new file run *nano* as follows:

```
# nano
```

When you have finished editing the file, type Ctrl-S to save the file, followed by Ctrl-X to exit. To open an existing file, use the Ctrl-R keyboard shortcut.

If you prefer to use a graphical editor within the GNOME desktop environment, *gedit* is a useful starting point for basic editing tasks. To launch *gedit* from the desktop press Alt-F2 to display the Enter a Command window as shown in Figure 1-3:

Introduction



Figure 1-3

Enter *gedit* into the text field and press the Enter key. After a short delay, gedit will load ready to open, create, and edit files:

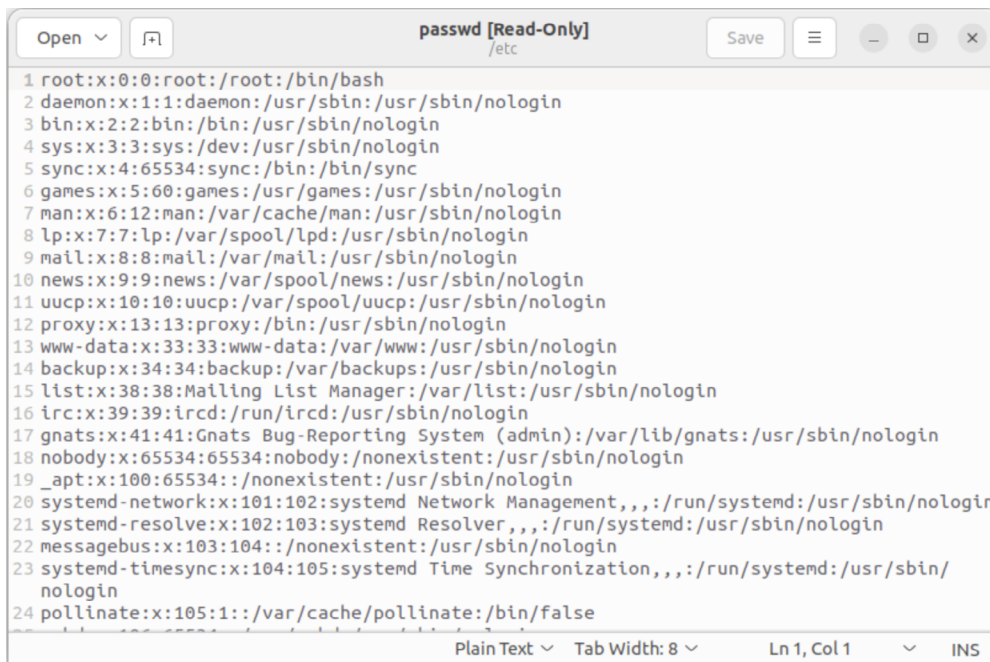


Figure 1-4

Alternatively, launch *gedit* from a terminal window either with or without the path to the file to open:

```
# gedit
# gedit /etc/passwd
```

1.4 Feedback

We want you to be satisfied with your purchase of this book. If you find any errors in the book or have any comments, questions, or concerns, please contact us at feedback@ebookfrenzy.com.

1.5 Errata

While we make every effort to ensure the accuracy of the content of this book, it is inevitable that a book covering a subject area of this size and complexity may include some errors and oversights. Any known issues with the book will be outlined, together with solutions, at the following URL:

<https://www.ebookfrenzy.com/errata/Ubuntu2204.html>

In the event that you find an error not listed in the errata, please let us know by emailing our support team at *feedback@ebookfrenzy.com*.

2. A Brief History of Ubuntu Linux

Ubuntu Linux is one of a number of variants (also referred to as *distributions*) of the Linux operating system and is the product of a U.K. company named Canonical Ltd. The company was founded in 1994 by Mark Shuttleworth. The origins of Linux, however, go back even further. This chapter will outline the history of both the Linux operating system and Ubuntu.

2.1 What exactly is Linux?

Linux is an operating system in much the same way that Windows is an operating system (and there are any similarities between Linux and Windows end). The term operating system is used to describe the software that acts as a layer between the hardware in a computer and the applications that we all run on a daily basis. When programmers write applications, they interface with the operating system to perform such tasks as writing files to the hard disk drive and displaying information on the screen. Without an operating system, every programmer would have to write code to access the hardware of the system directly. In addition, the programmer would have to be able to support every single piece of hardware ever created to be sure the application would work on every possible hardware configuration. Because the operating system handles all of this hardware complexity, application development becomes a much easier task. Linux is just one of a number of different operating systems available today.

2.2 UNIX Origins

To understand the history of Linux, we first have to go back to AT&T Bell Laboratories in the late 1960s. During this time, AT&T had discontinued involvement in developing a new operating system named Multics. However, two AT&T engineers, Ken Thompson, and Dennis Ritchie, decided to take what they had learned from the Multics project and create a new operating system named UNIX which quickly gained popularity and wide adoption both with corporations and academic institutions.

A variety of proprietary UNIX implementations eventually came to market, including those created by IBM (AIX), Hewlett-Packard (HP-UX), and Sun Microsystems (SunOS and Solaris). In addition, a UNIX-like operating system named MINIX was created by Andrew S. Tanenbaum and designed for educational use with source code access provided to universities.

2.3 Who Created Linux?

The origins of Linux can be traced back to the work and philosophies of two people. At the heart of the Linux operating system is something called the *kernel*. This is the core set of features necessary for the operating system to function. The kernel manages the system's resources and handles communication between the hardware and the applications. The Linux kernel was developed by Linus Torvalds, who, taking a dislike to MS-DOS and impatient for the availability of MINIX for the new Intel 80386 microprocessor, decided to write his own UNIX-like kernel. When he had

A Brief History of Ubuntu Linux

finished the first version of the kernel, he released it under an open-source license that enabled anyone to download the source code and freely use and modify it without having to pay Linus any money.

Around the same time, Richard Stallman at the Free Software Foundation, a strong advocate of free and open-source software, was working on an open-source operating system of his own. Rather than focusing initially on the kernel, however, Stallman began by developing open-source versions of all the UNIX tools, utilities, and compilers necessary to use and maintain an operating system. By the time he had finished developing this infrastructure, the obvious solution was to combine his work with the kernel Linus had written to create a complete operating system. This combination became known as GNU/Linux. Purists insist that Linux always be referred to as GNU/Linux (in fact, at one time, Richard Stallman refused to give press interviews to any publication which failed to refer to Linux as GNU/Linux). This is not unreasonable, given that the GNU tools developed by the Free Software Foundation make up a significant and vital part of GNU/Linux. Unfortunately, most people and publications refer to Linux as Linux, which will probably always continue to be the case.

2.4 The History of Ubuntu

As mentioned previously, Ubuntu is one of a number of Linux distributions. The source code that makes up the Ubuntu distribution originates from a highly regarded Linux distribution known as Debian, created by Ian Murdoch.

A South African internet mogul named Mark Shuttleworth (who made his fortune selling his company to VeriSign for around \$500 million) decided it was time for a more user-friendly Linux. He took the Debian distribution and worked to make it a more human-friendly distribution which he called Ubuntu. He subsequently formed a company called Canonical Ltd to promote and provide support for Ubuntu.

If you are new to Linux or already use Linux and want to try a different Linux distribution, it is unlikely you will find a better option than Ubuntu.

2.5 What does the word “Ubuntu” Mean?

The word “Ubuntu” is an ancient Zulu and Xhosa word that means “humanity to others”. Ubuntu also means “I am what I am because of who we all are”. It was chosen because these sentiments precisely describe the spirit of the Ubuntu distribution.

2.6 Summary

The origins of the Linux operating system can be traced back to the work of Linus Torvalds and Richard Stallman in the form of the Linux kernel combined with the tools and compilers built by the GNU project.

Over the years, the open-source nature of Linux has resulted in the release of a wide range of different Linux distributions. One such distribution is Ubuntu, based on the Debian Linux distribution created by Canonical Ltd, a company founded by Mark Shuttleworth

3. Installing Ubuntu on a Clean Disk Drive

There are now three ways in which an Ubuntu system can be deployed. One method is to either purchase new hardware or re-purpose an existing computer system on which to install and run the operating system. Alternatively, a virtualization platform such as VirtualBox or VMware can be used to install and run Ubuntu inside a virtual machine on an existing operating system. Another option is to create a cloud-based operating system instance using services such as Amazon AWS, Google Cloud, or Microsoft Azure (to name but a few). Since cloud-based instances are typically created by selecting a pre-configured, ready to run operating system image that is already optimized for the cloud platform and using that as the basis for the Ubuntu system, there is no need to perform a manual operating system installation in this situation.

If, on the other hand, you plan to install Ubuntu on your own hardware or make use of a virtualization environment, the first step on the path to learning about Ubuntu involves installing the operating system.

Ubuntu can be installed either in a clean disk environment (where an entire disk is cleared of any existing partitions and dedicated entirely to Ubuntu) or in a dual boot environment where Ubuntu co-exists with another operating system on the disk (typically a member of the Microsoft Windows family of operating systems).

In this chapter, we will cover the clean disk approach to installation from local or remote installation media. Dual boot installation with a Windows 10 system will be covered in *"Dual Booting Ubuntu with Windows"*.

3.1 Ubuntu Installation Options

Ubuntu can be downloaded free of charge from the following web page:

<https://ubuntu.com/download>

This page provides a number of download options depending on how the operating system is to be installed and used:

- **Ubuntu Desktop** - Downloads the installation media for the desktop edition of the operating system. This edition is intended for use on desktop and laptop systems where a graphical desktop environment is needed and is only available for 64-bit x86 systems. The desktop edition can be downloaded in the form of an ISO image which you can then write to a USB drive using the steps outlined later in this chapter. When booted, the desktop media will allow you to test out Ubuntu by running a Live Ubuntu session prior to performing the installation.

Installing Ubuntu on a Clean Disk Drive

- **Ubuntu Server** - Downloads the Live Server ISO installation media for the server edition of the operating system. This image is intended for performing an installation on servers on which the graphical desktop environment is not required and is available for x86, ARM, IBM POWER (PowerPC), and s390x (IBM System z mainframe) systems. The installation media does not include the option to try Ubuntu before installing and uses the text-based installer instead of the graphical installer used for Ubuntu Desktop. This allows Ubuntu to be installed on systems without a graphical console.

The Ubuntu Server image may also be used to perform Preboot Execution Environment (PXE) network installations. When using PXE to install Ubuntu, the Ubuntu image is installed on a specially configured server (referred to as a PXE boot server). The client system on which Ubuntu is to be installed is then configured to boot over the network from the image on the PXE boot server (assuming the client hardware supports PXE) to initiate the installation.

3.2 Server vs. Desktop Editions

Clearly, a decision between the Desktop and the Server Edition images needs to be made before installation can begin. If you would like to try Ubuntu before installing it, then the Desktop option is the best solution since it allows you to boot Ubuntu from the installation media without first installing it on a disk drive. This option also allows the installation to be initiated from within the live session.

If the graphical desktop environment is not required, and the destination system does not have internet access or a graphical console, then the Live Server ISO image is recommended since this allows a fully functional server to be built without the need to download any additional packages.

Regardless of the chosen installation method, packages can be added to and removed from the system after installation to configure the system to specific needs.

3.3 Obtaining the Ubuntu Installation Media

For the purposes of this chapter, the Ubuntu Desktop environment will be installed using the graphical installer. Begin, therefore, by downloading the Ubuntu Desktop 22.04 ISO image from the following URL:

<https://ubuntu.com/download/desktop>

The DVD ISO image is self-contained, including all of the packages necessary to install an Ubuntu system, and is named using the following convention:

`ubuntu-<version>-<edition>-<architecture>.iso`

For example, the Ubuntu 22.04 Desktop ISO image for 64-bit Intel/AMD systems is named as follows:

`ubuntu-22.04-desktop-amd64.iso`

Having downloaded the image, either burn it to disk or use the steps in the next section to write the media to a USB drive and configure your virtualization environment to treat it as a DVD drive.

3.4 Writing the ISO Installation Image to a USB Drive

These days it is more likely that an operating system installation will be performed from a USB drive than from a DVD. Having downloaded the ISO installation image for Ubuntu, the steps to write that image to a USB drive will differ depending on whether the drive is attached to a Linux, macOS, or Windows system. The steps outlined in the remainder of this section assume that the USB drive is new or has been reformatted to remove any existing data or partitions:

3.4.1 Linux

The first step in writing an ISO image to a USB drive on Linux is identifying the device name. Before inserting the USB drive, identify the storage devices already detected on the system by listing the devices in */dev* as follows:

```
# ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sda2
```

Attach the USB drive to the Linux system and run the *dmesg* command to get a list of recent system messages, one of which will be a report that the USB drive was detected and will be similar to the following:

```
[445597.988045] sd 6:0:0:0: [sdb] Attached SCSI removable disk
```

This output tells us that we should expect the device name to include “sdb” which we can confirm by listing device names in */dev* again:

```
# ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sda2  /dev/sdb
```

From this output, we can tell that the USB drive has been assigned to */dev/sdb*. The next step before writing the ISO image to the device is to run the *findmnt* command to make sure it has not been auto-mounted:

```
# findmnt /dev/sdb?
TARGET                SOURCE      FSTYPE OPTIONS
/media/demo/C24E-6727 /dev/sdb1  vfat     rw,nosuid,nodev, ...
```

If the *findmnt* command indicates that the USB drive has been mounted, unmount it before continuing:

```
# umount /media/demo/C24E-6727
```

Once the filesystem has been unmounted, use the *dd* command as follows to write the ISO image to the drive:

```
# dd if=/path/to/iso/<image name>.iso of=/dev/sdb bs=512k
```

The writing process can take some time (as long as 10 - 15 minutes) to complete depending on the image size and speed of the system on which it is running. Once the image has been written, output similar to the following will appear and the USB drive is ready to be used to install Ubuntu:

```
4056+1 records in
4056+1 records out
2126544896 bytes (2.1 GB, 2.0 GiB) copied, 625.911 s, 3.4 MB/s
```

Installing Ubuntu on a Clean Disk Drive

3.4.2 macOS

The first step in writing an ISO image to a USB drive attached to a macOS system is to identify the device using the *diskutil* tool. Before attaching the USB device, open a Terminal window and run the following command:

```
$ diskutil list
/dev/disk0 (internal, physical):
#:                                TYPE NAME                                SIZE      IDENTIFIER
0:      GUID_partition_scheme      *1.0 TB    disk0
1:      EFI EFI                    209.7 MB   disk0s1
2:      Apple_APFS Container disk2  1000.0 GB  disk0s2

/dev/disk1 (internal):
#:                                TYPE NAME                                SIZE      IDENTIFIER
0:      GUID_partition_scheme      28.0 GB    disk1
1:      EFI EFI                    314.6 MB   disk1s1
2:      Apple_APFS Container disk2  27.7 GB    disk1s2

/dev/disk2 (synthesized):
#:                                TYPE NAME                                SIZE      IDENTIFIER
0:      APFS Container Scheme -      +1.0 TB    disk2
      Physical Stores disk1s2, disk0s2
1:      APFS Volume Macintosh HD     473.6 GB   disk2s1
2:      APFS Volume Preboot           42.1 MB    disk2s2
3:      APFS Volume Recovery          517.0 MB   disk2s3
4:      APFS Volume VM                1.1 GB     disk2s4
```

Having established a baseline of detected devices, insert the USB drive into a port on the macOS system and run the command again. The same results should appear with one additional entry for the USB drive resembling the following:

```
/dev/disk3 (external, physical):
#:                                TYPE NAME                                SIZE      IDENTIFIER
0:                                *16.0 GB    disk3
```

In the above example, the USB drive has been assigned to `/dev/disk3`. Before proceeding, unmount the disk as follows:

```
$ diskutil unmountDisk /dev/disk3
Unmount of all volumes on disk3 was successful
```

Finally, use the *dd* command to write the ISO image to the device, taking care to reference the raw disk device (`/dev/rdisk3`) and entering your user password when prompted:

```
$ sudo dd if=/path/to/iso/image.iso of=/dev/rdisk3 bs=1m
```

Once the image has been written, the USB drive is ready.

3.4.3 Windows/macOS

Several free tools are available for Windows and macOS that will write an ISO image to a USB drive, but one written specifically for writing Linux ISO images is the Fedora Media Writer tool which can be downloaded from the following URL:

<https://getfedora.org/en/workstation/download/>

Once installed, insert the destination USB drive, launch the writer tool, and choose the *Select .iso file* option as highlighted in Figure 3-1:

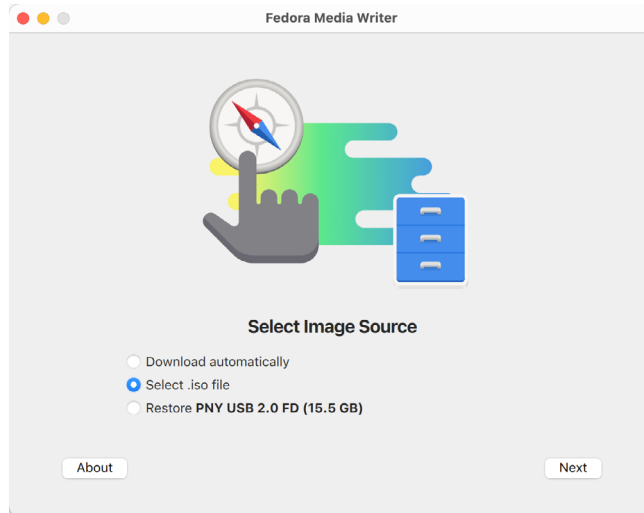


Figure 3-1

Click Next to proceed to the Write Options screen and select the USB Drive before clicking on the *Select...* button:

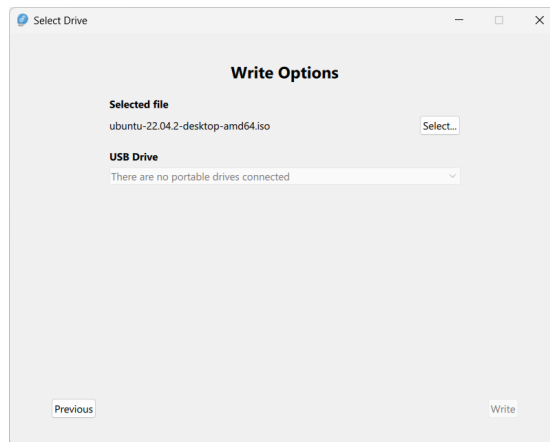


Figure 3-2

In the resulting file selection dialog, navigate to and select the Ubuntu installation ISO image and

Installing Ubuntu on a Clean Disk Drive

click the *Open* button. Finally, click the Write button to start writing the image to the USB drive:

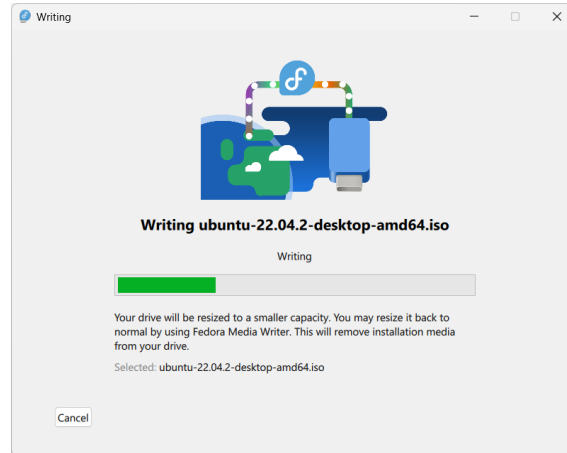


Figure 3-3

Once the image has been written, the device is ready to perform the installation.

3.5 Booting from the Ubuntu USB Image

Insert the Ubuntu installation media into the appropriate drive and power on the system. If the system tries to boot from the hard disk drive, you will need to enter the BIOS set up for your computer and change the boot order so that it boots from the installation media drive first. After the initial boot sequence completes, the GRUB menu shown in Figure 3-4 will appear:



Figure 3-4

Use the arrow keys to select the *Try or Install Ubuntu* menu option, then press Enter to boot into an Ubuntu Live session. After Ubuntu has booted completely, the screen shown in Figure 3-5 below will appear:

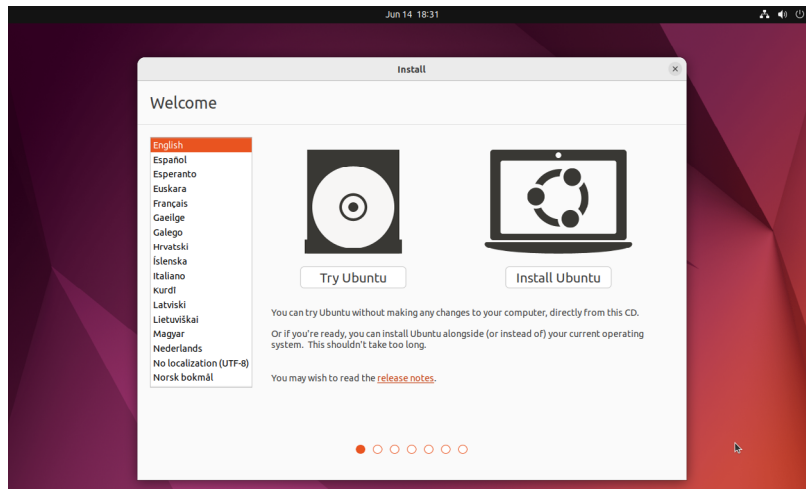


Figure 3-5

Options are provided to either try the Ubuntu Live session or to begin the installation process. If you experience a black screen when attempting to start or install Ubuntu, reboot the system and try again using the *safe graphics* options. If you are not ready to install Ubuntu, click on the Try Ubuntu button to safely explore the operating system without making changes to any installed hard drives.

3.6 Installing Ubuntu

From within Install dialog, select the option to begin the Ubuntu installation and wait for the initial screen of the installer to appear. Select your preferred language before clicking on the Continue button to proceed to the next screen:

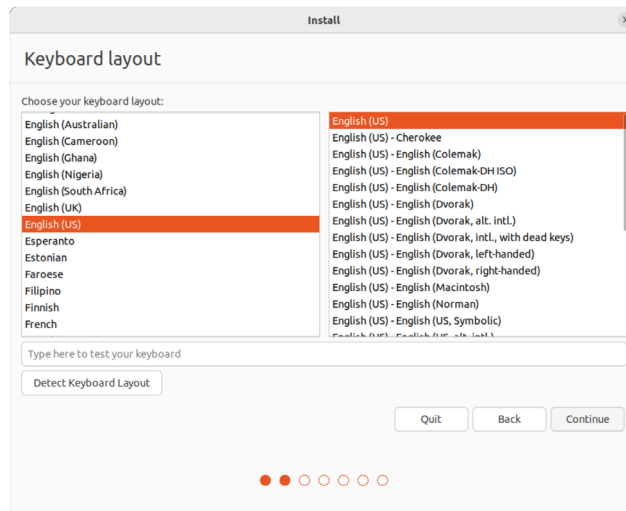


Figure 3-6

Either select your keyboard layout or, if you are unsure, click on the Detect Keyboard Layout

8. Using the Bash Shell on Ubuntu

22.04

An essential part of learning to work with Ubuntu and Linux distributions generally involves gaining proficiency in working in the shell environment. While graphical desktop environments such as GNOME, included with Linux, provide a user-friendly interface to the operating system, the shell environment provides far greater capabilities, flexibility, and automation than can ever be achieved using graphical desktop tools. The shell environment also provides a means for interacting with the operating system when a desktop environment is unavailable, a common occurrence when working with a server-based operating system such as Ubuntu or a damaged system that will not fully boot.

Therefore, this chapter aims to provide an overview of the default shell environment on Ubuntu (specifically the Bash shell).

8.1 What is a Shell?

The shell is an interactive command interpreter environment within which commands may be typed at a prompt or entered into a file as a script and executed. The origins of the shell can be traced back to the early days of the UNIX operating system. In fact, in the early days of Linux, before the introduction of graphical desktops, the shell was the only way for a user to interact with the operating system.

A variety of shell environments have been developed over the years. The first widely used shell was the Bourne shell, written by Stephen Bourne at Bell Labs.

Yet another early creation was the C shell which shared some syntax similarities with the C Programming Language and introduced usability enhancements such as command-line editing and history.

The Korn shell (developed by David Korn at Bell Labs) is based on features provided by both the Bourne and C shells.

The default shell on Ubuntu is the Bash shell (shorthand for Bourne Again SHell). This shell, which began life as an open-source version of the Bourne shell, was developed for the GNU Project by Brian Fox and is based on features provided by both the Bourne shell and the C shell.

8.2 Gaining Access to the Shell

From within the GNOME desktop environment, the shell prompt may be accessed from a Terminal window by selecting the Activities option in the top bar, entering Terminal into the search bar, and clicking the Terminal icon.

Using the Bash Shell on Ubuntu 22.04

When remotely logging into an Ubuntu server, for example, using SSH, the user is presented with a shell prompt. The chapter entitled “*Configuring SSH Key-based Authentication on Ubuntu 22.04*” will cover details on accessing a remote server using SSH. When booting a server-based system in which a desktop environment has not been installed, the shell is entered immediately after the user completes the login procedure at the physical console terminal or remote login session.

8.3 Entering Commands at the Prompt

Commands are entered at the shell command prompt simply by typing the command and pressing the Enter key. While some commands perform tasks silently, most will display some form of output before returning to the prompt. For example, the *ls* command can be used to display the files and directories in the current working directory:

```
$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

The available commands are either built into the shell itself or reside on the physical file system. The location on the file system of a command may be identified using the *which* command. For example, to find out where the *ls* executable resides on the file system:

```
$ which ls
alias ls='ls --color=auto'
      /usr/bin/ls
```

Clearly, the *ls* command resides in the */usr/bin* directory. Note also that an alias is configured, a topic that will be covered later in this chapter. Using the *which* command to locate the path to commands built into the shell will result in a message indicating the executable cannot be found. For example, attempting to find the location of the *history* command (which is built into the shell rather than existing as an executable on the file system) will result in output similar to the following:

```
$ which history
/usr/bin/which: no history in (/home/demo/.local/bin:/home/demo/bin:/usr/share/
Modules/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin)
```

8.4 Getting Information about a Command

Many of the commands available to the Linux shell can seem cryptic. To find out detailed information about what a command does and how to use it, use the *man* command specifying the name of the command as an argument. For example, to learn more about the *pwd* command:

```
$ man pwd
```

A detailed description of the *pwd* command will be displayed when the above command is executed. Many commands will also provide additional information when run with the *--help* command-line option:

```
$ wc --help
```

8.5 Bash Command-line Editing

Early shell environments did not provide any form of line editing capabilities. This meant that if you spotted an error at the beginning of a long command-line, you were typing, you had to delete

all the following characters, correct the error and then re-enter the remainder of the command. Fortunately, Bash provides a wide range of command-line editing options, as outlined in the following table:

Key Sequence	Action
Ctrl-b or Left Arrow	Move the cursor back one position
Ctrl-f or Right Arrow	Move the cursor forward one position
Delete	Delete the character currently beneath the cursor
Backspace	Delete the character to the left of the cursor
Ctrl-_	Undo previous change (can be repeated to undo all previous changes)
Ctrl-a	Move the cursor to the start of the line
Ctrl-e	Move the cursor to the end of the line
Meta-f or Esc then f	Move cursor forward one word
Meta-b or Esc then b	Move the cursor back one word
Ctrl-l	Clear the screen of everything except the current command
Ctrl-k	Delete to the end of the line from the current cursor position
Meta-d or Esc then d	Delete to end of the current word
Meta-DEL or Esc then DEL	Delete beginning to the current word
Ctrl-w	Delete from the current cursor position to the previous white space

Table 8-1

8.6 Working with the Shell History

In addition to command-line editing features, the Bash shell provides command-line history support. A list of previously executed commands may be viewed using the *history* command:

```
$ history
 1  ps
 2  ls
 3  ls -l /
 4  ls
 5  man pwd
 6  man apropos
```

In addition, Ctrl-p (or up arrow) and Ctrl-n (or down arrow) may be used to scroll back and forth through previously entered commands. Finally, when the desired command from the history is displayed, press the Enter key to execute it.

Another option is to enter the '!' character, followed by the first few characters of the command to be repeated, followed by the Enter key.

10. Managing Ubuntu 22.04 systemd Units

To gain proficiency in Ubuntu system administration, it is essential to understand the concepts of systemd units with a particular emphasis on two specific types known as targets and services. This chapter provides a basic overview of the different systemd units supported by Ubuntu and how to configure the many services that run in the background of a running Linux system.

10.1 Understanding Ubuntu systemd Targets

Ubuntu can be configured to boot into one of several states (referred to as targets), each designed to provide a specific level of operating system functionality. The system administrator configures the target to which a system will boot by default based on the purpose for which the system is being used. A desktop system, for example, will likely be configured to boot using the graphical user interface target. In contrast, a cloud-based server system would be more likely to boot to the multi-user target level.

During the boot sequence, a process named *systemd* looks in the */etc/systemd/system* folder to find the default target setting. Having identified the default target, it proceeds to start the systemd units associated with that target so that the system boots with all the necessary processes running.

For those familiar with older Ubuntu versions, systemd targets replace the older runlevel system.

10.2 Understanding Ubuntu systemd Services

A service is a process, typically running in the background, that provides specific functionality. The *sshd* service, for example, is the background process (also referred to as a *daemon*) that provides secure shell access to the system. Different systemd targets are configured to automatically launch different collections of services, depending on the functionality to be provided by that target.

Targets and services are types of systemd unit, a topic that will be covered later in this chapter.

10.3 Ubuntu systemd Target Descriptions

As previously outlined, Ubuntu can be booted into one of several target levels. The default target to which the system is configured to boot will, in turn, dictate which systemd units are started. The targets that relate specifically to system startup and shutdown can be summarized as follows:

- **poweroff.target** - This target shuts down the system. It is unlikely you would want this as your default target.
- **rescue.target** - Causes the system to start in a single-user mode under which only the root user can log in. The system does not start any networking, graphical user interface, or multi-user services in this mode. This run level is ideal for system administrators to perform system

maintenance or repair activities.

- **multi-user.target** - Boots the system into a multi-user mode with text-based console login capability.
- **graphical.target** - Boots the system into a networked, multi-user state with X Window System capability. By default, the graphical desktop environment will start at the end of the boot process. This is the most common run level for desktop or workstation use.
- **reboot.target** - Reboots the system. Another target that, for obvious reasons, you are unlikely to want as your default.

In addition to the above targets, the system includes about 70 other targets, many of which are sub-targets used by the above main targets. Behind the scenes, for example, *multi-user.target* will also start a target named *basic.target* which will, in turn, start the *sockets.target* unit, which is required for communication between different processes. This ensures that all the services on which the multi-user target depends are also started during the boot process.

A list of the targets and services on which a specified target is dependent can be viewed by running the following command in a terminal window:

```
# systemctl list-dependencies <target>
```

Figure 11-1, for example, shows a partial listing of the systemd unit dependencies for the graphical target (the complete listing contains over 140 targets and services required for a fully functional multi-user system):

```
[demo@demosever:~]$ systemctl list-dependencies graphical.target
graphical.target
● accounts-daemon.service
● apport.service
● gdm.service
● power-profiles-daemon.service
● switcheroo-control.service
○ systemd-update-utmp-runlevel.service
● udisks2.service
● multi-user.target
○ anacron.service
● apport.service
● avahi-daemon.service
● console-setup.service
● cron.service
● cups-browsed.service
```

Figure 10-1

The listing is presented as a hierarchical tree illustrating how some dependencies have sub-dependencies of their own. Scrolling to the bottom of the list, for example, would reveal that the graphical target depends on a cloud-related service called *cloud-init.target*, which, in turn, has its own service dependencies:

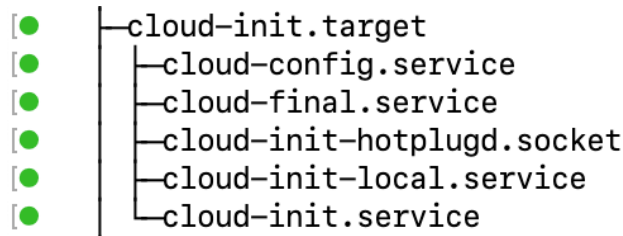


Figure 10-2

The colored dots to the left of each entry in the list indicate the current status of that service or target as follows:

- **Green** - The service or target is active and running.
- **White** - The service or target is inactive (dead). Typically because the service or target has yet to be enabled, has been stopped for some reason, or a condition on which the service or target depends has not been met.
- **Red** - The service or target failed to start due to a fatal error.

To find out more details about the status of a systemd unit, use the *systemctl status* command followed by the unit name as follows:

```
# systemctl status systemd-machine-id-commit.service
○ systemd-machine-id-commit.service - Commit a transient machine-id on disk
   Loaded: loaded (/usr/lib/systemd/system/systemd-machine-id-commit.service;
   static)
   Active: inactive (dead)
   Condition: start condition failed at Thu 2023-03-30 08:41:05 EDT; 16min ago
             └─ ConditionPathIsMountPoint=/etc/machine-id was not met
   Docs: man:systemd-machine-id-commit.service(8)
```

10.4 Identifying and Configuring the Default Target

The current default target for an Ubuntu system can be identified using the *systemctl* command as follows:

```
# systemctl get-default
multi-user.target
```

In the above case, the system is configured to boot using the multi-user target by default. The default setting can be changed anytime using the *systemctl* command with the *set-default* option. The following example changes the default target to start the graphical user interface the next time the system boots:

```
# systemctl set-default graphical.target
Removed /etc/systemd/system/default.target.
Created symlink /etc/systemd/system/default.target → /usr/lib/systemd/system/
graphical.target.
```

The output from the default change operation reveals the steps performed in the background by the *systemctl* command to implement the change. The current default is configured by establishing a symbolic link from the *default.target* file located in */etc/systemd/system* to point to the corresponding target file located in the */usr/lib/systemd/system* folder (in this case the *graphical.target* file).

10.5 Understanding systemd Units and Unit Types

As previously mentioned, targets and services are both types of systemd unit. All the files within the */usr/lib/systemd/system* folder are called systemd unit configuration files, each representing a systemd unit. Each unit is, in turn, categorized as being of a particular unit type. Ubuntu supports 12 different unit types, including the target and service unit types already covered in this chapter.

The type of a unit file is represented by the filename extension as outlined in Table 10-1 below:

Unit Type	Filename Extension	Type Description
Service	.service	System service.
Target	.target	Group of systemd units.
Automount	.automount	File system auto-mount point.
Device	.device	Device file recognized by the kernel.
Mount	.mount	File system mount point.
Path	.path	File or directory in a file system.
Scope	.scope	Externally created process.
Slice	.slice	Group of hierarchically organized units that manage system processes.
Snapshot	.snapshot	Saved state of the systemd manager.
Socket	.socket	Inter-process communication socket.
Swap	.swap	Swap device or a swap file.
Timer	.timer	Systemd timer.

Table 10-1

Note that the target unit type differs from other types in that it comprises a group of systemd units such as services or other targets.

10.6 Dynamically Changing the Current Target

The *systemctl set-default* command outlined previously specifies the target that will be used the next time the system starts but does not change the current system’s state. To change to a different

11. Ubuntu Software Package Management and Updates

It is highly unlikely that a newly installed Ubuntu system will contain all of the software packages necessary to perform the tasks for which it is intended. Even once all the required software has been installed, it is almost certain that newer versions of many of those packages will be released during the lifespan of the system. In some cases, you will need to ensure that these latest package releases are installed on the system so that bugs and security vulnerabilities are fixed.

This chapter introduces the basic concepts of software management on Ubuntu, explains how these issues are addressed, and explores the concepts of repositories and software packages while exploring how to list, install and remove the software packages that make up a functioning Ubuntu system.

11.1 Repositories

Linux is essentially comprised of a set of base packages that provide the core functionality of the operating system together with a range of other packages and modules that add functionality and features on top of the base operating system.

When Ubuntu is first installed, a number of different packages will be installed depending on the software options selected during the installation phase. Once the system is up and running, however, additional software can be installed as needed. Typically, all software that is part of Ubuntu (in other words, software that is not provided by a third-party vendor) is downloaded and installed on the system using the Advanced Package Tool (*apt*) command. As we have seen in earlier chapters, this typically consists of a command similar to the following being issued at the command prompt:

```
# apt install apache2
```

When such a command is issued, the requested software is downloaded from a remote *repository* and installed on the local system. By default, Ubuntu is configured to download software from a number of different repositories:

- **main** - Contains the core set of packages that are officially supported, tested and updated by Ubuntu.
- **restricted** - Proprietary drivers for hardware devices for which no open source equivalent exists.
- **universe** - Contains packages that are not officially supported by the Ubuntu team at Canonical. These packages are, however, maintained by the Ubuntu community and include packages not available within the main repository.

Ubuntu Software Package Management and Updates

- **multiverse** - Packages that may not conform to the open-source licensing terms under which Ubuntu is released due to copyright or other legal issues.

The list of currently enabled repositories on an Ubuntu system is contained within the `/etc/apt/sources.list` file which can be loaded into an editor to be viewed and modified. The file may be manually loaded into an editor, or edited using a choice of available editors using the following command:

```
# apt edit-sources
```

The first few lines of this file usually reference the main and restricted repositories, for example:

```
deb http://ports.ubuntu.com/ubuntu-ports/ jammy main restricted
```

In the above example the list is configured to allow packages to be downloaded from the main and restricted repositories. Entries for the universe and multiverse repositories will also be included in the file:

```
## N.B. software from this repository may not have been tested as
## extensively as that contained in the main release, although it includes
## newer versions of some applications which may provide useful features.
## Also, please note that software in backports WILL NOT receive any review
## or updates from the Ubuntu security team.
deb http://us.archive.ubuntu.com/ubuntu/ jammy-backports main restricted universe
multiverse
```

To disable a repository so that it will no longer be used to download packages, simply comment out the line by prefixing it with a '#' character:

```
#deb http://ports.ubuntu.com/ubuntu-ports/ jammy-backports main restricted
universe multiverse
```

In addition to the standard repositories there are also many third-party repositories. In the event that you need to use one of these, simply add an entry for it to the `sources.list` file. One such example is the partners repository, which can be added to the `sources.list` file as follows:

```
deb http://archive.canonical.com/ubuntu jammy partner
```

After making the change, run the following command to commit the changes:

```
# apt update
```

11.2 Managing Repositories with Software & Updates

As an alternative to using the command-line, repositories may be configured from within the GNOME desktop environment using the *Software & Updates* app. To launch this app, press the *special key* on the keyboard (on Windows keyboards this is the Windows key, on macOS the Command key and on Chromebooks the key displaying a magnifying glass) and enter Software & Updates into the search bar. In the results panel click on the corresponding icon to launch the app. Alternatively, open a terminal window and run the following command:

```
$ update-manager
```

When the app loads, click on the Settings button as shown in Figure 11-1:

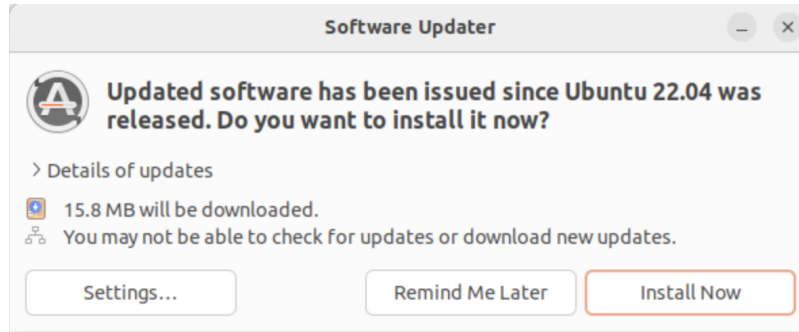


Figure 11-1

From the settings screen, enable or disable the required repositories listed under the *Downloadable from the Internet* heading:

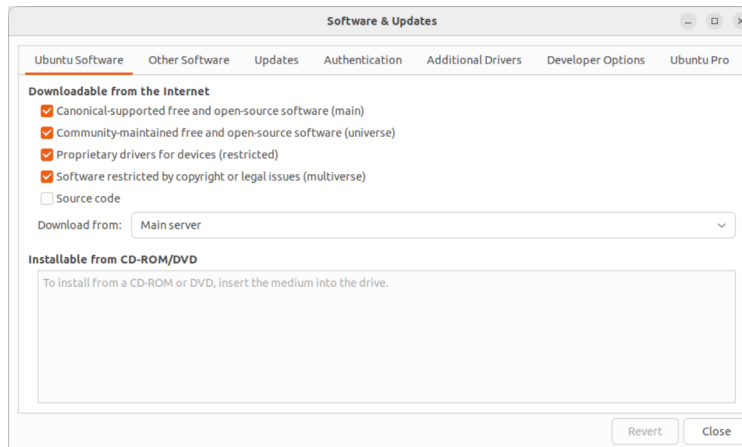


Figure 11-2

To enable partner repositories, select the Other Software tab as shown in Figure 11-3:

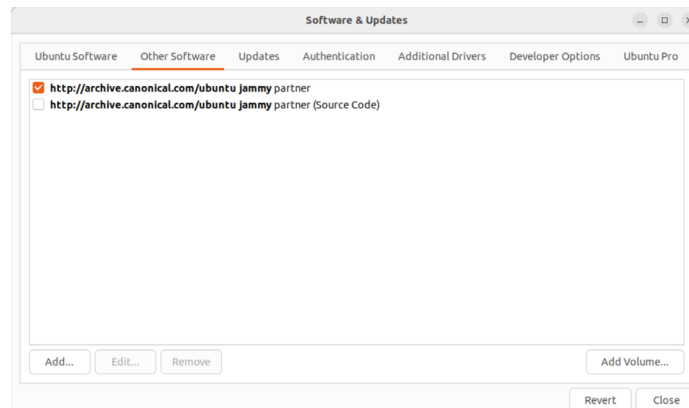


Figure 11-3

14. Ubuntu 22.04 Firewall Basics

A firewall is a vital component in protecting an individual computer system or network of computers from external attacks (typically from an internet connection). Any computer connected directly to an internet connection should ideally run a firewall to protect against malicious activity. Similarly, any internal network must have some form of firewall between it and an external internet connection.

Ubuntu is supplied with powerful firewall technology known as *iptables* built-in. Entire books can, and indeed have, been written about configuring *iptables*. If you would like to learn about *iptables*, we recommend the following:

https://www.linuxtopia.org/Linux_Firewall_iptables/index.html

This chapter will cover some basic concepts of firewalls, TCP/IP ports, and services. Firewall configuration on Ubuntu will be covered in the chapters entitled “*Ubuntu 22.04 Firewall Configuration with firewalld*” and “*Using gufw and ufw to Configure an Ubuntu Firewall*”.

14.1 Understanding Ports and Services

The predominant network communications protocol in use these days is TCP/IP. It is the protocol used by the internet and, as such, has swept away most of the formerly popular protocols used for local area networks (LANs).

TCP/IP defines a total of 65,535 ports, of which 1023 are considered *well-known ports*. It is essential to understand that these are not physical ports into which network cables are connected but rather virtual ports on each network connection which can be used by applications and services to communicate over a TCP/IP network connection. In reality, the number of ports used by popular network clients and services comprises an even smaller subset of the well-known group of ports.

An operating system can provide several different TCP/IP services. A comprehensive list of such services is provided in the table at the end of this chapter. Still, such services include HTTPS for running a secure web server, FTP for allowing file transfers, SSH for providing secure remote login access and file transfer, and SMTP for transporting email messages. Each service is, in turn, assigned to a standard TCP/IP port. For example, HTTPS is assigned to port 443, while SSH communication occurs on port 22.

14.2 Securing Ports and Services

A large part of securing servers involves defining roles and, based on the roles, defining which services and ports should be enabled. For example, a server that acts solely as a web server should only run the HTTPS service (in addition to perhaps SSH for remote administration access). All other services should be disabled and, ideally, removed entirely from the operating system

Ubuntu 22.04 Firewall Basics

(thereby making it harder for an intruder to re-enable the service).

Securing a system involves removing any unnecessary services from the operating system and ensuring that the ports associated with the non-essential services are blocked using a firewall. The rules that define which ports are accessible and under what circumstances are determined using iptables.

Many operating systems are installed with several services installed and activated by default. Before installing a new operating system, the installation must be carefully planned. This planning involves deciding which services are not required and identifying which services have been installed and enabled by default. Deployment of new operating system installations should never be rushed. The fewer services and open ports available on a system, the smaller the surface area and opportunities for attackers.

14.3 Ubuntu Services and iptables Rules

By default, a newly installed Ubuntu system has no iptables rules defined to restrict access to ports. The following command may be executed in a terminal window to view the current iptables settings:

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                               destination
```

As illustrated in the above output, no rules are currently defined. While this may appear to be an unsafe configuration, it is essential to remember that a newly installed Ubuntu system also has few services running by default, making the ports useless to a potential attacker. For example, accessing a web server on a newly installed Ubuntu system is impossible because no web server services are installed or running by default. Once services begin to be activated on the system, it will be important to establish a firewall strategy by defining iptables rules.

Several methods are available for defining iptables rules, including using command line tools and configuration files. For example, to block access to port 25 (used by the SMTP mail transfer protocol) from IP address 192.168.2.76, the following command could be issued in a terminal window:

```
# iptables -A INPUT -s 192.168.2.76 -p tcp --destination-port 25 -j DROP
```

If we now check the current rules, we will see that this one is currently listed:

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                               destination
```

```
DROP          tcp  --  192.168.2.76          anywhere          tcp dpt:smtp
```

```
Chain FORWARD (policy ACCEPT)
target        prot opt source                destination
```

```
Chain OUTPUT (policy ACCEPT)
target        prot opt source                destination
```

The rule may subsequently be removed as follows:

```
# iptables -D INPUT -s 192.168.2.76 -p tcp --destination-port 25 -j DROP
```

Given the complexity of iptables it is unsurprising that several user-friendly configuration tools have been created to ease the rule creation process. One such tool is the *firewall-cmd* command-line tool which will be covered in the chapter “*Ubuntu 22.04 Firewall Configuration with firewallld*”.

14.4 Well-Known Ports and Services

Before moving on to cover more complex firewall rules, it is first worth taking time to outline some of the key services that can be provided by a Ubuntu system, together with the corresponding port numbers:

Port	Assignment	Description
20	FTP	File Transfer Protocol (Data) - The File Transfer protocol provides a mechanism for transferring specific files between network-connected computer systems. The transfer is typically performed using the ftp client. Most modern web browsers can also browse and download files on a remote FTP server. FTP uses TCP (rather than UDP) to transfer files, which is considered a highly reliable transport mechanism. FTP does not encrypt data and is not considered a secure file transfer protocol. Secure Copy Protocol (SCP) and Secure File Transfer Protocol (SFTP) are strongly recommended in place of FTP.
21	FTP	File Transfer (Control) - Traditionally, FTP has two ports assigned (port 20 and port 21). Port 20 was initially considered the data transfer port, while port 21 was assigned to communicate control information. However, in modern implementations, port 20 is rarely used, with all communication taking place on port 21.

22	SSH	Secure Shell - The Secure Shell provides a safe, encrypted, remote login session to a host over a TCP/IP network. The original mechanism for remote access was the Telnet protocol. However, because Telnet transmits data in plain text, its use is strongly discouraged in favor of the secure shell, which encrypts all communications, including login and password credentials. SSH also provides the mechanism by which files can be securely transferred using the Secure Copy Protocol (SCP) and is also the basis for the Secure File Transfer Protocol (SFTP). SSH also replaces both the rsh and rlogin clients.
23	Telnet	Telnet - Telnet is a terminal emulation protocol that can log into a remote system over a TCP/IP connection. The access is text-based, allowing the user to type into a command prompt on the remote host, and text displayed by the remote host is displayed on the local Telnet client. Telnet encrypts neither the password nor the text communicated between the client and server. As such, the use of telnet is strongly discouraged. Most modern systems will have port 23 closed and the telnet service disabled to prevent its use. SSH should be used in place of Telnet.
25	SMTP	Simple Mail Transfer Protocol - SMTP defines the mechanism by which email messages are sent from one network host to another. SMTP is a straightforward protocol requiring the mail service to always be available at the receiving host. Typically the receiving host will store incoming messages in a spool for subsequent access by the recipient using the POP3 or IMAP protocols. In addition, SMTP uses the TCP transport protocol to ensure error-free message delivery.
53	DNS	Domain Name Server - The service used by TCP/IP networks to translate host names and Fully Qualified Domain Names (FQDN) to IP addresses.
69	TFTP	Trivial File Transfer Protocol - TFTP is a stripped-down version of the File Transfer Protocol (FTP). It has a reduced command set and lacks authentication. The most significant feature of TFTP is that it uses UDP to transfer data. This results in high-speed transfer speeds but, consequently, lacks data reliability. TFTP is typically used in network-based booting for diskless workstations.

80	HTTP	Hypertext Text Transfer Protocol - HTTP is used to download text, graphics, and multimedia from a web server to a web browser. It defines the command and control mechanism between the browser and server, defining client requests and server responses. HTTP is based on the TCP transport protocol and, as such, is a connection-oriented protocol.
110	POP3	Post Office Protocol - The POP3 protocol is a mechanism for storing and retrieving incoming email messages from a server. In most corporate environments, incoming email is stored on an email server and then downloaded to an email client running on the user's desktop or laptop when the user checks email. However, POP3 downloads all new messages to the client and does not allow the user to choose which messages to download, view headers, or download only parts of messages. For this reason, the IMAP protocol is increasingly being used in place of POP3.
119	NNTP	Network News Transfer Protocol - The protocol responsible for posting and retrieving messages to and from Usenet News Servers (i.e., newsgroups and discussion forums hosted on remote servers). NNTP operates at the Application layer of the OSI stack and uses TCP to ensure error-free message retrieval and transmission.
123	NTP	Network Time Protocol - A protocol designed to synchronize computer clocks with an external time source. Using this protocol, an operating system or application can request the current time from a remote NTP server. The remote NTP server is usually based on the time provided by a nuclear clock. NTP is useful for ensuring that all systems in a network are set to the same, accurate time of day. This is of particular importance in security situations when, for example, the time a file was accessed or modified on a client or server is in question.
143	IMAP4	Internet Message Access Protocol, Version 4 - IMAP4 is an advanced and secure email retrieval protocol. IMAP is similar to POP3, allowing users to access email messages stored on an email server. However, IMAP includes many additional features, such as the ability to selectively download messages, view message headers, search messages, and download part of a message. In addition, IMAP4 uses authentication and fully supports Kerberos authentication.

161	SNMP	Simple Network Management Protocol - Provides a mechanism whereby network administrators can collect information about network devices (such as hubs, bridges, routers, and switches). The SNMP protocol enables agents running on network devices to communicate their status to a central manager and, in turn, allows the manager to send new configuration parameters to the device agent. The agents can further be configured to notify the manager when certain events, known as traps, occur. SNMP uses UDP to send and receive data.
443	HTTPS	Hypertext Transfer Protocol Secure - The standard HTTP (non-secure) protocol transfers data in clear text (i.e., with no encryption and visible to anyone who might intercept the traffic). While this is acceptable for most web browsing purposes, it poses a severe security risk when confidential information such as credit card details needs to be transmitted from the browser to the web server. HTTPS addresses this using the Secure Sockets Layer (SSL) to send encrypted data between the client and server.
2049	NFS	Network File System - Originally developed by Sun Microsystems and subsequently widely adopted throughout the industry, NFS allows a file system on a remote system to be accessed over the network by another system as if the file system were on a local disk drive. NFS is widely used on UNIX and LINUX-based systems. Later versions of Microsoft Windows can also access NFS-shared file systems on UNIX and LINUX-based systems.

Table 14-1

14.5 Summary

A newly installed Ubuntu system is generally considered secure due to the absence of services running on the system ports. Once the system begins to be configured for use, however, it is important to ensure that it is protected by implementing a firewall. When configuring firewalls, it is important to understand the various ports and the corresponding services.

Several firewall options are available, the most basic being the command-line configuration of the iptables firewall interface. More intuitive and advanced options are available via firewalld, which will be covered in the next chapter.

15. Using gufw and ufw to Configure an Ubuntu Firewall

In the previous chapter, we looked at ports and services on an Ubuntu system. We also briefly looked at iptables firewall rules on Ubuntu including the creation of a few very simple rules from the command line. In this chapter, we will look at a more user-friendly approach to iptables configuration using two tools named gufw and ufw. As we will see, gufw and ufw provide a high level of control over both inbound and outbound network traffic and connections without the need to understand the lower-level iptables syntax.

15.1 An Overview of gufw and ufw

Included with Ubuntu is a package called ufw which is an acronym for Uncomplicated Firewall. This package provides a command line interface for managing and configuring rules for the Netfilter iptables-based firewall. The gufw tool provides a user-friendly graphical interface to ufw designed to make firewall management possible without the need to issue ufw commands at the command line.

15.2 Installing gufw on Ubuntu

Whilst ufw is installed on Ubuntu by default, the gufw package is not. To install gufw, therefore, open a Terminal window (Ctrl-Alt-T) and enter the following command at the resulting prompt:

```
# apt install gufw
```

15.3 Running and Enabling gufw

Once installed, launch *gufw* by pressing Alt-F2 within the GNOME desktop and entering *gufw* into the Run a command text box. When invoked for the first time, it is likely that the firewall will be disabled, as illustrated in Figure 15-1.

To enable the firewall, move the Status switch (A) to the on position. By default, the main panel (D) will be displaying the gufw home page containing some basic information about the tool. Selecting options from the row of buttons (C) will change the information displayed in the panel. For example, select the Rules button to add, remove and view rules.

The gufw tool is provided with a small set of pre-configured profiles for work, home, and public environments. To change the profile and view the settings, select the profile from the menu (B). To modify an existing profile, select it from the menu and use the Incoming and Outgoing menus to change the selections. To configure specific rules, display the Rules screen and add, remove, and modify rules as required. These will then be applied to the currently selected profile.

Using gufw and ufw to Configure an Ubuntu Firewall

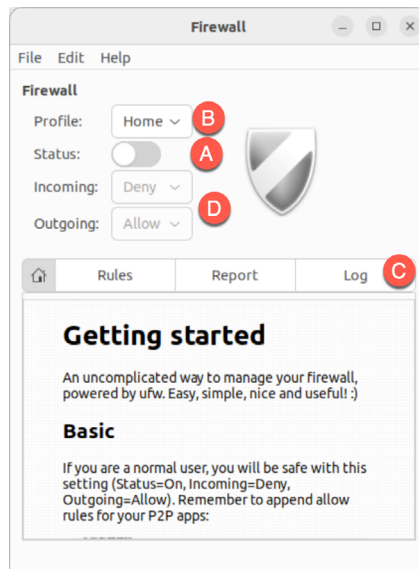


Figure 15-1

The currently selected profile dictates how the firewall handles traffic in the absence of any specific policy rules. By default, the Home profile, for example, is configured to deny all incoming traffic and allow all outgoing traffic. These default policy settings are changed using the Incoming: and Outgoing: menus (E).

Exceptions to the default policy are defined through the creation of additional rules. With the Home profile denying incoming traffic, for example, rules would need to be added to enable certain acceptable types of incoming connections. Such rules are referred to in the security community as a whitelist.

If, on the other hand, the incoming policy was changed to Allow all traffic, then all incoming traffic would be permitted unless rules were created for specific types of connections that must be blocked. These rules, unsurprisingly, are referred to as a blacklist. The blacklist/whitelist approach applies equally to incoming and outgoing connections.

15.4 Creating a New Profile

While it is possible to modify the pre-defined profiles, it will typically make more sense to create one or more profiles to configure the firewall for your specific needs. New profiles are created by selecting the *Edit -> Preferences...* menu option to display the dialog shown in Figure 15-2:

Using gufw and ufw to Configure an Ubuntu Firewall

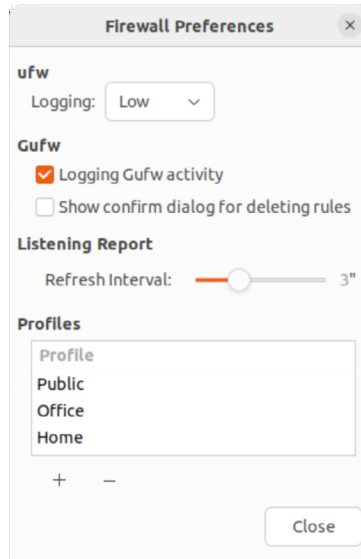


Figure 15-2

To add a new profile, click on the '+' button located beneath the list of profiles. A new profile named Profile 1 will appear in the list. To give the profile a more descriptive name, double-click on the entry to enter edit mode and enter a new name:

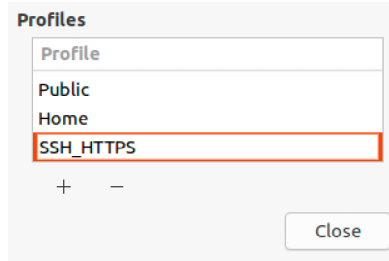


Figure 15-3

Once the profile has been created and named, click on the Close button to return to the main screen, select it from the Profile menu and turn on the Status switch:

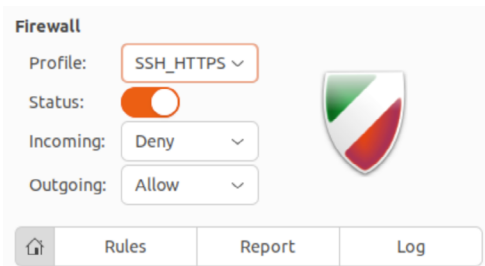


Figure 15-4

16. Basic Ubuntu Firewall Configuration with firewalld

All Linux distributions are provided with a firewall solution of some form. In the case of Ubuntu this takes the form of the Uncomplicated Firewall outlined in the previous chapter. This chapter will introduce a more advanced firewall solution available for Ubuntu in the form of firewalld.

16.1 An Introduction to firewalld

Originally developed for Red Hat-based Linux distributions, the firewalld service uses a set of rules to control incoming network traffic and define which traffic is to be blocked and which is to be allowed to pass through to the system and is built on top of a more complex firewall tool named *iptables*.

The firewalld system provides a flexible way to manage incoming traffic. The firewall could, for example, be configured to block traffic arriving from a specific external IP address or to prevent all traffic arriving on a particular TCP/IP port. Rules may also be defined to forward incoming traffic to different systems or to act as an internet gateway to protect other computers on a network.

In keeping with common security practices, a default firewalld installation is configured to block all access with the exception of SSH remote login and the DHCP service used by the system to obtain a dynamic IP address (both of which are essential if the system administrator is to be able to gain access to the system after completing the installation).

The key elements of firewall configuration on Ubuntu are *zones*, *interfaces*, *services*, and *ports*.

16.1.1 Zones

By default, firewalld is installed with a range of pre-configured *zones*. A zone is a preconfigured set of rules which can be applied to the system at any time to quickly implement firewall configurations for specific scenarios. The *block* zone, for example, blocks all incoming traffic, while the *home* zone imposes less strict rules on the assumption that the system is running in a safer environment where a greater level of trust is expected. New zones may be added to the system, and existing zones modified to add or remove rules. Zones may also be deleted entirely from the system. Table 16-1 lists the set of zones available by default on an Ubuntu system:

Zone	Description
drop	The most secure zone. Only outgoing connections are permitted and all incoming connections are dropped without any notification to the connecting client.

Zone	Description
block	Similar to the drop zone with the exception that incoming connections are rejected with an icmp-host-prohibited or icmp6-adm-prohibited notification.
public	Intended for use when connected to public networks or the internet where other computers are not known to be trustworthy. Allows select incoming connections.
external	When a system is acting as the internet gateway for a network of computers, the external zone is applied to the interface that is connected to the internet. This zone is used in conjunction with the <i>internal</i> zone when implementing masquerading or network address translation (NAT) as outlined later in this chapter. Allows select incoming connections
internal	Used with the <i>external</i> zone and applied to the interface that is connected to the internal network. Assumes that the computers on the internal network are trusted. Allows select incoming connections.
dmz	For use when the system is running in the demilitarized zone (DMZ). These are generally computers that are publicly accessible but isolated from other parts of your internal network. Allows select incoming connections.
work	For use when running a system on a network in a work environment where other computers are trusted. Allows select incoming connections.
home	For use when running a system on a home network where other computers are trusted. Allows select incoming connections.
trusted	The least secure zone. All incoming connections are accepted.

Table 16-1

To review specific settings for a zone, refer to the corresponding XML configuration file located on the system in the `/usr/lib/firewalld/zones` directory. The following, for example, lists the content of the *public.xml* zone configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Public</short>
  <description>For use in public areas. You do not trust the other computers
on networks to not harm your computer. Only selected incoming connections are
accepted.</description>
  <service name="ssh"/>
  <service name="mdns"/>
  <service name="dhcpv6-client"/>
</zone>
```

16.1.2 Interfaces

Any Ubuntu system connected to the internet or a network (or both) will contain at least one *interface* in the form of either a physical or virtual network device. When firewalld is active, each of these interfaces is assigned to a zone allowing different levels of firewall security to be assigned to different interfaces. Consider a server containing two interfaces, one connected externally to the internet and the other to an internal network. In such a scenario, the external facing interface would most likely be assigned to the more restrictive *external* zone, while the internal interface might use the *internal* zone.

16.1.3 Services

TCP/IP defines a set of services that communicate on standard ports. Secure HTTPS web connections, for example, use port 443, while the SMTP email service uses port 25. To selectively enable incoming traffic for specific services, firewalld rules can be added to zones. The *home* zone, for example, does not permit incoming HTTPS connections by default. This traffic can be enabled by adding rules to a zone to allow incoming HTTPS connections without having to reference the specific port number.

16.1.4 Ports

Although common TCP/IP services can be referenced when adding firewalld rules, situations will arise where incoming connections need to be allowed on a specific port that is not allocated to a service. This can be achieved by adding rules that reference specific ports instead of services.

16.2 Checking firewalld Status

The firewalld service is not usually installed and enabled by default on all Ubuntu installations. The status of the service can be checked via the following command:

```
# systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/lib/systemd/system/firewalld.service; enabled; vendor
  preset: enabled)
   Active: active (running) since Mon 2023-07-17 19:21:53 UTC; 16s ago
     Docs: man:firewalld(1)
  Main PID: 27151 (firewalld)
    Tasks: 2 (limit: 4517)
   Memory: 24.0M
      CPU: 244ms
   CGroup: /system.slice/firewalld.service
           └─27151 /usr/bin/python3 /usr/sbin/firewalld --nofork --nopid

Jul 17 19:21:53 demoserver systemd[1]: Starting firewalld - dynamic firewall
daemon...
Jul 17 19:21:53 demoserver systemd[1]: Started firewalld - dynamic firewall daemon.
```

If necessary, the firewalld service may be installed as follows:

```
# apt install firewalld
```

18. Ubuntu 22.04 Remote Desktop Access with Vino

Ubuntu can be configured to provide remote access to the graphical desktop environment over a network or internet connection. Although not enabled by default, it is relatively straightforward to display and access an Ubuntu desktop from a system anywhere else on a network or the internet. This can be achieved regardless of whether that system is running Linux, Windows, or macOS. In fact, there are even apps available for Android and iOS that will allow you to access your Ubuntu desktop from just about anywhere that a data signal is available.

Remote desktop access can be useful in a number of scenarios. It enables you or another person, for example, to view and interact with your Ubuntu desktop environment from another computer system either on the same network or over the internet. This is useful if you need to work on your computer when you are away from your desk, such as while traveling. It is also useful in situations where a co-worker or IT support technician needs access to your desktop to resolve a problem.

The Ubuntu remote desktop supports connections using either Virtual Network Computing (VNC) or Microsoft's Remote Desktop Protocol (RDP). An advantage of using RDP is that it makes the Ubuntu desktop easily accessible from Windows clients. In this chapter, therefore, we will cover the key aspects of configuring and using remote desktops within Ubuntu using RDP.

18.1 Remote Desktop Access Types

Before starting, it is important to understand that there are essentially two types of remote desktop access. The approach covered in this chapter is useful if you primarily use Ubuntu as a desktop operating system and require remote access to your usual desktop session. When configured, you will take over your desktop session and view and control it remotely.

The second option is intended for situations where you need to start and access one or more remote desktop sessions on a remote server-based system, regardless of whether the remote system has a graphical console attached. This allows you to launch multiple desktop sessions in the background on the remote system and view and control those desktops over a network or internet connection.

18.2 Secure and Insecure Remote Desktop Access

In this chapter, we will cover both secure and insecure remote desktop access methods. Assuming that you are accessing one system from another within the context of a secure internal network then it is generally safe to use the insecure access method. If, on the other hand, you plan to access your desktop remotely over any kind of public network, you must use the secure method of access to avoid your system and data being compromised.

18.3 Enabling Remote Desktop Access on Ubuntu

Remote desktop access on Ubuntu is provided by the Vino package. Vino is a remote desktop server that was developed specifically for use with the GNOME desktop.

The first step in enabling remote access is to install this package:

```
# apt install vino
```

Once Vino has been installed, the next step is to enable remote desktop access from within GNOME. Begin by opening the settings app as shown in Figure 18-1:

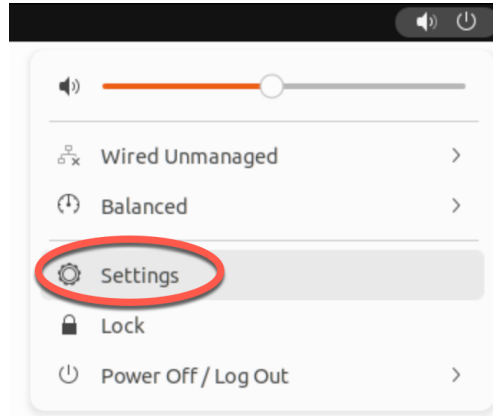


Figure 18-1

From within the Settings application, select the Sharing option (marked A in Figure 18-2):

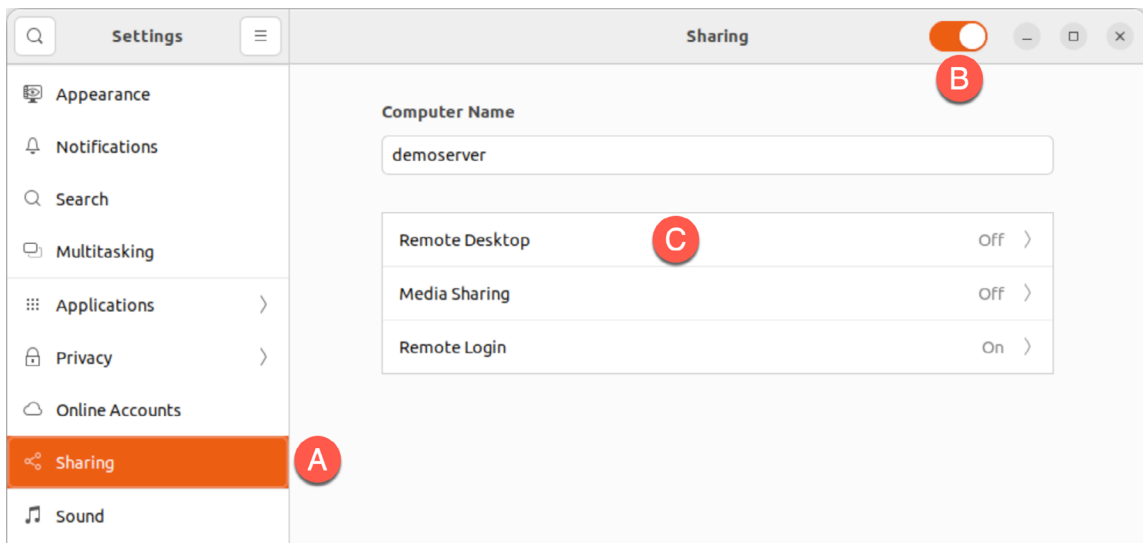


Figure 18-2

Turn on the Sharing switch (B) and click on the Remote Desktop option (C) to display the dialog

shown in Figure 18-3 below:

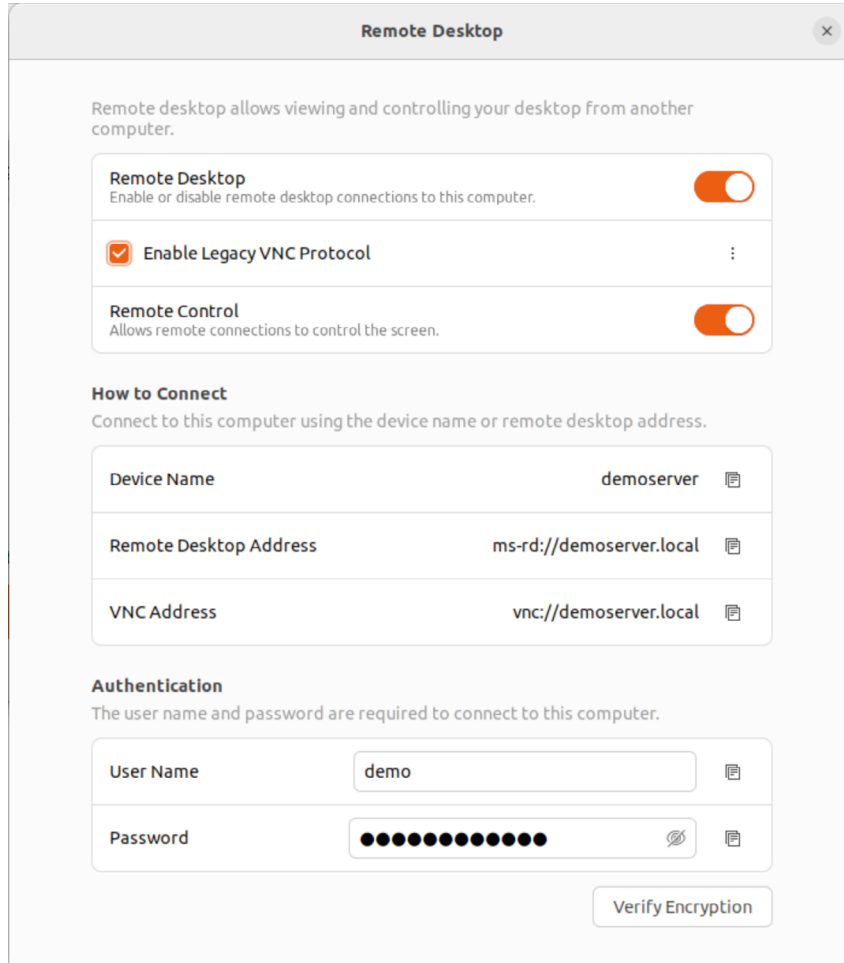


Figure 18-3

The Remote Desktop dialog provides the following configuration options to manage remote desktop access:

- **Remote Control** - If enabled, the remote session will be able to use the mouse and keyboard to interact with the desktop environment. If this option disabled, the remote session will only allow the desktop to be viewed.
- **Enable Legacy VNC Protocol** - Ubuntu supports remote desktop via VNC and Microsoft's Remote Desktop Protocol (RDP). When enabled, two additional options are available within the menu shown in Figure 18-4 below:

21. Sharing Files between Ubuntu 22.04 and Windows with Samba

Although Linux has made some inroads into the desktop market, its origins and future are very much server based. It is unsurprising, therefore, that Ubuntu can act as a file server. It is also common for Ubuntu and Windows systems to be used side by side in networked environments. Therefore, it is a common requirement that files on an Ubuntu system be accessible to Linux, UNIX, and Windows-based systems over network connections. Similarly, shared folders and printers residing on Windows systems may also need to be accessible from Ubuntu-based systems.

Windows systems share resources such as file systems and printers using the Server Message Block (SMB) protocol. For an Ubuntu system to serve such resources over a network to a Windows system and vice versa, it must support SMB. This is achieved using a technology called Samba. In addition to providing integration between Linux and Windows systems, Samba may also provide folder sharing between Linux systems (as an alternative to NFS covered in the previous chapter).

In this chapter, we will look at the steps necessary to share file system resources and printers on an Ubuntu system with remote Windows and Linux systems and to access Windows resources from Ubuntu.

21.1 Accessing Windows Resources from the GNOME Desktop

Before getting into more details of Samba sharing, it is worth noting that if all you want to do is access Windows shared folders from within the GNOME desktop, then support is already provided within the GNOME Files application. The Files application is located in the dash as highlighted in Figure 21-1:

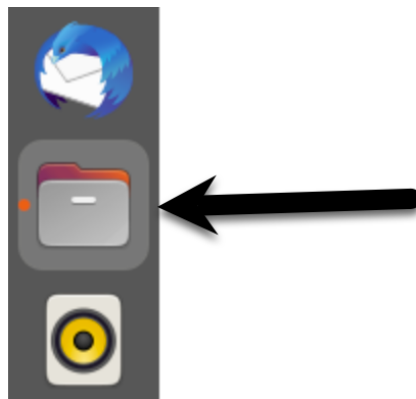


Figure 21-1

Once launched, select the *Other Locations* option in the left-hand navigation panel, followed by

Sharing Files between Ubuntu 22.04 and Windows with Samba

the Windows Network icon in the main panel to browse available Windows resources:

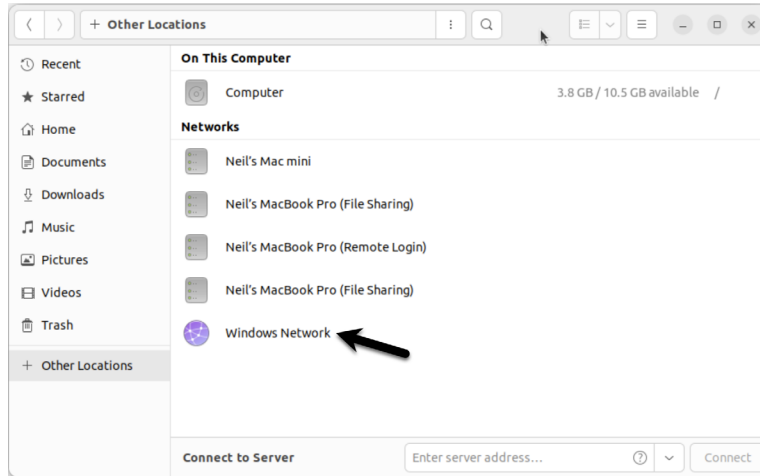


Figure 21-2

21.2 Samba and Samba Client

Samba allows both Ubuntu resources to be shared with Windows systems and Windows resources to be shared with Ubuntu systems. Ubuntu accesses Windows resources using the Samba client. On the other hand, Ubuntu resources are shared with Windows systems by installing and configuring the Samba service.

21.3 Installing Samba on Ubuntu

The default settings used during the Ubuntu installation do not typically install the necessary Samba packages. Unless you specifically requested that Samba be installed, it is unlikely that you have Samba installed on your system. To check whether Samba is installed, open a terminal window and run the following command and check for the [installed] indicator for each package:

```
# apt -qq list samba-common samba smbclient
```

Any missing packages can be installed using the *apt* command-line tool:

```
# apt install samba-common samba smbclient
```

21.4 Configuring the Ubuntu Firewall to Enable Samba

Next, the firewall protecting the Ubuntu system must be configured to allow Samba traffic. If you are using the Uncomplicated Firewall (ufw) run the following command:

```
# ufw allow samba
```

Alternatively, if you are using *firewalld*, run the *firewall-cmd* command as follows:

```
# firewall-cmd --permanent --add-port={139/tcp,445/tcp}
# firewall-cmd --reload
```

Before starting the Samba service, some configuration steps are necessary to define how the Ubuntu system will appear to Windows systems and the resources to be shared with remote

clients. Most configuration tasks occur within the `/etc/samba/smb.conf` file.

21.5 Configuring the `smb.conf` File

Samba is a highly flexible and configurable system that provides many options for controlling how resources are shared on Windows networks. Unfortunately, this flexibility can lead to the sense that Samba is overly complex. In reality, however, the typical installation does not need many configuration options, and the learning curve to set up a basic configuration is relatively short.

For this chapter, we will look at joining an Ubuntu system to a Windows workgroup and setting up a directory as a shared resource that a specific user can access. This is a configuration known as a *standalone Samba server*. More advanced configurations, such as integrating Samba within an Active Directory environment, are also available, though these are outside the scope of this book.

The first step in configuring Samba is to edit the `/etc/samba/smb.conf` file.

21.5.1 Configuring the [global] Section

The `smb.conf` file is divided into sections. The first section is the [global] section, where settings that apply to the entire Samba configuration can be specified. While these settings are global, each option may be overridden within other configuration file sections.

The first task is defining the Windows workgroup name on which the Ubuntu resources will be shared. This is controlled via the `workgroup =` directive of the [global] section, which by default is configured as follows:

```
workgroup = WORKGROUP
```

Begin by changing this to the actual name of the workgroup if necessary.

In addition to the workgroup setting, the other settings indicate that this is a standalone server on which user passwords will protect the shared resources. Before moving on to configuring the resources to be shared, other parameters also need to be added to the [global] section as follows:

```
[global]
.
.
    netbios name = LinuxServer
```

The “netbios name” property specifies the name by which the server will be visible to other systems on the network.

21.5.2 Configuring a Shared Resource

The next step is configuring the shared resources (in other words, the resources that will be accessible from other systems on the Windows network). To achieve this, the section is given a name by which it will be referred when shared. For example, if we plan to share the `/sampleshare` directory of our Ubuntu system, we might entitle the section [sampleshare]. In this section, a variety of configuration options are possible. For this example, however, we will define the directory that is to be shared, indicate that the directory is both browsable and writable, and declare the resource public so that guest users can gain access:

Sharing Files between Ubuntu 22.04 and Windows with Samba

```
[sampleshare]
    comment = Example Samba share
    path = /sampleshare
    browseable = Yes
    public = yes
    writable = yes
```

To restrict access to specific users, the “valid users” property may be used, for example:

```
valid users = demo, bobyong, marcewing
```

21.5.3 Removing Unnecessary Shares

The *smb.conf* file is pre-configured with sections for sharing printers and the home folders of the users on the system. If these resources do not need to be shared, the corresponding sections can be commented out so that Samba ignores them. In the following example, the [homes] section has been commented out:

```
.
.
#[homes]
#    comment = Home Directories
#    valid users = %S, %D%w%S
#    browseable = No
#    read only = No
#    inherit acls = Yes
.
.
```

21.6 Creating a Samba User

Any user that requires access to a Samba shared resource must be configured as a Samba User and assigned a password. This task is achieved using the *smbpasswd* command-line tool. Consider, for example, that a user named demo is required to be able to access the */sampleshare* directory of our Ubuntu system from a Windows system. To fulfill this requirement, we must add demo as a Samba user as follows:

```
# smbpasswd -a demo
New SMB password:
Retype new SMB password:
Added user demo.
```

Now that we have completed the configuration of an elementary Samba server, it is time to test our configuration file and then start the Samba services.

21.7 Testing the smb.conf File

The settings in the *smb.conf* file may be checked for errors using the *testparm* command-line tool as follows:

```
# testparm
Load smb config files from /etc/samba/smb.conf
```

Loaded services file OK.

Weak crypto is allowed

Server role: ROLE_STANDALONE

Press enter to see a dump of your service definitions

Global parameters

```
[global]
    log file = /var/log/samba/%m.log
    netbios name = LINUXSERVER
    printcap name = cups
    security = USER
    wins support = Yes
    idmap config * : backend = tdb
    cups options = raw
```

```
[sampleshare]
    comment = Example Samba share
    guest ok = Yes
    path = /sampleshare
    read only = No
```

```
[homes]
    browseable = No
    comment = Home Directories
    inherit acls = Yes
    read only = No
    valid users = %S %D%w%S
```

```
[printers]
    browseable = No
    comment = All Printers
    create mask = 0600
    path = /var/tmp
    printable = Yes
```

.

.

21.8 Starting the Samba and NetBIOS Name Services

For an Ubuntu server to operate within a Windows network, the Samba (SMB) and NetBIOS nameservice (NMB) services must be started. Optionally, also enable the services so that they start each time the system boots:

```
# systemctl enable smbd nmbd
# systemctl start smbd nmbd
```

Sharing Files between Ubuntu 22.04 and Windows with Samba

Before attempting to connect from a Windows system, use the *smbclient* utility to verify that the share is configured:

```
# smbclient -U demo -L localhost
```

```
Password for [WORKGROUP\demo]:
```

Sharename	Type	Comment
-----	----	-----
sampleshare	Disk	Example Samba share
print\$	Disk	Printer Drivers
IPC\$	IPC	IPC Service (demosever server (Samba, Ubuntu))
demo	Disk	Home Directories
HP_OfficeJet_Pro_9020_series_9F6907 Printer		HP_OfficeJet_Pro_9020_series

21.9 Accessing Samba Shares

Now that the Samba resources are configured and the services are running, it is time to access the shared resource from a Windows system. On a suitable Windows system on the same workgroup as the Ubuntu system, open Windows Explorer and right-click on the Network entry in the side panel to display the menu shown in Figure 21-3:

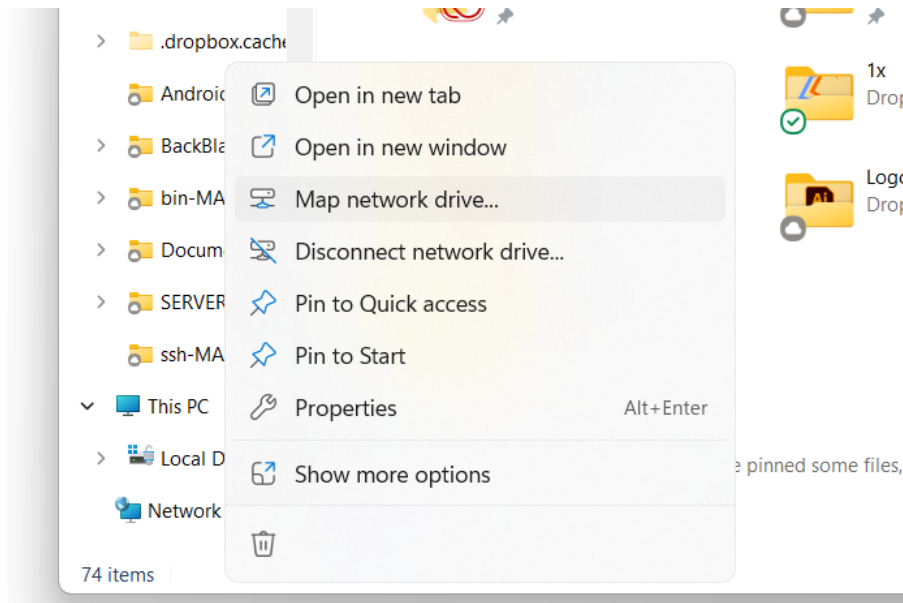


Figure 21-3

Select the *Map network drive...* menu option to display the dialog illustrated in Figure 21-4:

Sharing Files between Ubuntu 22.04 and Windows with Samba

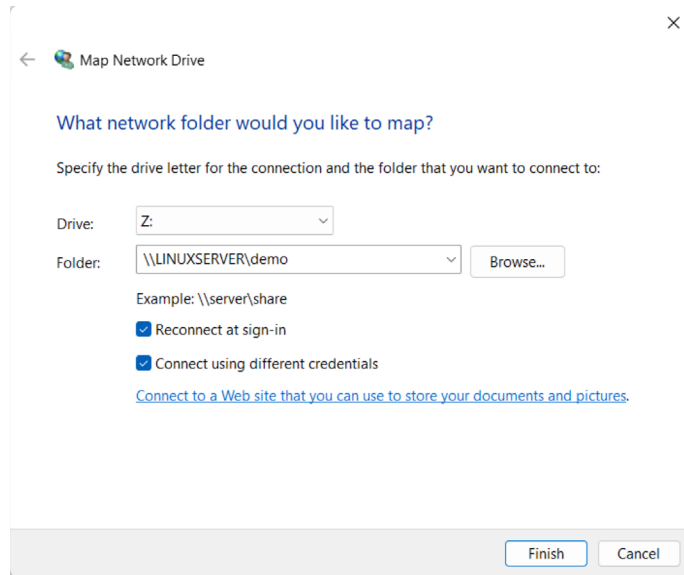


Figure 21-4

Select a drive letter and enter the path to a samba share. For example, assuming that the server name is LinuxServer and the samba user name is demo, the path to the user's home folder would be as follows:

```
\\LINUXSERVER\\demo
```

Enable the Connect using different credentials checkbox and click finish. When the network credentials dialog appears, enter the Samba user name and the password that was assigned earlier using the *smbpasswd* command:

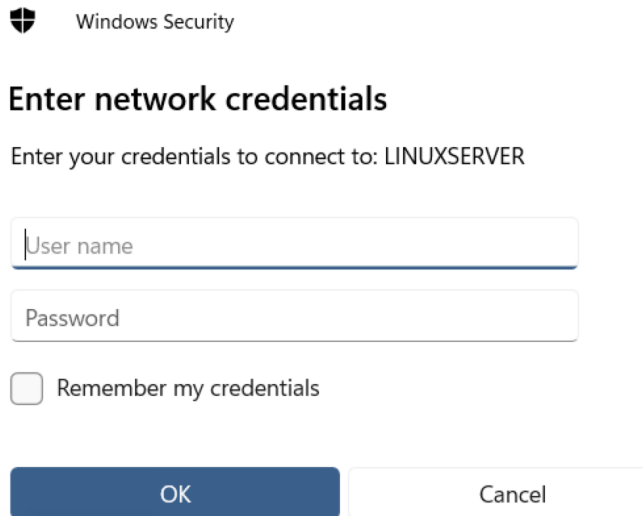


Figure 21-5

Sharing Files between Ubuntu 22.04 and Windows with Samba

After the connection is established, a new Windows Explorer dialog will appear containing the contents of the shared Ubuntu folder:

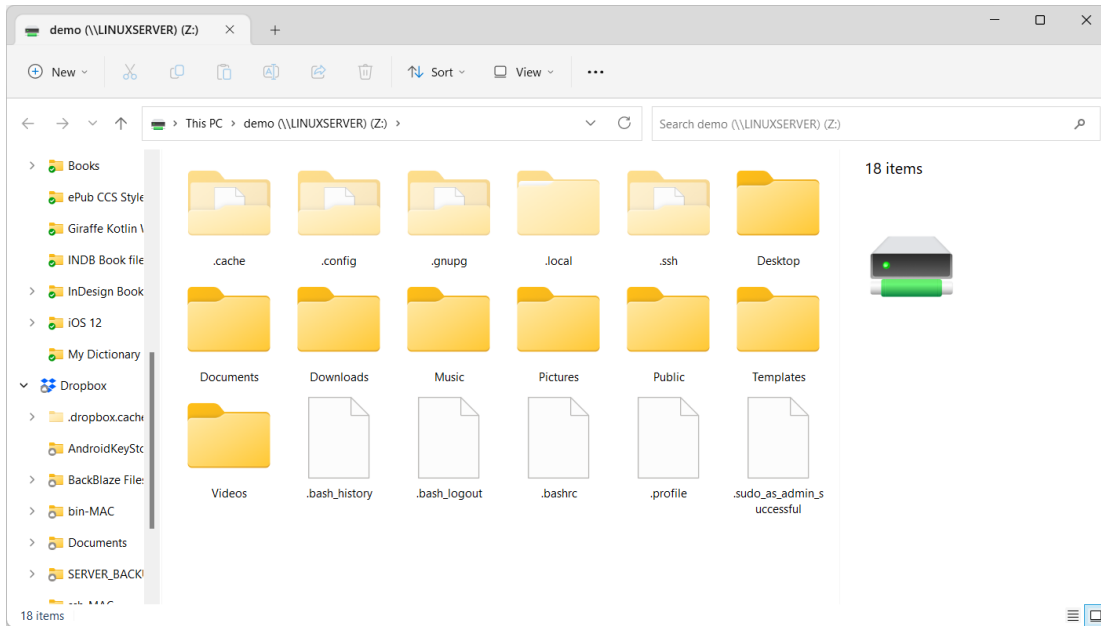


Figure 21-6

21.10 Accessing Windows Shares from Ubuntu

As previously mentioned, Samba is a two-way street, allowing not only Windows systems to access files and printers hosted on an Ubuntu system but also allowing the Ubuntu system to access shared resources on Windows systems. This is achieved using the *samba-client* package, installed at this chapter's start. If it is not currently installed, install it from a terminal window as follows:

```
# apt install smbclient
```

Shared resources on a Windows system can be accessed from the Ubuntu desktop using the Files application or the command-line prompt using the *smbclient* and *mount* tools. The steps in this section assume that the Windows system has enabled appropriate network-sharing settings.

To access any shared resources on a Windows system using the GNOME desktop, launch the Files application and select the Other Locations option. This will display the screen shown in Figure 21-7 below, including an icon for the Windows Network (if one is detected):

Sharing Files between Ubuntu 22.04 and Windows with Samba

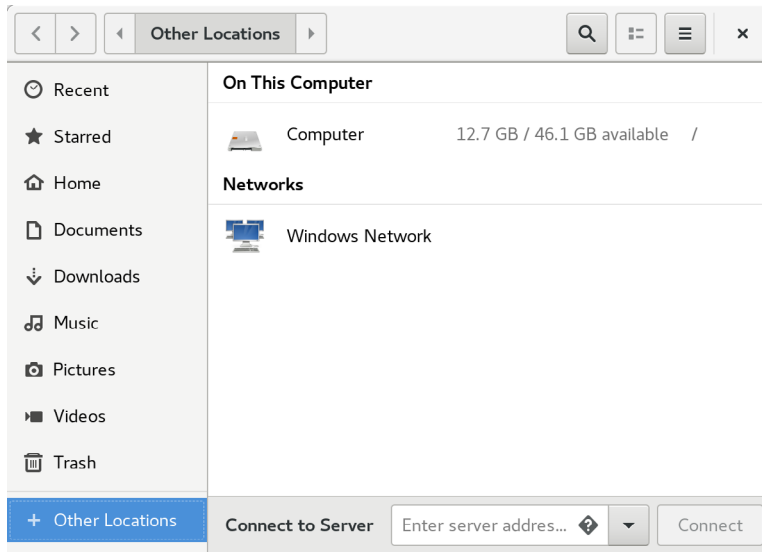


Figure 21-7

Selecting the Windows Network option will display the Windows systems detected on the network and allow access to any shared resources.

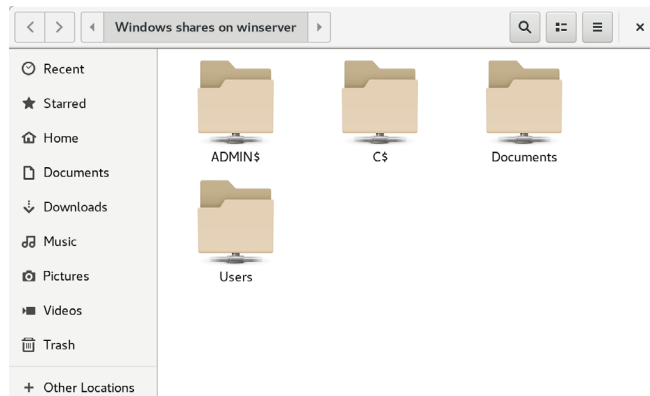


Figure 21-8

Alternatively, the Connect to Server option may be used to connect to a specific system. Note that the name or IP address of the remote system must be prefixed by *smb://* and may be followed by the path to a specific shared resource, for example:

```
smb://WinServer/Documents
```

Without a desktop environment, a remote Windows share may be mounted from the command line using the mount command and specifying the cifs filesystem type. The following command, for example, mounts a share named Documents located on a Windows system named WinServer at a local mount point named */winfiles*:

```
# mount -t cifs //WinServer/Documents /winfiles -o user=demo
```

21.11 Summary

In this chapter, we have looked at how to configure an Ubuntu system to act as both a Samba client and server, allowing the sharing of resources with Windows systems. Topics covered included the installation of Samba client and server packages and configuring Samba as a standalone server. In addition, the basic concepts of SELinux were introduced together with the steps to provide Samba access to a shared resource.

22. An Overview of Virtualization Techniques

Virtualization is the ability to run multiple operating systems simultaneously on a single computer system. While not necessarily a new concept, Virtualization has come to prominence in recent years because it provides a way to fully utilize the CPU and resource capacity of a server system while providing stability (in that if one virtualized guest system crashes, the host and any other guest systems continue to run).

Virtualization is also helpful in trying out different operating systems without configuring dual boot environments. For example, you can run Windows in a virtual machine without re-partitioning the disk, shut down Ubuntu, and boot from Windows. Instead, you start up a virtualized version of Windows as a guest operating system. Similarly, virtualization allows you to run other Linux distributions within an Ubuntu system, providing concurrent access to both operating systems.

When deciding on the best approach to implementing virtualization, clearly understanding the different virtualization solutions currently available is essential. Therefore, this chapter's purpose is to describe in general terms the virtualization techniques in common use today.

22.1 Guest Operating System Virtualization

Guest OS virtualization, also called application-based virtualization, is the most straightforward concept to understand. In this scenario, the physical host computer runs a standard unmodified operating system such as Windows, Linux, UNIX, or macOS. Running on this operating system is a virtualization application that executes in much the same way as any other application, such as a word processor or spreadsheet, would run on the system. Within this virtualization application, one or more virtual machines are created to run the guest operating systems on the host computer.

The virtualization application is responsible for starting, stopping, and managing each virtual machine and essentially controlling access to physical hardware resources on behalf of the individual virtual machines. The virtualization application also engages in a process known as binary rewriting, which involves scanning the instruction stream of the executing guest system and replacing any privileged instructions with safe emulations. This makes the guest system think it is running directly on the system hardware rather than in a virtual machine within an application.

The following figure illustrates guest OS-based virtualization:

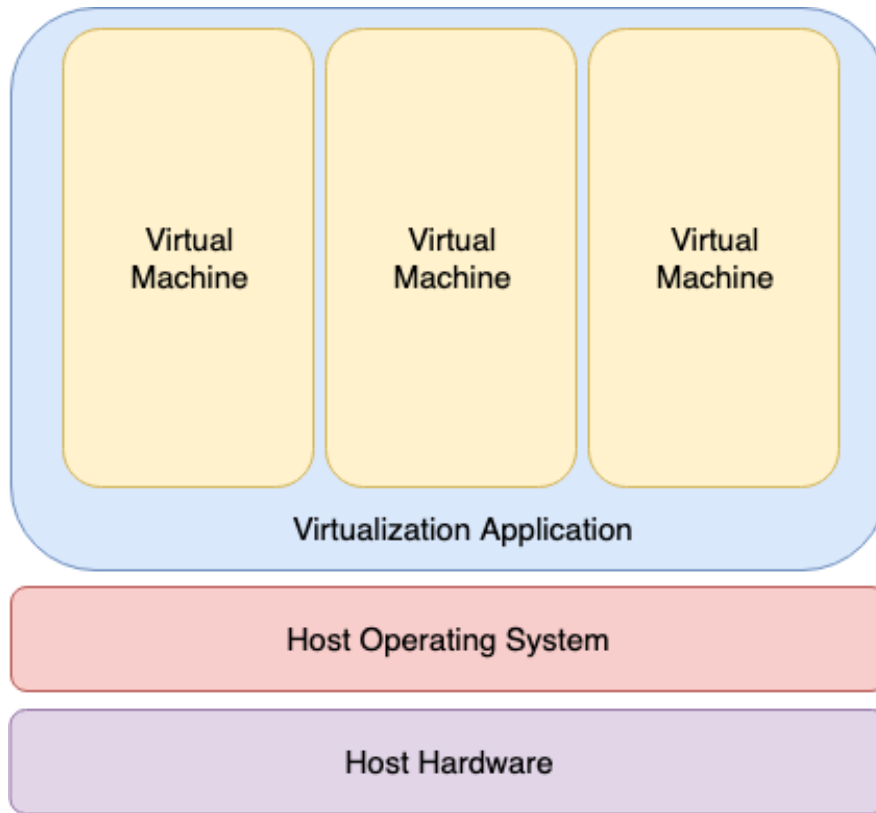


Figure 22-1

As outlined in the above diagram, the guest operating systems operate in virtual machines within the virtualization application, which, in turn, runs on top of the host operating system in the same way as any other application. The multiple layers of abstraction between the guest operating systems and the underlying host hardware are not conducive to high levels of virtual machine performance. However, this technique has the advantage that no changes are necessary to host or guest operating systems, and no special CPU hardware virtualization support is required.

22.2 Hypervisor Virtualization

In hypervisor virtualization, the task of a hypervisor is to handle resource and memory allocation for the virtual machines and provide interfaces for higher-level administration and monitoring tools. Hypervisor-based solutions are categorized as being either Type-1 or Type-2.

Type-2 hypervisors (sometimes called hosted hypervisors) are installed as software applications that run on top of the host operating system, providing virtualization capabilities by coordinating access to resources such as the CPU, memory, and network for guest virtual machines. Figure 21-2 illustrates the typical architecture of a system using Type-2 hypervisor virtualization:

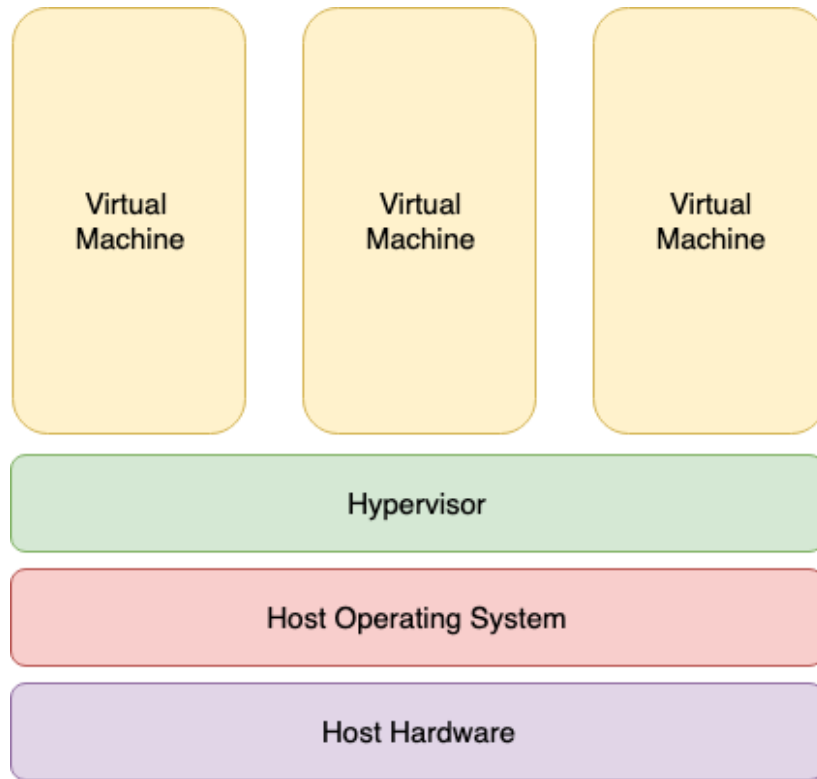


Figure 22-2

To understand how Type-1 hypervisors work, it helps to understand Intel x86 processor architecture. The x86 family of CPUs provides a range of protection levels known as rings in which code can execute. Ring 0 has the highest level privilege, and it is in this ring that the operating system kernel normally runs. Code executing in ring 0 is said to be running in system space, kernel mode, or supervisor mode. All other code, such as applications running on the operating system, operate in less privileged rings, typically ring 3.

In contrast to Type-2 hypervisors, Type-1 hypervisors (also referred to as metal or native hypervisors) run directly on the hardware of the host system in ring 0. With the hypervisor occupying ring 0 of the CPU, the kernels for any guest operating systems running on the system must run in less privileged CPU rings. Unfortunately, most operating system kernels are written explicitly to run in ring 0 because they need to perform tasks only available in that ring, such as the ability to execute privileged CPU instructions and directly manipulate memory. Several different solutions to this problem have been devised in recent years, each of which is described below:

22.2.1 Paravirtualization

Under paravirtualization, the kernel of the guest operating system is modified specifically to run on the hypervisor. This typically involves replacing privileged operations that only run in ring 0 of the CPU with calls to the hypervisor (known as hypercalls). The hypervisor, in turn, performs

An Overview of Virtualization Techniques

the task on behalf of the guest kernel. Unfortunately, this typically limits support to open-source operating systems such as Linux, which may be freely altered, and proprietary operating systems where the owners have agreed to make the necessary code modifications to target a specific hypervisor. These issues notwithstanding, the ability of the guest kernel to communicate directly with the hypervisor results in greater performance levels than other virtualization approaches.

22.2.2 Full Virtualization

Full virtualization provides support for unmodified guest operating systems. The term unmodified refers to operating system kernels that have not been altered to run on a hypervisor and, therefore, still execute privileged operations as though running in ring 0 of the CPU. In this scenario, the hypervisor provides CPU emulation to handle and modify privileged and protected CPU operations made by unmodified guest operating system kernels. Unfortunately, this emulation process requires both time and system resources to operate, resulting in inferior performance levels when compared to those provided by paravirtualization.

22.2.3 Hardware Virtualization

Hardware virtualization leverages virtualization features built into the latest generations of CPUs from both Intel and AMD. These technologies, called Intel VT and AMD-V, respectively, provide extensions necessary to run unmodified guest virtual machines without the overheads inherent in full virtualization CPU emulation. In very simplistic terms, these processors provide an additional privilege mode (ring -1) above ring 0 in which the hypervisor can operate, thereby leaving ring 0 available for unmodified guest operating systems.

The following figure illustrates the Type-1 hypervisor approach to virtualization:

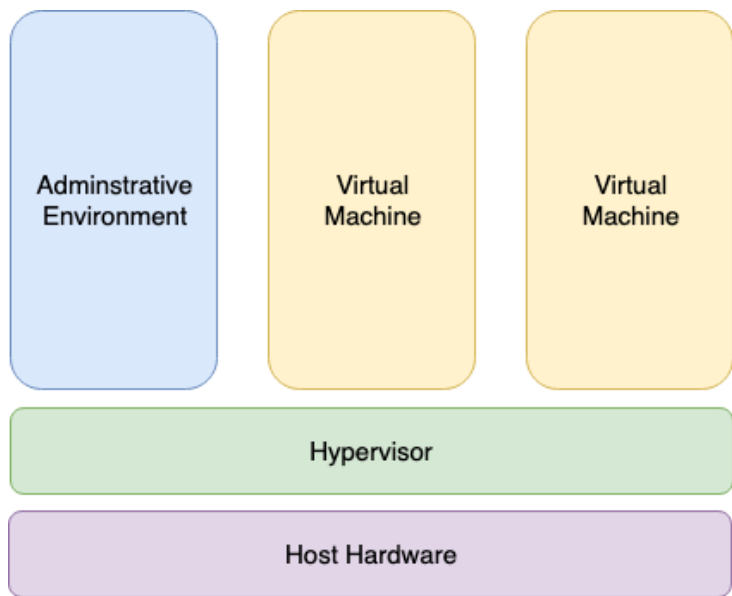


Figure 22-3

As outlined in the above illustration, in addition to the virtual machines, an administrative

192

operating system or management console also runs on top of the hypervisor allowing the virtual machines to be managed by a system administrator.

22.3 Virtual Machine Networking

Virtual machines will invariably need to be connected to a network to be of any practical use. One option is for the guest to be connected to a virtual network running within the host computer's operating system. In this configuration, any virtual machines on the virtual network can see each other, but Network Address Translation (NAT) provides access to the external network. When using the virtual network and NAT, each virtual machine is represented on the external network (the network to which the host is connected) using the IP address of the host system. This is the default behavior for KVM virtualization on Ubuntu and generally requires no additional configuration. Typically, a single virtual network is created by default, represented by the name *default* and the device *virbr0*.

For guests to appear as individual and independent systems on the external network (i.e., with their own IP addresses), they must be configured to share a physical network interface on the host. The quickest way to achieve this is to configure the virtual machine to use the “direct connection” network configuration option (also called MacVTap), which will provide the guest system with an IP address on the same network as the host. Unfortunately, while this gives the virtual machine access to other systems on the network, it is not possible to establish a connection between the guest and the host when using the MacVTap driver.

A better option is to configure a network bridge interface on the host system to which the guests can connect. This provides the guest with an IP address on the external network while also allowing the guest and host to communicate, a topic covered in the chapter entitled “*Creating an Ubuntu 22.04 KVM Networked Bridge Interface*”.

22.4 Summary

Virtualization is the ability to run multiple guest operating systems within a single host operating system. Several approaches to virtualization have been developed, including a guest operating system and hypervisor virtualization. Hypervisor virtualization falls into two categories known as Type-1 and Type-2. Type-2 virtualization solutions are categorized as paravirtualization, full virtualization, and hardware virtualization, the latter using special virtualization features of some Intel and AMD processor models.

Virtual machine guest operating systems have several options in terms of networking, including NAT, direct connection (MacVTap), and network bridge configurations.

29. An Introduction to Linux Containers

The preceding chapters covered the concept of virtualization, emphasizing creating and managing virtual machines using KVM. This chapter will introduce a related technology in the form of Linux Containers. While there are some similarities between virtual machines and containers, key differences will be outlined in this chapter, along with an introduction to the concepts and advantages of Linux Containers. The chapter will also introduce some Ubuntu container management tools. Once the basics of containers have been covered in this chapter, the next chapter will work through some practical examples of creating and running containers on Ubuntu.

29.1 Linux Containers and Kernel Sharing

In simple terms, Linux containers are a lightweight alternative to virtualization. A virtual machine contains and runs the entire guest operating system in a virtualized environment. The virtual machine, in turn, runs on top of an environment such as a hypervisor that manages access to the physical resources of the host system.

Containers work by using a concept referred to as kernel sharing, which takes advantage of the architectural design of Linux and UNIX-based operating systems.

To understand how kernel sharing and containers work, it helps first to understand the two main components of Linux or UNIX operating systems. At the core of the operating system is the kernel. In simple terms, the kernel handles all the interactions between the operating system and the physical hardware. The second key component is the root file system which contains all the libraries, files, and utilities necessary for the operating system to function. Taking advantage of this structure, containers each have their own root file system but share the host operating system's kernel. This structure is illustrated in the architectural diagram in Figure 29-1 below.

This type of resource sharing is made possible by the ability of the kernel to dynamically change the current root file system (a concept known as change root or chroot) to a different root file system without having to reboot the entire system. Linux containers are essentially an extension of this capability combined with a container runtime, the responsibility of which is to provide an interface for executing and managing the containers on the host system. Several container runtimes are available, including Docker, lxd, containerd, and CRI-O.

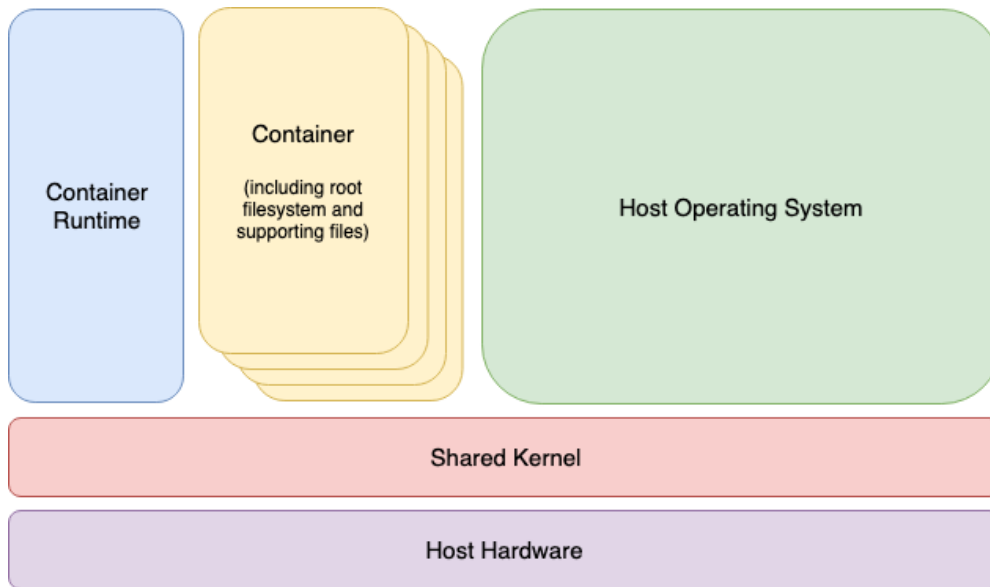


Figure 29-1

29.2 Container Uses and Advantages

The main advantage of containers is that they require considerably less resource overhead than virtualization allowing many container instances to be run simultaneously on a single server. They can be started and stopped rapidly and efficiently in response to demand levels. In addition, containers run natively on the host system providing a level of performance that a virtual machine cannot match.

Containers are also highly portable and can be easily migrated between systems. Combined with a container management system such as Docker, OpenShift, and Kubernetes, it is possible to deploy and manage containers on a vast scale spanning multiple servers and cloud platforms, potentially running thousands of containers.

Containers are frequently used to create lightweight execution environments for applications. In this scenario, each container provides an isolated environment containing the application together with all of the runtime and supporting files required by that application to run. The container can then be deployed to any other compatible host system that supports container execution and runs without any concerns that the target system may not have the necessary runtime configuration for the application - all of the application's dependencies are already in the container.

Containers are also helpful when bridging the gap between development and production environments. By performing development and QA work in containers, they can be passed to production and launched safely because the applications run in the same container environments in which they were developed and tested.

Containers also promote a modular approach to deploying large and complex solutions. Instead of developing applications as single monolithic entities, containers can be used to design applications

as groups of interacting modules, each running in a separate container.

One possible drawback of containers is that the guest operating systems must be compatible with the shared kernel version. It is not, for example, possible to run Microsoft Windows in a container on a Linux system. Nor is it possible for a Linux guest system designed for the 2.6 version of the kernel to share a 2.4 version kernel. These requirements are not, however, what containers were designed for. Rather than being seen as limitations, these restrictions should be considered some of the key advantages of containers in providing a simple, scalable, and reliable deployment platform.

29.3 Ubuntu Container Tools

Ubuntu provides several tools for creating, inspecting, and managing containers. The main tools are as follows:

- **buildah** – A command-line tool for building container images.
- **podman** – A command-line based container runtime and management tool. Performs tasks such as downloading container images from remote registries and inspecting, starting, and stopping images.
- **skopeo** – A command-line utility used to convert container images, copy images between registries and inspect images stored in registries without downloading them.
- **runc** – A lightweight container runtime for launching and running containers from the command line.
- **OpenShift** – An enterprise-level container application management platform consisting of command-line and web-based tools.

All of the above tools comply with the Open Container Initiative (OCI), a set of specifications designed to ensure that containers conform to the same standards between competing tools and platforms.

29.4 The Ubuntu Docker Registry

Although Ubuntu is provided with a set of tools designed to be used in place of those provided by Docker, those tools still need access to Ubuntu images for use when building containers. For this purpose, the Ubuntu team maintains a set of Ubuntu container images within the Docker Hub. The Docker Hub is an online container *registry* made of multiple *repositories*, each containing a wide range of container images available for download when building containers. The images within a repository are each assigned a *repository tag* (for example, 21.04, 20.10, 22.04, latest etc.) which can be referenced when performing an image download. The following, for example, is the URL of the Ubuntu 22.04 image contained within the Docker Hub:

```
docker://docker.io/library/ubuntu:22.04
```

In addition to downloading (referred to as “pulling” in container terminology) container images

An Introduction to Linux Containers

from Docker and other third party hosts registries, you can also use registries to store your own images. This can be achieved either by hosting your own registry, or by making use of existing services such as those provided by Docker, Amazon AWS, Google Cloud, Microsoft Azure and IBM Cloud to name a few of the many options.

29.5 Container Networking

By default, containers are connected to a network using a Container Networking Interface (CNI) bridged network stack. In the bridged configuration, all the containers running on a server belong to the same subnet and, as such, can communicate with each other. The containers are also connected to the external network by bridging the host system's network connection. Similarly, the host can access the containers via a virtual network interface (usually named `podman0`) which will have been created as part of the container tool installation.

29.6 Summary

Linux Containers offer a lightweight alternative to virtualization and take advantage of the structure of the Linux and Unix operating systems. Linux Containers share the host operating system's kernel, with each container having its own root file system containing the files, libraries, and applications. As a result, containers are highly efficient and scalable and provide an ideal platform for building and deploying modular enterprise-level solutions. In addition, several tools and platforms are available for building, deploying, and managing containers, including third-party solutions and those provided with Ubuntu.

36. Ubuntu 22.04 System and Process Monitoring

An essential part of running and administering an Ubuntu system involves monitoring the overall system health regarding memory, swap, storage, and processor usage. This includes knowing how to inspect and manage the system and user processes running in the background. This chapter will outline some tools and utilities that can be used to monitor system resources and processes on an Ubuntu system.

36.1 Managing Processes

Even when an Ubuntu system appears idle, many *system processes* will run silently in the background to keep the operating system functioning. For example, when you execute a command or launch an app, *user processes* are started, running until the associated task is completed.

To obtain a list of active user processes you are currently running within the context of a single terminal or command-prompt session, use the *ps* command as follows:

```
$ ps
  PID TTY          TIME CMD
10395 pts/1    00:00:00 bash
13218 pts/1    00:00:00 ps
```

The output from the *ps* command shows that two user processes are running within the context of the current terminal window or command prompt session, the bash shell into which the command was entered and the *ps* command itself.

To list all active processes running for the current user, use the *ps* command with the *-a* flag. This command will list all running processes that are associated with the user regardless of where they are running (for example, processes running in other terminal windows):

```
$ ps -a
  PID TTY          TIME CMD
 5442 tty2          00:00:00 gnome-session-b
 6350 pts/0          00:00:00 sudo
 6354 pts/0          00:00:00 su
 6355 pts/0          00:00:00 bash
 9849 pts/2          00:00:00 nano
 9850 pts/1          00:00:00 ps
```

As shown in the above output, the user is running processes related to the GNOME desktop, the shell session, the *nano* text editor, and the *ps* command.

To list the processes for a specific user, run *ps* with the *-u* flag followed by the user name:

Ubuntu 22.04 System and Process Monitoring

```
# ps -u john
  PID TTY          TIME CMD
  914 ?            00:00:00 systemd
  915 ?            00:00:00 (sd-pam)
  970 ?            00:00:00 gnome-keyring-d
  974 tty1        00:00:00 gdm-x-session
.
.
```

Note that each process is assigned a unique process ID which can be used to stop the process by sending it a termination (TERM) signal via the *kill* command. For example:

```
$ kill 13217
```

The advantage of ending a process with the TERM signal is that it allows the process to exit gracefully, potentially saving any data that might otherwise be lost.

If the standard termination signal does not terminate the process, repeat the *kill* command with the -9 option. This command sends a KILL signal which should cause even frozen processes to exit but does not give the process a chance to exit gracefully, possibly resulting in data loss:

```
$ kill -9 13217
```

To list all of the processes running on a system (including all user and system processes), execute the following command:

```
$ ps -ax
  PID TTY          STAT TIME COMMAND
    1 ?            Ss   0:22 /usr/lib/systemd/systemd rhgb --switched-root
    2 ?            S    0:00 [kthreadd]
    3 ?            I<   0:00 [rcu_gp]
    4 ?            I<   0:00 [rcu_par_gp]
    5 ?            I<   0:00 [netns]
```

To list all processes and include information about process ownership, CPU, and memory use, execute the *ps* command with the -aux option:

```
$ ps -aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           2  0.0  0.0      0      0 ?        S      09:59    0:00 [kthreadd]
root           3  0.0  0.0      0      0 ?        I<     09:59    0:00 [rcu_gp]
root           4  0.0  0.0      0      0 ?        I<     09:59    0:00 [rcu_par_gp]
root           5  0.0  0.0      0      0 ?        I<     09:59    0:00 [netns]
root           7  0.0  0.0      0      0 ?        I<     09:59    0:00 [kworker/0:0H-
events_highpri]
root           9  0.0  0.0      0      0 ?        I<     09:59    0:00 [kworker/0:1H-
events_highpri]
.
.
demo          9788  0.1  1.4 763248 50480 ?        Ss1    15:05    0:00 /usr/libexec/
```

```
gnome-terminal-serv
```

```
demo          9814  0.0  0.1 224108  5664 pts/2    Ss   15:05   0:00 bash
demo          9849  0.0  0.0 222412  3588 pts/2    S+   15:06   0:00 nano
demo          9873  0.0  0.1 233416  6280 pts/1    R+   15:08   0:00 ps -aux
```

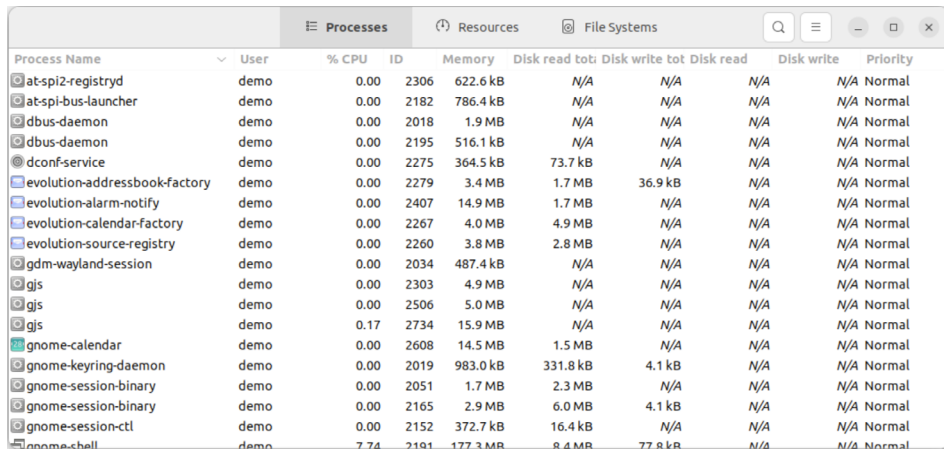
A Linux process can start its own sub-processes (referred to as *spawning*), resulting in a hierarchical parent-child relationship between processes. To view the process tree, use the `ps` command and include the `-H` option. Below is part of the tree output for a `ps -aH` command execution:

```
$ ps -aH
  PID TTY          TIME CMD
 10036 pts/3        00:00:00 ps
   6350 pts/0        00:00:00 sudo
   6354 pts/0        00:00:00 su
   6355 pts/0        00:00:00 bash
  5442 tty2        00:00:00 gnome-session-b
```

Process information may also be viewed via the System Monitor tool from the GNOME desktop. This tool can either be launched by searching for “System Monitor” within the desktop environment or from the command line as follows:

```
$ gnome-system-monitor
```

Once the System Monitor has launched, select the Processes button located in the toolbar to list the processes running on the system, as shown in Figure 36-1 below:



Process Name	User	% CPU	ID	Memory	Disk read tot	Disk write tot	Disk read	Disk write	Priority
at-spi2-registr	demo	0.00	2306	622.6 kB	N/A	N/A	N/A	N/A	Normal
at-spi-bus-launcher	demo	0.00	2182	786.4 kB	N/A	N/A	N/A	N/A	Normal
dbus-daemon	demo	0.00	2018	1.9 MB	N/A	N/A	N/A	N/A	Normal
dbus-daemon	demo	0.00	2195	516.1 kB	N/A	N/A	N/A	N/A	Normal
dconf-service	demo	0.00	2275	364.5 kB	73.7 kB	N/A	N/A	N/A	Normal
evolution-addressbook-factory	demo	0.00	2279	3.4 MB	1.7 MB	36.9 kB	N/A	N/A	Normal
evolution-alarm-notify	demo	0.00	2407	14.9 MB	1.7 MB	N/A	N/A	N/A	Normal
evolution-calendar-factory	demo	0.00	2267	4.0 MB	4.9 MB	N/A	N/A	N/A	Normal
evolution-source-registry	demo	0.00	2260	3.8 MB	2.8 MB	N/A	N/A	N/A	Normal
gdm-wayland-session	demo	0.00	2034	487.4 kB	N/A	N/A	N/A	N/A	Normal
gjs	demo	0.00	2303	4.9 MB	N/A	N/A	N/A	N/A	Normal
gjs	demo	0.00	2506	5.0 MB	N/A	N/A	N/A	N/A	Normal
gjs	demo	0.17	2734	15.9 MB	N/A	N/A	N/A	N/A	Normal
gnome-calendar	demo	0.00	2608	14.5 MB	1.5 MB	N/A	N/A	N/A	Normal
gnome-keyring-daemon	demo	0.00	2019	983.0 kB	331.8 kB	4.1 kB	N/A	N/A	Normal
gnome-session-binary	demo	0.00	2051	1.7 MB	2.3 MB	N/A	N/A	N/A	Normal
gnome-session-binary	demo	0.00	2165	2.9 MB	6.0 MB	4.1 kB	N/A	N/A	Normal
gnome-session-ctl	demo	0.00	2152	372.7 kB	16.4 kB	N/A	N/A	N/A	Normal
gnome-shell	demo	7.74	2191	177.3 MB	8.4 MB	77.8 kB	N/A	N/A	Normal

Figure 36-1

To change the processes listed (for example, to list all processes or just your own processes), use the menu as illustrated in Figure 36-2:

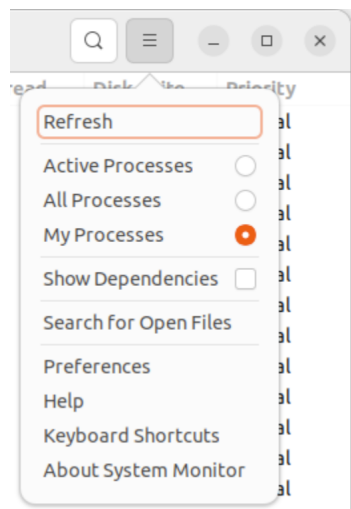


Figure 36-2

To filter the list of processes, click on the search button in the title bar and enter the process name into the search field:

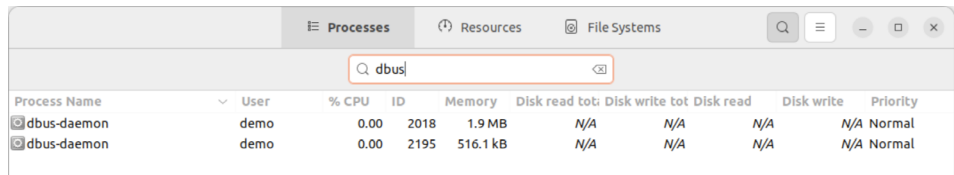


Figure 36-3

To display additional information about a specific process, select it from the list and click on the button located in the bottom right-hand corner (marked A in Figure 36-4) of the dialog:

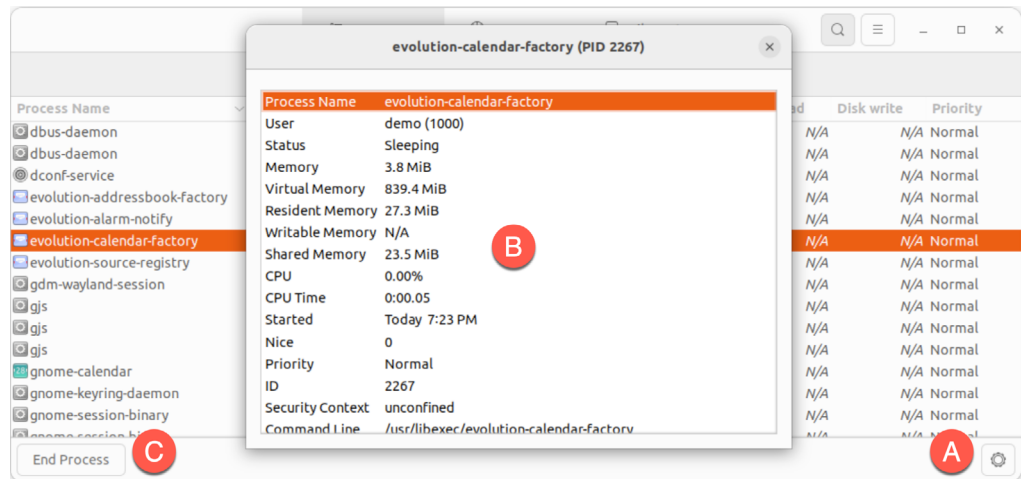


Figure 36-4

A dialog similar to that marked B in the above figure will appear when the button is clicked. Select a process from the list and click the End Process button (C) to terminate it.

To monitor CPU, memory, swap, and network usage, click on the Resources button in the title bar to display the screen shown in Figure 36-5:

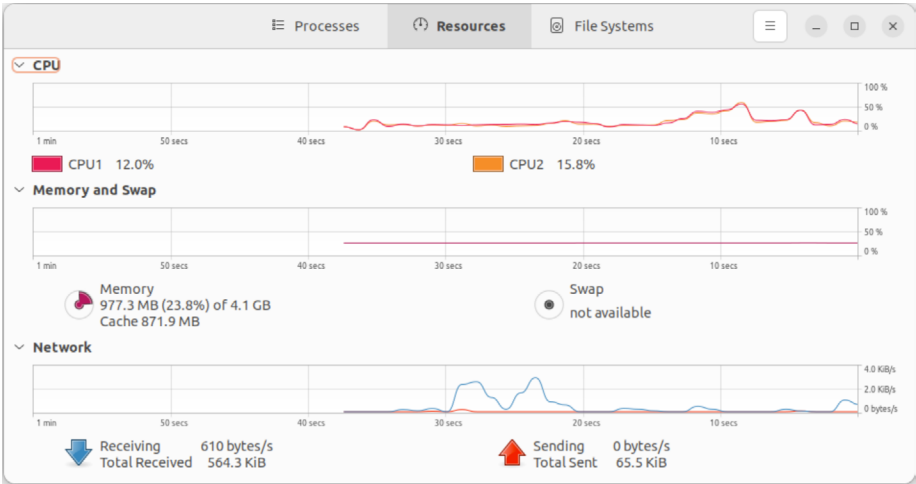


Figure 36-5

Similarly, a summary of storage space used on the system can be viewed by selecting the File Systems toolbar button:

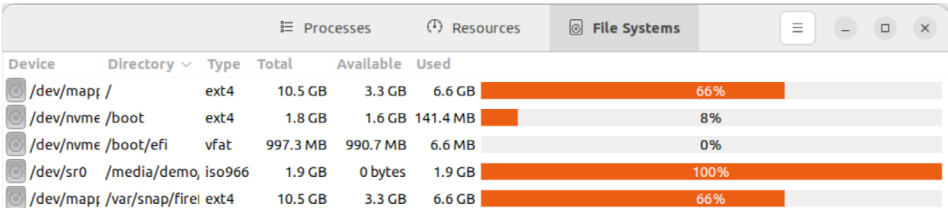


Figure 36-6

36.2 Real-time System Monitoring with top

As the chapter “An Overview of the Cockpit Web Interface” outlined, the Cockpit web interface can perform basic system monitoring. The previous section also explained how the GNOME System Monitor tool could be used to monitor processes and system resources. This chapter also explored how the *ps* command can provide a snapshot of the processes running on an Ubuntu system. However, the *ps* command does not provide a real-time view of the processes and resource usage on the system. The *top* command is an ideal tool for real-time monitoring of system resources and processes from the command prompt.

When running, *top* will list the processes running on the system ranked by system resource usage (with the most demanding process in the *top* position). The upper section of the screen displays

Ubuntu 22.04 System and Process Monitoring

memory and swap usage information together with CPU data for all CPU cores. All of this output is constantly updated, allowing the system to be monitored in real-time:

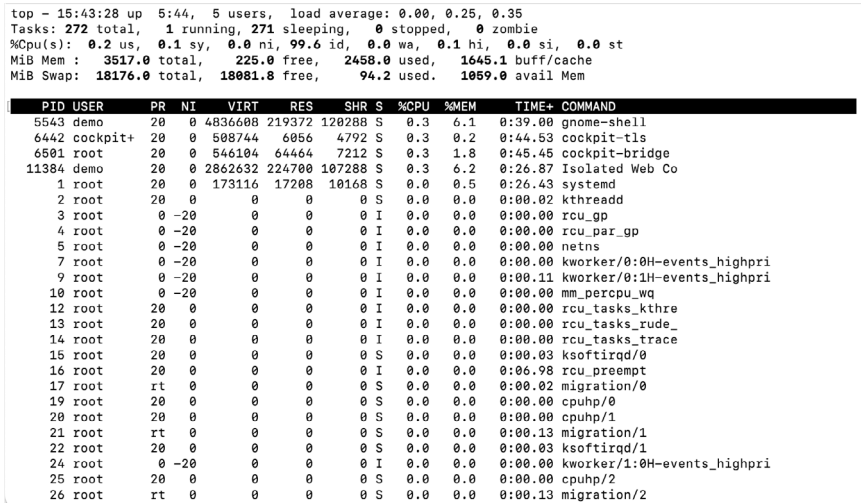


Figure 36-7

To limit the information displayed to the processes belonging to a specific user, start `top` with the `-u` option followed by the user name:

```
$ top -u john
```

For a complete listing of the features available in `top`, press the keyboard ‘h’ key or refer to the man page:

```
$ man top
```

36.3 Command-Line Disk and Swap Space Monitoring

Disk space can be monitored from within Cockpit and using the GNOME System Monitor. To identify disk usage from the command line, however, the `df` command provides a helpful and quick overview:

To review current swap space and memory usage, run the `free` command:

```
# free
```

	total	used	free	shared	buff/cache	available
Mem:	3823720	879916	1561108	226220	1382696	2476300

To continuously monitor memory and swap levels, use the `free` command with the `-s` option, specifying the delay in seconds between each update (keeping in mind that the `top` tool may provide a better way to view this data in real-time):

```
$ free -s 1
```

	total	used	free	shared	buff/cache	available
Mem:	3823720	879472	1561532	226220	1382716	2476744
Swap:	2097148	0	2097148			

```
Mem:          3823720      879140      1559940      228144      1384640      2475152
Swap:         2097148          0      2097148
.
.
```

To monitor disk I/O from the command line, consider using the *iotop* command, which can be installed as follows:

```
# apt install iotop
```

Once installed and executed (*iotop* must be run with system administrator privileges), the tool will display a real-time list of disk I/O on a per-process basis:

Total DISK READ :		0.00 B/s	Total DISK WRITE :		0.00 B/s
Actual DISK READ:		0.00 B/s	Actual DISK WRITE:		0.00 B/s
TID	PRI	USER	DISK READ	DISK WRITE	COMMAND
1	be/4	root	0.00 B/s	0.00 B/s	systemd rhgb --switched-root --system --deserialize 31
2	be/4	root	0.00 B/s	0.00 B/s	[kthreadd]
3	be/4	root	0.00 B/s	0.00 B/s	[rcu_gp]
4	be/4	root	0.00 B/s	0.00 B/s	[rcu_par_gp]
5	be/4	root	0.00 B/s	0.00 B/s	[netns]
7	be/4	root	0.00 B/s	0.00 B/s	[kworker/0:0H-events_highpri]
9	be/4	root	0.00 B/s	0.00 B/s	[kworker/0:1H-events_highpri]
10	be/4	root	0.00 B/s	0.00 B/s	[mm_percpu_wq]
12	be/4	root	0.00 B/s	0.00 B/s	[rcu_tasks_kthre]
13	be/4	root	0.00 B/s	0.00 B/s	[rcu_tasks_rude_]
14	be/4	root	0.00 B/s	0.00 B/s	[rcu_tasks_trace]
15	be/4	root	0.00 B/s	0.00 B/s	[ksoftirqd/0]
16	be/4	root	0.00 B/s	0.00 B/s	[rcu_preempt]
17	be/4	root	0.00 B/s	0.00 B/s	[migration/0]
19	be/4	root	0.00 B/s	0.00 B/s	[cpuhp/0]
20	be/4	root	0.00 B/s	0.00 B/s	[cpuhp/1]
21	be/4	root	0.00 B/s	0.00 B/s	[migration/1]
22	be/4	root	0.00 B/s	0.00 B/s	[ksoftirqd/1]
24	be/4	root	0.00 B/s	0.00 B/s	[kworker/1:0H-events_highpri]
25	be/4	root	0.00 B/s	0.00 B/s	[cpuhp/2]
26	be/4	root	0.00 B/s	0.00 B/s	[migration/2]
27	be/4	root	0.00 B/s	0.00 B/s	[ksoftirqd/2]
29	be/4	root	0.00 B/s	0.00 B/s	[kworker/2:0H-events_highpri]
30	be/4	root	0.00 B/s	0.00 B/s	[cpuhp/3]
31	be/4	root	0.00 B/s	0.00 B/s	[migration/3]
32	be/4	root	0.00 B/s	0.00 B/s	[ksoftirqd/3]
34	be/4	root	0.00 B/s	0.00 B/s	[kworker/3:0H-events_highpri]
37	be/4	root	0.00 B/s	0.00 B/s	[kdevtmpfs]

Figure 36-8

36.4 Summary

Even a system that appears to be doing nothing will have many system processes running in the background. Activities performed by users on the system will result in additional processes being started. Processes can also spawn their own child processes. Each process will use some system resources, including memory, swap space, processor cycles, disk storage, and network bandwidth. This chapter has explored a set of tools that can be used to monitor both process and system resources on a running system and, when necessary, kill errant processes that may be impacting the performance of a system.

Index

Symbols

! 73
 #! 77
 >> 75
 | 75
 \$DISPLAY variable 168
 .bashrc 77
 /etc/exports 174
 /etc/fstab 33, 37, 175, 177, 265
 /etc/gdm/custom.conf 168
 /etc/group 81
 /etc/httpd 245
 /etc/passwd 81
 /etc/samba/smb.conf 181
 /etc/shadow 81
 /etc/sshd_config 168
 /etc/ssh/sshd_config.d 168
 /etc/sudoers 81
 /etc/systemd/system 92
 /home 79
 /proc/swaps file 276
 .requires 92
 .ssh 150, 151
 /usr/lib/systemd/system 92
 /var/log/maillog 258
 .wants 92

A

access control list 202
 ACL 202
 Advanced Package Tool 95
 AIX 7
 alias 75
 Aliases 75
 AMD-V 192

Andrew S. Tanenbaum 7
 Apache
 mod_ssl 248
 Apache web server 243
 apt 95
 edit-sources 96
 install 98
 package management 98
 purge 98
 remove 98
 search 99
 show 99
 sources list 96
 update 98
 upgrade 100
 apt-file 99
 apt list 98
 authorized_keys file 154

B

Bash
 scripts 77
 Bash shell 71
 aliases 75
 .bashrc 77
 chmod 78
 command-line editing 72
 do loops 77
 echo 76
 environment variables 76
 filename completion 74
 Filename shorthand 74
 for loop 77
 history 73
 HOME 76
 input and output redirection 74
 PATH 76
 path completion 74
 pipes 75

Index

- sh 78
- stderr 75
- stdin 74
- stdout 74
- basic.target 88
- Bell Labs 71
- Boot Menu
 - editing 41
- Bourne Again SHell 71
- Bourne shell 71
- Brian Fox 71
- buildah 235

C

- CA 248
- Canonical Ltd 7, 8
- cat 74
- CentOS
 - history of 7
- certbot 249
- Certificate Authority 248
- change root 233
- chmod 78
- chroot 233
- cifs filesystem 187
- CNI 236
- Cockpit 92
 - accessing 60
 - account management 64
 - applications 65
 - cockpit-machines 199
 - cockpit-storaged 176
 - create VM 199
 - Drives 266
 - enabling 60
 - extensions 59, 65
 - installing 60
 - logs 62
 - Multiple Servers 67
 - networking 63
 - NFS 175

- overview 59
- persistent metrics 68
- port 60
- select bridge 223
- services 64, 92
- storage 63, 265
- system 61
- systemd 92
- terminal access 66
- user management 81
- virtual machines 65
- cockpit-machines 199
- cockpit.socket 60
- cockpit-storaged 176
- Compressed X11 Forwarding 169
- Connection Profiles 118
- containerd 233
- Container Networking Interface 236
- Containers
 - overview 233
 - pull image 237
 - running image 239
 - save to image 241
- CRI-O 233
- C shell 71

D

- daemon 87
- David Korn 71
- dd 11, 276
- DDNS 243
- Debian 8
- default.target 90
- df 274, 286
- disk drive
 - detecting 261
- disk I/O 287
- Disk partition
 - formatting 40
- disk usage 286
- diskutil 12

DISPLAY variable 168

dmesg 11

DNS 126

DNS MX Records 257

Docker 233, 234

do loops 77

Domain Name Server 126

dual boot 37

Dynamic DNS 243

DynDNS 243

E

echo 76

Email Server 253

env 76

Environment Variables 76

Errata 5

Exim 253

export 76

exportfs 174

ext2 274

ext3 274

ext4 274

F

fdisk 37, 262, 272

 create partition 262

 list partitions 262

Fedora Media Writer 13

Fetchmail 253

Filename Shorthand 74

File System

 creating 263

 mounting 264

File Transfer (Control) 125

File Transfer Protocol (Data) 125

findmnt 11

firewall

 gufw 129

 ufw 129

Firewall

 overview 123, 139

 web server settings 244

firewall-cmd 125, 142

 mail settings 255

 NFS settings 173

firewall-config 146

firewalld

 default zone 142

 display zone information 142

 firewall-cmd 142

 firewall-config 146

 ICMP rules 145

 interfaces 139, 141

 list services 143

 overview 139

 permanent settings 142

 port forwarding 145

 port rules 143, 144

 ports 139, 141

 reload 142

 runtime settings 142

 services 139

 status 141

 zone creation 144

 zone/interface assignments 144

 zones 139

 zone services 143

for 77

ForwardX11Trusted 169

FQDN 126

free 276

 -s flag 286

Free Software Foundation 8

fsck 264

fstab 175, 265

FTP 123, 125

Full Virtualization 192

G

GDM 20

gedit 169

Index

- getfacl 202
- GNOME 3 47
- GNOME desktop
 - installing 47
- GNOME Desktop 47
 - installation 47
 - starting 47
- GNOME Display Manager 20
- gnome-system-monitor 283
- GNU/Linux 8
- GNU project 8
- graphical.target 88
- groupadd 80
- groupdel 80
- groups 80
- Guest OS virtualization 189
- gufw 129
 - adding simple rules 133
 - advanced rules 134
 - allow 132
 - deny 132
 - enabling 129
 - installing 129
 - limit 133
 - preconfigured rules 132
 - profiles 130
 - reject 132
 - running 129

H

- Hardware Virtualization 192
- Hewlett-Packard 7
- hibernation 276
- history 72, 73
- HOME 76
- HP-UX 7
- HTTP 127, 247
- httpd.conf 249
- httpd-le-ssl.conf 249
- HTTPS 123, 128, 248
- hypercalls 191

- Hypertext Text Transfer Protocol 127
- Hypertext Transfer Protocol Secure 128
- Hypervisor 190
 - hypercalls 191
 - type-1 190
 - type-2 190
- Hypervisor Virtualization 190

I

- Ian Murdoch 8
- IBM 7
- id_rsa file 150, 151, 153
- id_rsa.pub file 150, 154
- if statements 77
- IMAP 126
- IMAP4 127
- Input and Output Redirection 74
- Installation
 - clean disk 9
- Intel VT 192
- Intel x86
 - Rings 191
- Internet Message Access Protocol, Version 4 127
- internet service provider 243
- I/O redirection 75
- iotop 287
 - installing 287
- iptables 123, 125, 139
 - rules 124
 - tool 124
- ip tool 116
- ISO image
 - write to USB drive 11
- ISP 243

J

- Journalled File Systems 263

K

- Kerberos 127
- kernel 7

kill 282
 -9 flag 282
 KMail 253
 Korn shell 71
 Kubernetes 234
 KVM
 hardware requirements 195
 installation 196
 overview 195
 virt-manager 196
 kvm_amd 196
 kvm_intel 196
 KVM virtualization 193

L

LE 270
 Let's Encrypt 248
 libvirt 207
 libvirt daemon 197
 Linus Torvalds 7
 Linux Containers. *See* Containers
 Livepatch 102
 enabling 102
 Logical Extent 270
 Logical Volume 270
 Logical Volume Management 269
 loopback interface 115
 lost+found 264
 ls 72, 75
 lscpu 195
 lsmod 196
 LV 270
 lvdisplay 271, 274, 277
 lvextend 274
 LVM 269
 lxd 233

M

macOS
 writing ISO to USB drive 12
 MacVTap 193

Mail Delivery Agent 253
 Mail Exchanger 257
 Mail Transfer Agent 253
 Mail User Agent 253
 main.cf 255
 man 72
 Mark Shuttleworth 7, 8
 Martin Hellman 149
 MDA 253
 MINIX 7
 mkfs.xfs 40, 263
 mkswap 276, 277
 mod_ssl 248
 mount 33, 175, 186, 264, 270
 MTA 253
 MUA 253
 multi-user.target 88
 MX 257

N

NAT 193, 201
 NetBIOS 183
 NetBIOS nameservice 183
 Network Address Translation 193
 Networked Bridge Interface 219
 Network File System 128
 NetworkManager
 Connection Profiles 118
 enabling 114
 installing 114
 permissions 122
 Network News Transfer Protocol 127
 Network Time Protocol 127
 NFS 128
 Cockpit 175
 firewall settings 173
 nfs-utils 174
 NMB 183
 nmcli 113, 221
 activate connection 117
 add bridge 221

Index

- add connections 119
- command line options 114
- deactivate connection 117
- delete connection 119
- device status 115
- general status 115
- interactive 120
- modify connection 118
- permissions 122
- reload 118
- show connections 116
- switch connection 117
- wifi scan 117
- nm-connection-editor 113
 - create bridge 225
- nmtui 113
- NNTP 127
- NTP 127
- nvme 261

O

- OCI 235
- Open Container Initiative 235
- OpenShift 234, 235
- OSI stack 127

P

- Paravirtualization 191, 192
- Partition
 - mounting 40
- passwd 79
- PATH 76
- Path Completion 74
- PE 270
- Physical Extent 270
- Physical Volume 270
- Pipes 75
- podman 235
 - attach 240
 - exec 239
 - list images 238

- pause 241
- ps -a 240
- rm 241
- start 240
- unpause 241
- POP3 127
- Port Forwarding 145, 244
- Ports
 - securing 123
- Postfix 253, 254
 - configuring 255
 - installing 255
 - main.cf 255
 - postmap 259
 - sasl_passwd 258
 - starting 257
 - testing 257
- postmap 259
- Post Office Protocol 127
- poweroff.target 87
- PowerShell 153
- Preboot Execution Environment 10
- private key 149
- ps 75, 281
 - a flag 281
 - aux flags 282
 - H flag 283
 - TERM signal 282
 - u flag 281
- public key 149
- public key encryption 149
- PuTTY 155
 - X11 Forwarding 170
- PuTTYgen 155
- PuTTY Key Generator 155
- PV 270
- pvccreate 273
- pvdisplay 272
- pwd 72
- PXE 10, 208

Q

QEMU 208
 QEMU/KVM Hypervisor 197
 Qmail 254

R

reboot.target 88
 Red Hat, Inc. 7
 Remmina Desktop Client 162
 Remote Desktop Access 159, 160
 enabling 160
 secure vs. insecure 159
 Repositories 95
 disabling 96
 main 95
 managing 96
 multiverse 96
 restricted 95
 sources list 96
 universe 95
 rescue.target 87
 resize2fs 274
 RFC 3339 110
 Richard Stallman 8
 rlogin 126
 root user 1
 rsh 126
 runc 235

S

safe graphics 15
 Samba 173, 180
 add user 182
 firewall settings 180
 installing 180
 NetBIOS 183
 smbclient 184, 186
 smb.conf 181
 smbpasswd 182
 testparm 183
 Samba Client 180

sasl_passwd 258
 SATA 261
 Secure File Transfer Protocol 125
 Secure Shell 126, 149
 Secure Socket Layer 248
 Secure Sockets Layer 128
 Sendmail 253, 254
 Server Message Block 179
 Services
 securing 123
 setfacl 202
 Settings App 83
 users 83
 SFTP 125
 sh 78
 Shell Scripts 77
 Simple Mail Transfer Protocol 126
 Simple Mail Transport Protocol 254
 Simple Network Management Protocol 128
 skopeo 235, 237
 get image info 237
 SMB 179
 smbclient 184, 186
 smb.conf 181
 testing 182
 testparm 182
 smbpasswd 182
 SMTP 123, 126, 254
 SMTP Relay 254, 258
 snap
 channels 108
 commands 106
 disable 111
 enable 111
 find 106
 info 106, 109
 install 108
 list 107
 logs 111
 overview 105
 packages 105

Index

- refresh 109
- refresh.hold 109
- refresh.metered 109
- refresh.retain 109
- refresh.timer 109
- remove 108
- services 111
- set system 110
- start 111
- stop 111
- switch 108
- SNMP 128
- sockets.target 88
- Solaris 7
- sources.list file 96
- spawning 283
- ssh
 - C flag 169
 - X11 Forwarding 168
 - X flag 168
- SSH 123, 126, 149
 - Microsoft Windows 153
 - Multiple Keys 152
- ssh client 151
- ssh-copy-id 151, 154
- sshd_config.d 168
- sshd_config.d directory 152
- sshd_config file 152
- sshd service 152
- ssh-keygen 150
- SSH Service
 - installing 150
 - starting 150
- SSL 128, 248
- SSL certificate 248
- SSL Labs 250
- stderr 75
- stdin 74
- stdout 74
- storage devices
 - identify 11

- Storage Pools 203
- Storage Volumes 203
- su - command 1
- sudo 2
 - wheel group 80
- SunOS 7
- Superuser 1
- swapoff 277
- swapoff u 279
- swapon 276, 280
- swap space
 - add partition 277
 - add swap file 276
 - add to volume group 279
 - current usage 276
 - extend logical swap volume 277
 - recommended 275
- system
 - units 90
 - unit types 90
- systemctl 89
- systemd 87
 - services 87
 - targets 87
- System Monitor 283
- system processes 281

T

- TCP/IP 123
 - Well-Known Ports 125
- Telnet 126
- Terminal window 2
- TERM signal 282
- testparm
 - smb.conf 182
- TFTP 126
- TLS 248
- top 285
 - u flag 286
- Transport Layer Security 248
- Trivial File Transfer Protocol 126

Trusted X11 Forwarding 169
 Type-1 hypervisors 191
 Type-2 hypervisors 191

U

Ubuntu
 history of 8
 meaning 8
 Ubuntu Pro
 enabling 102
 UDP 125
 ufw 129, 135
 command-line options 135
 disabling 135
 enabling 135
 logging 137
 reload 137
 resetting 137
 status 135
 umount 11, 175
 Uncomplicated Firewall 129
 UNIX 7, 71
 origins of 7
 update-manager 96
 Updates 99
 automatic 100
 USB drive
 device name 11
 userdel 79
 usermod 80
 user processes 281
 Users and Groups 79

V

VcXsrv 169
 VG 269
 vgdisplay 270
 vgextend 274
 vgs 271
 Vino 160
 installing 160

virbr0 219, 220
 virsh 215, 217, 222, 229
 destroy 217, 231
 dumxml 217
 edit 224
 help 230
 list 230
 reboot 232
 restore 231
 resume 231
 save 231
 setmem 232
 setmemmax 232
 shell 229
 shutdown 217, 231
 start 217, 231
 suspend 231
 virt-install 199, 215, 223
 virt-manager 196, 207, 224
 installation 196
 New VM wizard 208
 storage pools 210
 VirtualBox 195
 Virtualization 189
 AMD-V 192
 full 192
 guest 189
 hardware 192
 hypercalls 191
 hypervisor 190
 Intel VT 192
 KVM virtualization 193
 MacVTap 193
 Type-1 190
 Type-2 190
 virt-manager 196
 Virtual Machine Networking 193
 virt-viewer 202, 215
 vmdk 209
 VMware 195
 Volume Group 269

Index

W

Wayland 167
WaylandEnable 168
wc 75
Web Server 243
 testing 245
Well-Known Ports 125
wheel group 80
which 72
Whitfield Diffie 149
wildcard character 74
wildcards 74
Windows partition
 filesystem access 32
 reclaiming 37
 unmounting 37
Windows PowerShell 153
wipefs 273

X

X11 Forwarding 167
 compressed 169
X11Forwarding 168
x86 family 191
Xen 196
XFS file system 263
XFS filesystem 274
xfs_growfs 274
XLaunch 169
X.org 167
X Window System 167

Z

ZFS filesystem 18