

Practical Linux with Raspberry Pi OS



Quick Start

—

Ashwin Pajankar

Practical Linux with Raspberry Pi OS

Quick Start

Ashwin Pajankar

Apress®

Practical Linux with Raspberry Pi OS: Quick Start

Ashwin Pajankar
Nashik, Maharashtra, India

ISBN-13 (pbk): 978-1-4842-6509-3
<https://doi.org/10.1007/978-1-4842-6510-9>

ISBN-13 (electronic): 978-1-4842-6510-9

Copyright © 2021 by Ashwin Pajankar

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 NY Plaza, New York, NY 10014. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

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

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-6509-3. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*I dedicate this book to Subrahmanyan Chandrasekhar,
a great Indian-American astrophysicist.*

Table of Contents

- About the Authorxi**
- About the Technical Reviewerxiii**
- Acknowledgmentsxv**
- Introductionxvii**

- Chapter 1: Introduction to Raspberry Pi.....1**
 - Single-Board Computers.....2
 - Raspberry Pi.....2
 - Linux and Distributions5
 - Raspberry Pi OS.....6
 - Raspberry Pi OS Setup.....6
 - Preparing the SD Card Manually.....14
 - Booting Up the Pi Board for the First Time17
 - Configuring the RPi Board.....18
 - Connecting Various RPi Board Models to the Internet29
 - Summary.....34

- Chapter 2: Getting Ready.....35**
 - Operating System Shell.....36
 - Raspberry Pi OS GUI.....36
 - The Command Prompt38
 - Updating the RPi OS39

TABLE OF CONTENTS

- Linux Filesystem 39
- Remotely Accessing the RPi..... 43
- Summary..... 51
- Chapter 3: Directory Commands and Text Editors 53**
- Absolute and Relative Paths 54
- Commands: pwd, tree, and cd..... 54
- Command: ls 57
- Command: touch..... 60
- Various Text Editors..... 61
- Create and Delete Directories 62
- Case-Sensitive Names of Directories and Files 64
- Summary..... 65
- Chapter 4: More Commands 67**
- Configuring the RPi Board..... 68
- What Is sudo? 69
- Getting Help on Commands 69
- Network-Related Commands 70
- Commands: File Operations 73
- Printing a String 75
- Control Operators..... 75
- Filename Globbing 77
- Command: History..... 78
- Pipes 78
- Summary..... 79

Chapter 5: Useful Unix Commands and Tools	81
Shell and Environment Variables	81
Useful Linux Commands	83
Useful Unix Tools	86
Summary.....	89
Chapter 6: Shell Scripting.....	91
Unix File Permissions.....	92
Command: nohup	94
Beginning Shell Scripting.....	95
User Input.....	96
Expressions in the Shell.....	97
If Statement	99
Switch Case	101
Length of a Shell Variable	102
Command-Line Arguments	102
Function	103
Loops in the Shell	104
Comparing Strings	106
File Operations.....	108
Summary.....	110
Chapter 7: I/O Redirection and Cron.....	111
I/O Redirection	111
stdin.....	112
stdout	112
Stderr.....	113

TABLE OF CONTENTS

Crontab 114

Summary..... 116

Chapter 8: Introduction to High-Level Programming Languages117

C and C++ Programming 118

Python Programming Language..... 120

 History of the Python Programming Language..... 120

 Python Enhancement Proposals 121

 Applications of Python..... 122

Python 3 on Debian Derivatives 123

Python Modes 123

 Interactive Mode..... 127

 Script Mode 128

Summary..... 131

Chapter 9: Programming with RPi GPIO 133

General-Purpose Input/Output Pins 133

Programming with GPIO..... 138

Summary..... 141

Chapter 10: Explore the RPi OS GUI 143

GUI Utilities on the RPi OS..... 143

Other Desktop Environments 145

 XFCE 145

 KDE Plasma 148

Summary..... 150

Appendix: Additional Tools 151

- Raspberry Pi Imager 151
- Additional Utilities 152
- Manjaro Linux 153
- FreeBSD 153
- Additional OSs..... 153

Index..... 155

About the Author

Ashwin Pajankar holds a Master of Technology from IIIT Hyderabad. He started programming and tinkering with electronics at the tender age of 7. BASIC (Beginners' All-Purpose Symbolic Instruction Code) was the first programming language he worked with. He was gradually exposed to C programming, 8085, and x86 assembly programming during his higher secondary schooling. He is proficient in x86 assembly, C, C++, Java, Python, and Linux shell programming. He is also proficient with Raspberry Pi, Arduino, BBC Micro Bit, and other single-board computers (SBCs) and microcontrollers. Ashwin is passionate about training and mentoring. He has trained more than 100,000 trainees and professionals through live training sessions and online training courses. He has published more than a dozen books with both international and Indian publishers. He has also reviewed numerous books and educational video courses. This is his sixth book with Apress, and he is working on more books. He regularly conducts programming boot camps and hands-on training for software companies in Nashik, India.

He is also an avid YouTuber with more than 10,000 subscribers to his channel. You can find him on LinkedIn.

About the Technical Reviewer

Massimo Nardone has more than 22 years of experiences in security, web/mobile development, the cloud, and IT architecture. His true IT passions are security and Android.

He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years.

He holds a Master of Science in Computing Science from the University of Salerno, Italy.

He has worked as a project manager, software engineer, research engineer, chief security architect, information security manager, PCI/SCADA auditor, and senior lead IT security/cloud/SCADA architect for many years.

Acknowledgments

I am thankful to Mr. Massimo Nardone, the technical reviewer of the book. I want to express my gratitude toward him for helping me make this book better. I would also like to express my gratitude to the Apress team. Aaron Black helped with the coordination of the entire book process, and Jessica Vakili guided the editorial process. James Markham helped me with the editorial review. I am also thankful to Celestin Suresh and Aditee Mirashi for giving me an opportunity to write this book.

Introduction

I have been working with Raspberry Pi since 2014. I was introduced to it while volunteering as an organizer at a hackathon at my former workplace. And as I have mentioned in my author biography, I have decent experience with assembly programming, microcontrollers, and digital electronics since my time as an undergraduate student of computer science. Raspberry Pi and Arduino are great choices for someone who wants the best of both of the worlds of computer science and electronics. Additionally, if you wish to use Linux, C, Python, and shell programming to drive your motors, LEDs, and other peripherals, then Raspberry Pi and similar single-board computers are the best fit for your applications.

While this is an introductory book for beginners who are new to the world of SBCs and hardware hacking, once you follow the book in detail, you will be very much comfortable exploring the world of SBCs and Raspberry Pi on your own.

This entire book is based on the challenges and struggles I faced while working the very first time with Raspberry Pi (abbreviated as RPi hereafter). I have listed all the tips and tricks I learned while exploring the RPi in the first few months. At that time, I really wished that I could get a book that would make my journey easier and so decided to write one myself. So, when the opportunity presented itself, I compiled all my experiences in this book so that anyone looking for help to get started with Raspberry Pi can benefit from this.

INTRODUCTION

While this book primarily covers the RPi OS flavor of Linux, I have made sure that I also introduce the readers to other important topics, such as programming with high-level languages like C, C++, and Python. I also introduce readers to GPIO (General-Purpose Input/Output) programming and various buses. Finally, I cover installation of a few popular Linux desktop environments such as XFCE, LXDE, and KDE Plasma. The appendix covers a few additional tips and tricks.

I hope that the book serves the readers well and they will enjoy the book as much as I enjoyed writing it.

CHAPTER 1

Introduction to Raspberry Pi

I hope you have gone through the Table of Contents and Introduction. If not, I highly recommend you go through them. This is the very first chapter of the book, and I welcome you all to the exciting journey of learning Linux with the Raspberry Pi Operating System.

In this chapter, we will learn the details about the most popular platform and single-board computer family of our times, the Raspberry Pi. Then we will learn a bit about Linux and the distribution of Linux that is popularly used with the Raspberry Pi family (hereafter, I will use the abbreviation RPi), the Raspberry Pi Operating System. We will learn how to install it on a RPi board. The following is the list of the topics that we will learn in this chapter:

- Single-board computers
- Raspberry Pi
- Linux and distributions
- Raspberry Pi OS setup
- Configuring the RPi board
- Connecting various RPi board models to the Internet

After completing this chapter, we will be comfortable with the installation and the basic usage of the RPi board and the RPi OS.

Single-Board Computers

Single-board computers (also known as SBCs) have all the components of a fully functioning computer like the processor, GPU, RAM, and I/O on a single printed circuit board. This is in contrast with desktop or laptop computers that have a motherboard which has various slots for RAM, the processor, and the graphics card. Desktop or laptop computers can be upgraded by replacing processors and graphics cards. We can also add more RAM chips in the RAM slots. However, SBCs cannot be upgraded like that. This is one of the major differences between traditional desktops/laptops that are totally modular and SBCs. The key benefit of the lack of modularity of SBCs is that the size of an entire computer is very small. Most of the SBCs are a little bigger than a regular credit/debit card, and they are very compact.

SBCs are used as technology demonstrators (prototypes), educational computers, and embedded systems. There is a recent surge in the popularity of SBCs due to advances in the fabrication process and manufacturing technologies. We are living in an era where a new SBC or a new version of an existing one is announced almost on a monthly basis. The market is full of various SBCs and SBC families. A few prominent SBC families are Raspberry Pi, Banana Pro, BeagleBoards, and Orange Pi. Raspberry Pi is the most popular family of single-board computers available in the market, and it is one of the best-selling computers in the world. In the next section, we will have an overview of the Raspberry Pi family of computers.

Raspberry Pi

Raspberry Pi is a family of SBCs developed by the Raspberry Pi Foundation (www.raspberrypi.org/). It consists of many board models, and all the current models under production are listed on the foundation's products

page (www.raspberrypi.org/products/). Throughout the book, I will be using a Raspberry Pi 4 Model B (the latest board model in the family) with 4 GB RAM.

Table 1-1 lists the specifications of the Raspberry Pi 4 Model B.

Table 1-1. *Technical Specifications of Raspberry Pi 4 B*

Component	Specification
Processor	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC at 1.5 GHz
RAM	LPDDR4-3200 SDRAM (2 GB or 4 GB or 8 GB)
Networking	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE Gigabit Ethernet
USB	2 USB 3.0 ports, 2 USB 2.0 ports
General-Purpose Input/Output	Raspberry Pi standard 40-pin GPIO header
Display	2 micro-HDMI ports (up to 4kp60 supported) Two-lane MIPI Display Serial Interface port (www.mipi.org/specifications/dsi)
Camera connector	Two-lane MIPI Camera Serial Interface port
Audio	Four-pole stereo audio and composite video port
Secondary storage	MicroSD card slot for OS and data storage
Power	5 V DC via USB-C connector or 5 V DC via GPIO header (minimum 3A)

There are many models of the boards in this family that are currently under production, and if you visit the hobby electronics store near your home, you may find older out-of-production board models at a bargain price. To keep it brief, I will be discussing the technicalities and specifications of the other models only when needed. We can purchase Raspberry Pi boards at authorized retailers (the list can be found in the products page) or at popular ecommerce websites such as Amazon.

Figure 1-1 shows a RPi 4 B board.



Figure 1-1. Photograph of a RPi 4 B board

Figure 1-2 shows schematics of the components of RPi 4 B.

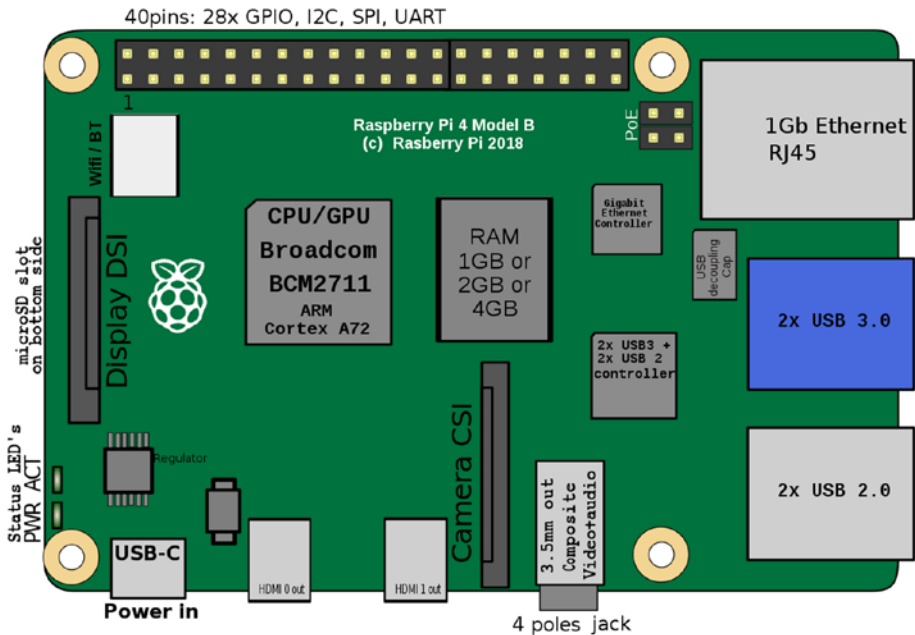


Figure 1-2. *Components on a RPi 4 B board*

Throughout this book, I will be explaining all the demonstrations using a 4 GB RAM model of this board.

Linux and Distributions

Linux is a family of open source Unix-like free operating systems. It is based on the Linux kernel, a free and open source operating system kernel by Linus Torvalds. The Linux OS is packaged or in a Linux distribution. A Linux distribution includes the Linux kernel developed and supporting system software, libraries, and APIs for programmers. Many of these components are part of the GNU Project (www.gnu.org/home.en.html), and that is why many people refer to Linux as GNU/Linux. As Linux is free and open source, anyone can create a custom distribution of Linux.

The following URLs have more information about GNU and Linux projects:

www.linux.org/

www.gnu.org/

www.fsf.org/

The top 25 distributions of Linux can be found here:

www.linux.org/pages/download/

Raspberry Pi OS

The Raspberry Pi Operating System is a derivative of a popular Linux distribution known as Debian. It is officially provided by the Raspberry Pi Foundation, and it is the most recommended operating system for the RPi family of SBCs. It is fully optimized for the RPi board models, and all the board models are supported by it. Formerly, it was known as the Raspbian OS, and it was created by Peter Green and Mike Thompson. In this chapter, we will learn in detail how to install the RPi OS on a microSD card and how to boot up a Pi board with that microSD card. The DistroWatch page about the OS can be found here: <https://distrowatch.com/table.php?distribution=raspios>.

Raspberry Pi OS Setup

As we have seen already, the Raspberry Pi OS is the most preferred OS for the Pi boards. In this section, we will learn how to set up the RPi OS on RPi boards. Though I will be using a RPi 4 B 4 GB model for all the demonstrations, for the convenience of the readers, we will discuss the setup process for all the board models ever produced by the foundation except the compute modules. Let us see all the components needed for the setup one by one:

- 1) We need a RPi board of any model.
- 2) We need an appropriate power supply. For Raspberry Pi 4 B, we need a USB-C power supply. Figure 1-3 is an image of a USB-C male pin.

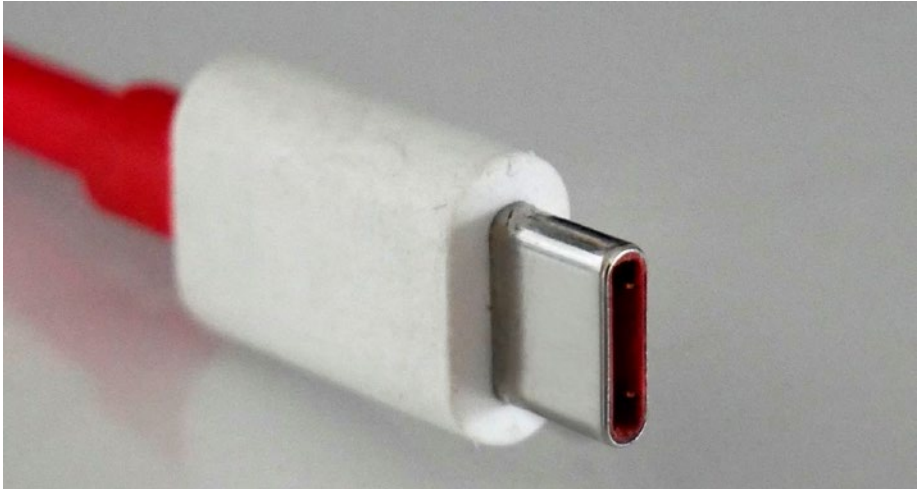


Figure 1-3. *USB-C male header*

The Raspberry Pi Foundation has an official 15.3 W power supply for RPi 4 B. We can find more information at www.raspberrypi.org/products/type-c-power-supply/.

All other models of RPi boards need to be supplied by a micro-USB power supply. Figure 1-4 is an image of a micro-USB male pin.



Figure 1-4. *Micro-USB male pin*

The Raspberry Pi Foundation has an official universal power supply. We can find more information at www.raspberrypi.org/products/raspberry-pi-universal-power-supply/.

- 3) We also need a pair of a USB mouse and a USB keyboard. A USB keyboard and mouse combo that uses a single USB port is preferred. It is available in the form of a keyboard with a built-in mousepad as shown in Figure 1-5.



Figure 1-5. A USB keyboard with a built-in mousepad

The board models RPi Zero and RPi Zero W have only a single micro-USB port, so this is mandatory for such models if we want to use them with a keyboard and a mouse. Also, for RPi Zero and RPi Zero W, we need a USB to micro-USB OTG converter as shown in Figure 1-6.



Figure 1-6. A USB OTG converter

- 4) The RPi board models use a microSD card to store OS and data. RPi 1 Model A and RPi 1 Model B use a SD card, and the rest of the models use a microSD card. We can get more information about the SD cards and compatibility at www.raspberrypi.org/documentation/installation/sd-cards.md and https://elinux.org/RPi_SD_cards. I recommend to purchase a class 10 card of 16 GB size. Also purchase a microSD to SD card converter if you are using RPi 1 Model A and RPi 1 Model B. Figure 1-7 is an image of a microSD card with a microSD to SD card converter.



Figure 1-7. A microSD card and a microSD to SD card converter

- 5) We need a HDMI or a VGA monitor for display.
- 6) All the models of RPi boards except the models RPi 4 B, RPi Zero, and RPi Zero W have HDMI output and can be directly connected to a HDMI monitor with a HDMI male-to-male cable. A HDMI male pin is shown in Figure 1-8.



Figure 1-8. *A HDMI male head*

RPi 4 B has micro-HDMI output. So we need a micro-HDMI to HDMI converter. RPi Zero and RPi Zero W have got mini-HDMI output. So, for them, we need a mini-HDMI to HDMI converter. Figure 1-9 shows us the HDMI, mini-HDMI, and micro-HDMI male pins, respectively.

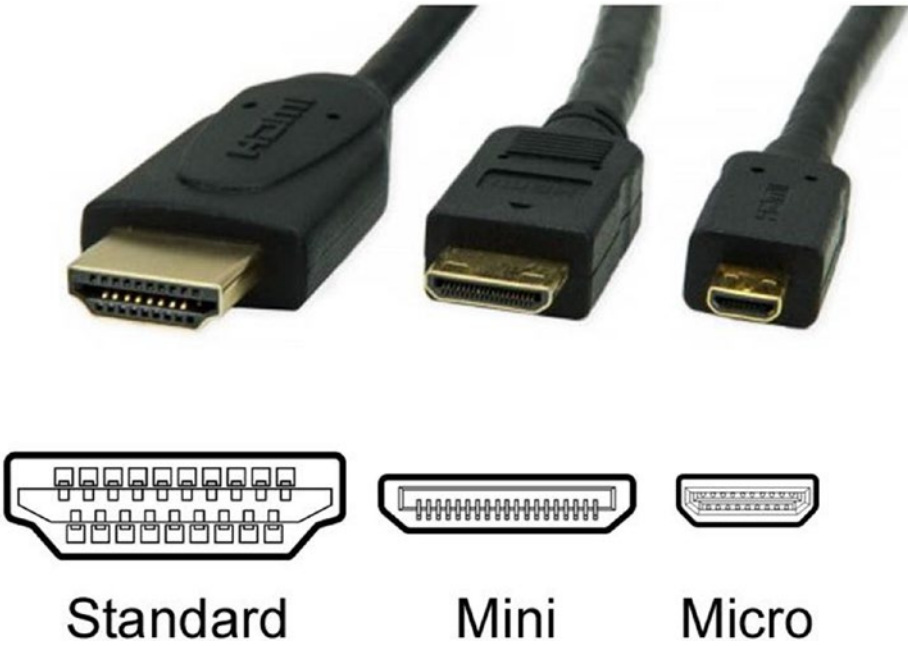


Figure 1-9. Various types of HDMI male pins

Figure 1-10 is an image of a HDMI to VGA converter, a mini-HDMI to HDMI converter, and a micro-HDMI to HDMI converter.



Figure 1-10. Various types of HDMI converters

If we are using a VGA monitor, then we need to use the HDMI to VGA converter shown in the preceding image.

- 7) We need a SD card reader. Many laptops have a built-in SD card reader. If your laptop does not have one, you need a separate card reader. Figure 1-11 is a representational image of a SD card reader.



Figure 1-11. SD card reader

- 8) Finally, we need a computer with the Windows, Linux, or macOS operating system.

Preparing the SD Card Manually

Preparing the SD card manually is recommended as it gives us the full control, and we can change the settings before the very first booting of the board. For that, the Raspberry Pi Foundation provides us a utility known as the **Raspberry Pi Imager**. It can be downloaded from www.raspberrypi.org/downloads/. It is available for Windows, Linux, and macOS. Download it and install it on your OS. Once you open it, it shows the window in Figure 1-12.



Figure 1-12. *Raspberry Pi Imager window*

As we can see in Figure 1-12, we have an option for choosing a SD card. Insert the microSD card you have into the card reader device or the card reader slot of your laptop. The Imager will detect it. Then click the **CHOOSE OS** button. It will open the window in Figure 1-13.

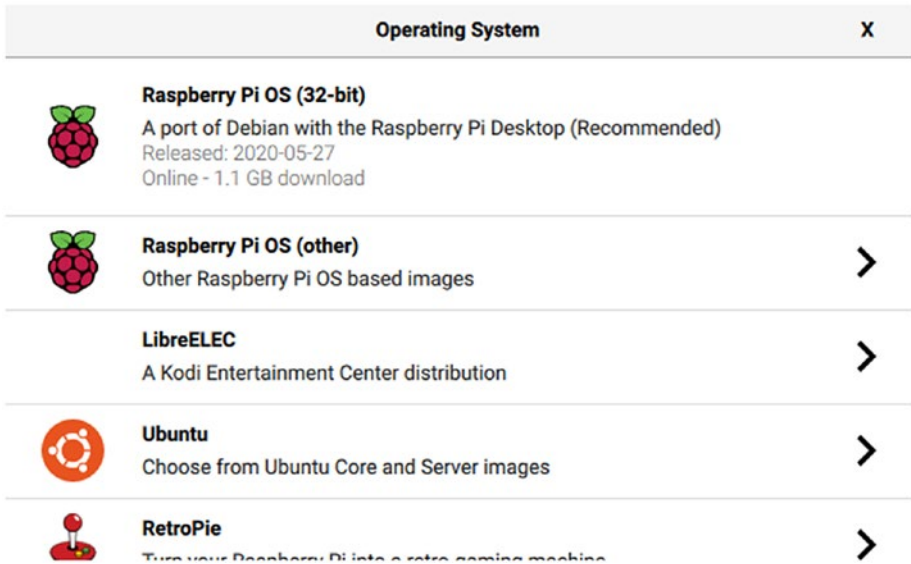


Figure 1-13. Option for choosing the OS

We can see the options for various operating systems. We need to choose the second option, **Raspberry Pi OS (other)**. Then it shows the options displayed in Figure 1-14.

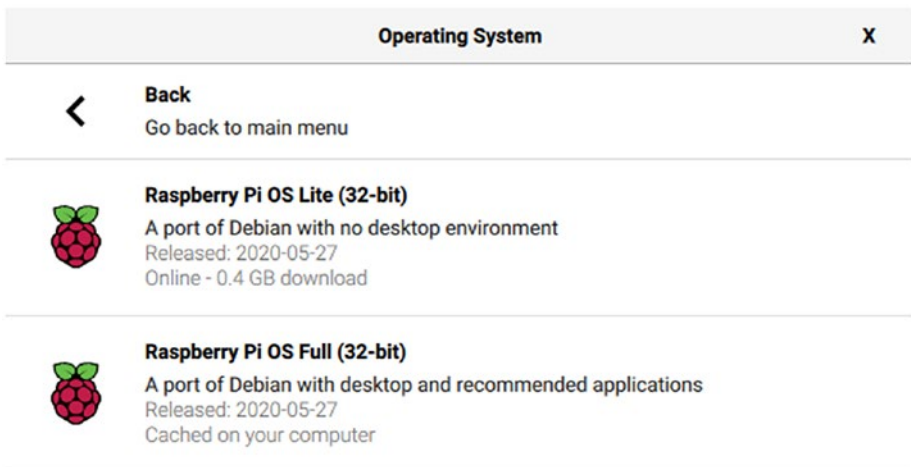


Figure 1-14. More choices under Raspberry Pi OS (other)

Choose **Raspberry Pi OS Full (32-bit)**. Then click the button labeled as **WRITE**. It will start writing the OS to the microSD card as shown in Figure 1-15.



Figure 1-15. Writing the OS to the microSD card

Once the OS is written to the microSD card, safely remove the card from the SD card reader and reinsert it. In the Windows OS, it will show as a drive with the label **boot**. In this drive, there is a file labeled as **config.txt**. This file stores all the options related to booting, and it acts in the same way as the BIOS (Basic Input/Output System) to initialize the booting. In case we are using a HDMI monitor, we do not have to modify the settings. But if we are using a VGA monitor, we need to make changes to a few lines as follows:

- Change `#disable_overscan=1` to `disable_overscan=1`.
- Change `#hdmi_force_hotplug=1` to `hdmi_force_hotplug=1`.
- Change `#hdmi_group=1` to `hdmi_group=2`.

- Change `#hdmi_mode=1` to `hdmi_mode=16`.
- Change `#hdmi_drive=2` to `hdmi_drive=2`.
- Change `#config_hdmi_boost=4` to `config_hdmi_boost=4`.
- Save the file.

By default, the commented options (which have the symbol `#` at the beginning) are disabled. We must enable these options by uncommenting their respective lines by removing the symbol `#` at the beginning of these commented lines.

Booting Up the Pi Board for the First Time

Let us boot up our RPi board for the first time with the microSD card we prepared. The following are the steps:

- Insert the microSD card into the microSD card slot of the RPi board. RPi 1 Model A and RPi 1 Model B have slots for a SD card. So, for these board models, we must use a microSD to SD card converter. Insert the microSD card into the microSD to SD card converter and then insert the converter into the RPi 1 Model A and RPi 1 Model B SD card slot.
- Connect the Pi to the HDMI monitor. As discussed earlier, in case you have a VGA monitor, connect it using the HDMI to VGA converter.
- Connect the USB mouse and USB keyboard. It is recommended to have a single keyboard with a mousepad. For RPi Zero or RPi Zero W, you need to first connect it to a USB OTG cable and then connect the USB OTG cable to the RPi Zero or RPi Zero W board.

- Connect the RPi board to an appropriate power supply (we have discussed this earlier). Connect the monitor to a power source too. Make sure that the power to the RPi board and the monitor is switched off at this point.
- Check all the connections once and then switch on the power supply of the RPi and the monitor.

At this stage, our RPi board will start booting up. We will see a green light blinking on the Pi board. It means that it is booting up. Well, congratulations on our very first success!

Note If the HDMI monitor is showing the message **No Signal** and not showing any visual output, then power down the RPi board and change the line `#hdmi_force_hotplug=1` to `hdmi_force_hotplug=1` in the file `/boot/config.txt` on the microSD card. Boot up the RPi again with this changed setting, and the HDMI monitor will definitely show the output.

Configuring the RPi Board

Once the RPi boots for the first time, it will show a configuration wizard window as shown in [Figure 1-16](#).



Figure 1-16. *Welcome window*

Click the button Next, and the window in Figure 1-17 will appear.



Figure 1-17. *Country and Language*

In the preceding image, set the **Country** and the **Language**; it will automatically select the **Timezone** according to the **Country** selected. We can change that too if we wish. Click the **Next** button, and the window in Figure 1-18 will appear.

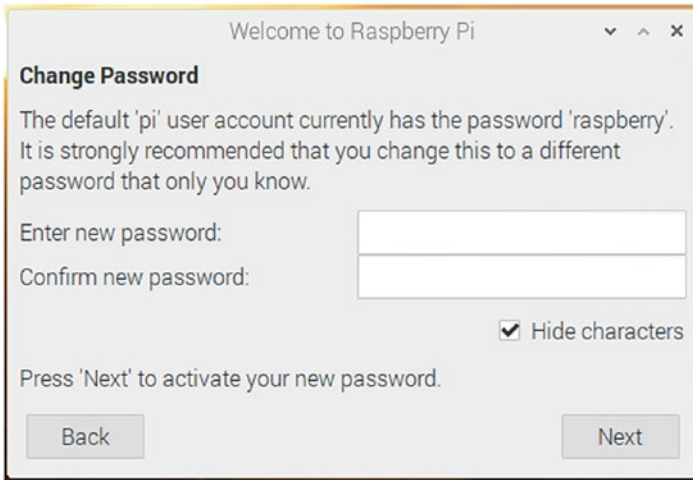


Figure 1-18. *Change the password*

Here, we can change the default password of the user **pi** if we want. If we leave it blank, then it will retain the default password. Click the **Next** button, and it will show the window in Figure 1-19.

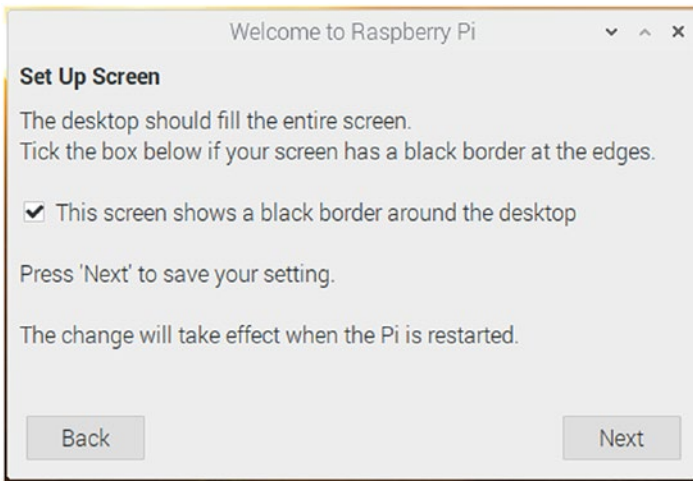


Figure 1-19. *Set Up Screen window*

Check the checkbox if black borders are visible at the edges of the desktop view. The Raspberry Pi OS will correct it on the next boot. The window in Figure 1-20 will appear after we click the **Next** button only if the RPi board model has WiFi.

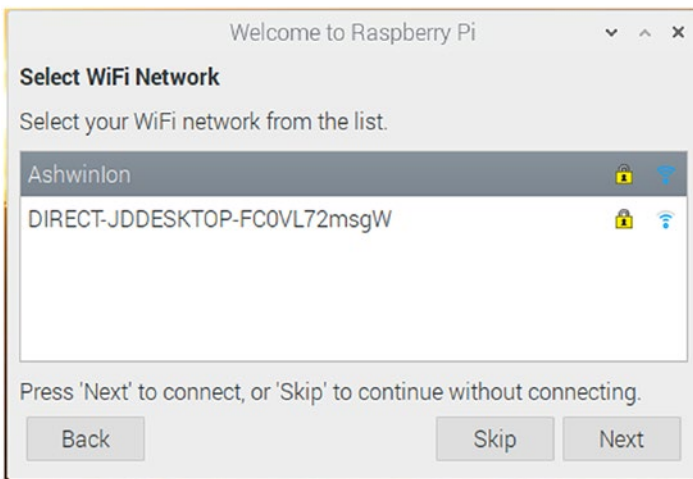


Figure 1-20. *WiFi network selection window*

Choose the WiFi network for which you know the credentials and click the **Next** button, and the window in Figure 1-21 will appear.

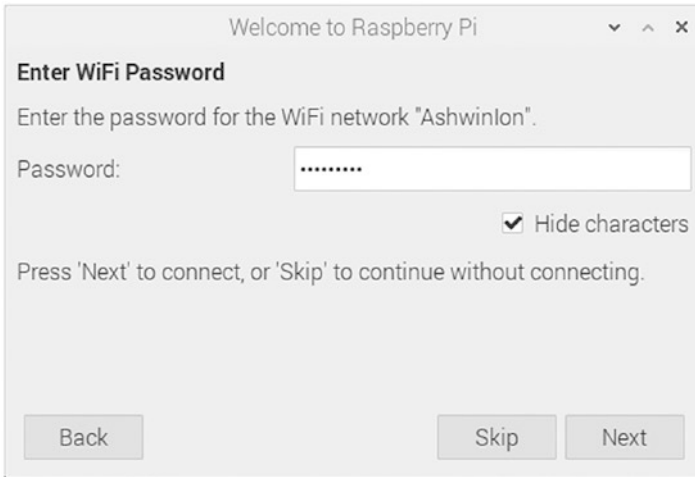


Figure 1-21. *WiFi network password*

Enter the password of the selected WiFi network and click the **Next** button. It will show Update Software window as shown in Figure 1-22.

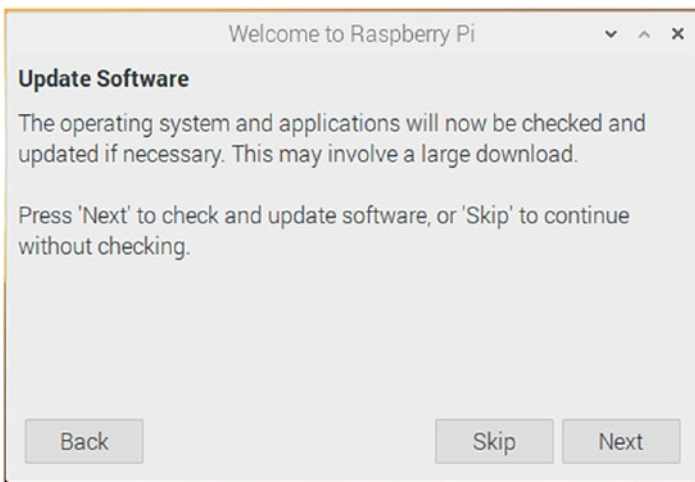


Figure 1-22. *Update Software window*

If we click the **Next** button, then it will update the RPi OS. We will learn how to do it manually in the next chapter. For now, we will skip this by clicking the **Skip** button. It will then show the window in Figure 1-23.

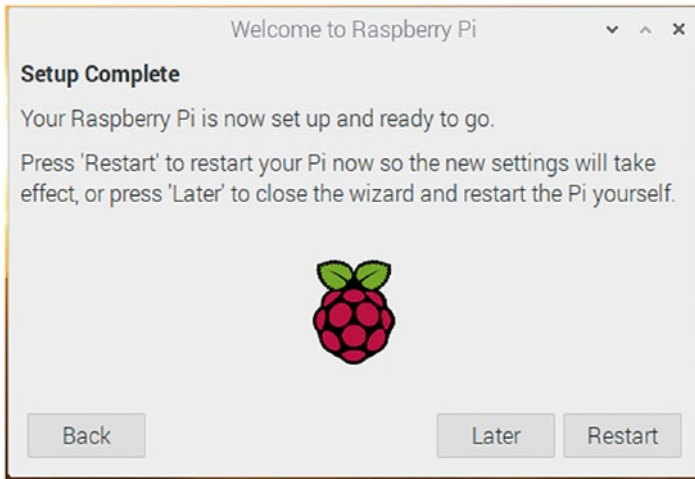


Figure 1-23. Setup Complete

This means that the configuration is successful. We need to configure a few more settings manually before a reboot, so click the **Later** button.

In the top-left corner of the desktop, we see a Raspberry icon. It is the menu for the Raspberry Pi OS, and it functions in the same way as the Windows logo in the Microsoft Windows OS. Click that Raspberry logo and navigate to **Preferences** ► **Raspberry Pi Configuration** as shown in Figure 1-24.

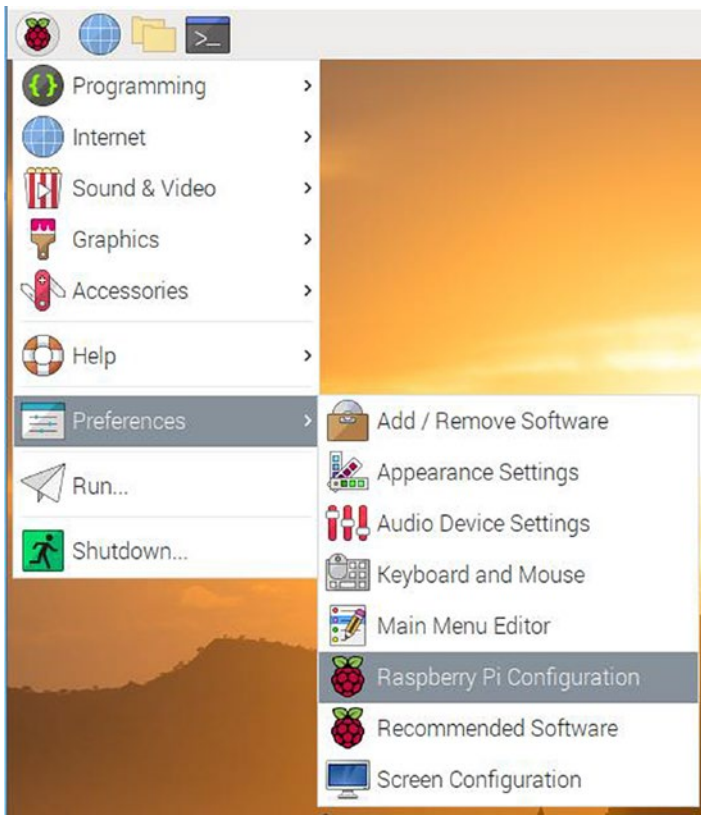


Figure 1-24. *Raspberry Pi Configuration in the RPi OS menu*

Clicking the option shown in the preceding image will open the configuration window shown in Figure 1-25.

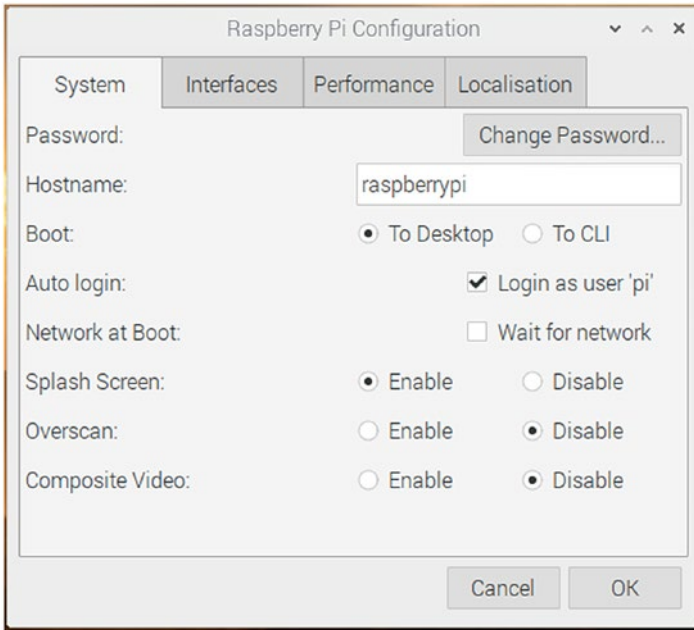


Figure 1-25. *Raspberry Pi Configuration window*

Click the tab Interfaces and it will show options for interfaces as shown in Figure 1-26.

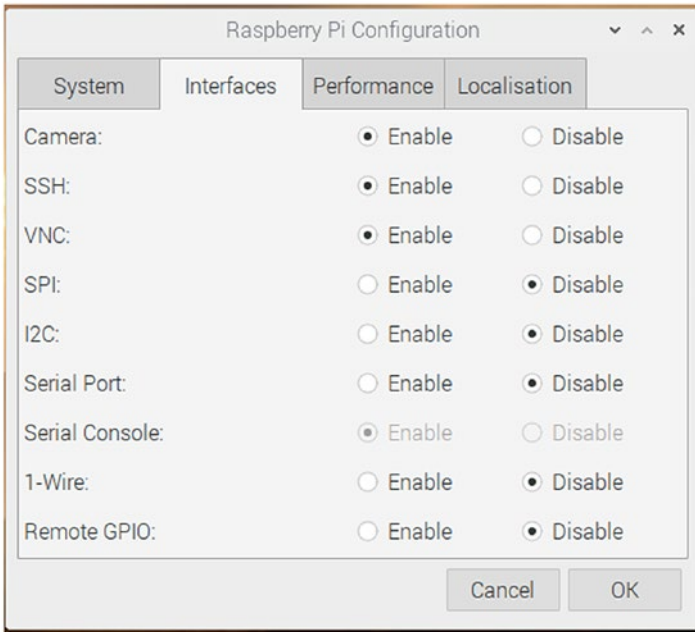


Figure 1-26. Raspberry Pi Interfaces window

In this window, click the **Enable** radio buttons for the options **Camera**, **SSH**, and **VNC**. Then click the **Performance** tab and it will show performance options as shown in Figure 1-27.

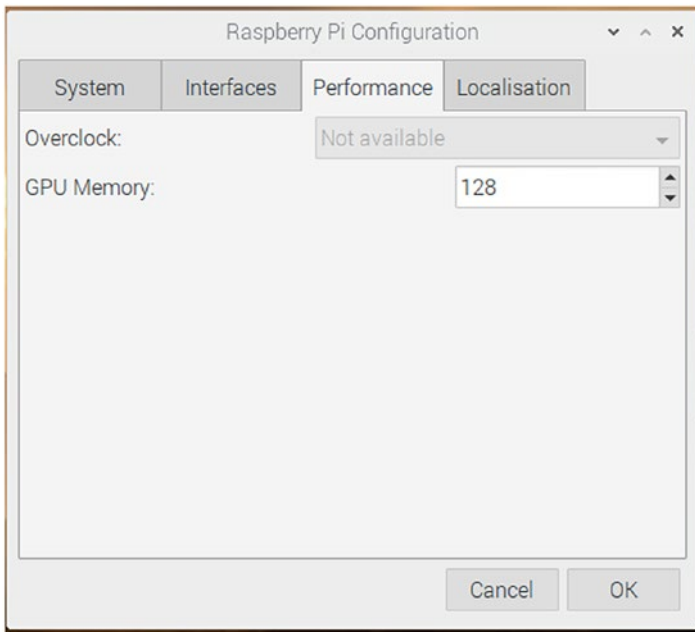


Figure 1-27. *Raspberry Pi Performance*

The option to overclock through this utility is disabled for many board models. Here, we can set the GPU memory. I recommend setting it to 128 MB. This much amount of RAM is used by GPU as video memory (RPi does not have a dedicated GPU memory). Finally, click the **Localisation** tab and it will show Localization options as shown in [Figure 1-28](#).

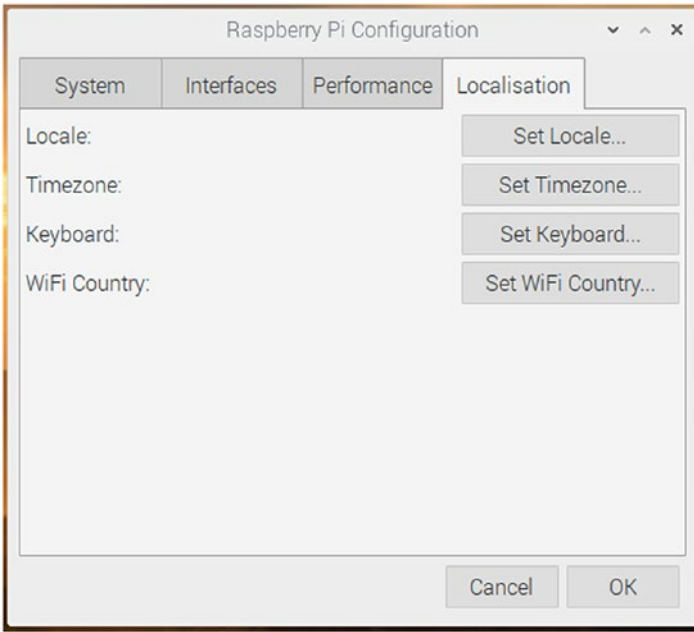


Figure 1-28. *Raspberry Pi Localisation*

Here, we can set the options as per our localization requirements.

Once all these settings are changed as per our choice, we can reboot the RPi board by clicking the last button labeled as Shutdown in the RPi OS. It opens the window in Figure 1-29.

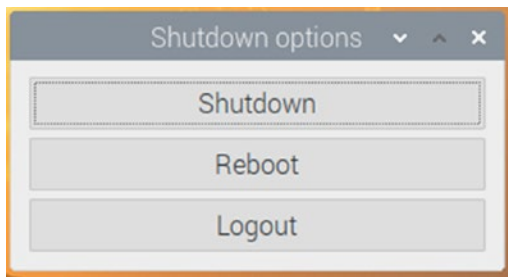


Figure 1-29. *Raspberry Pi Shutdown options*

Just click the **Reboot** button, and the RPi will reboot. All our changes will take effect after the reboot is completed. If we have not changed the default password for the user **pi**, then at startup, the message in Figure 1-30 will be shown.

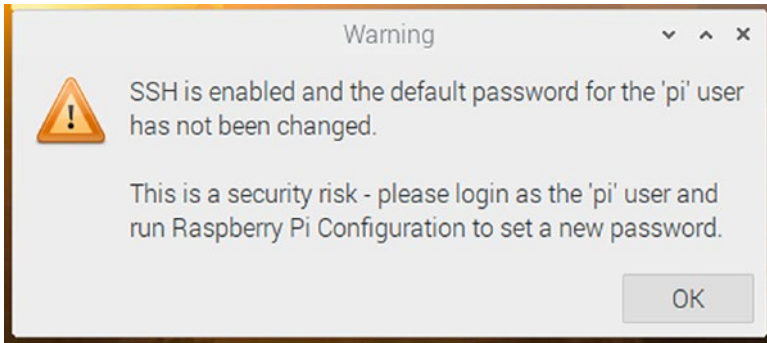


Figure 1-30. Raspberry Pi warning message after booting to the desktop

Connecting Various RPi Board Models to the Internet

We can directly plug in the Ethernet cable to the RJ45 Ethernet port on the RPi boards that have it. It will automatically detect the connection and connect to the Internet. Just make sure that DHCP (Dynamic Host Configuration Protocol) is enabled at the WiFi router or the managed switch or the Internet gateway. The models RPi 1 A, RPi 1 A+, RPi Zero, RPi Zero W, and RPi 3 A+ do not have an Ethernet port. However, RPi Zero W and RPi 3 A+ have built-in WiFi for connecting to WANs. We can use a simple USB WiFi dongle for the rest of the models. Figure 1-31 is an image of a USB WiFi dongle.



Figure 1-31. A USB WiFi dongle

Plug in this USB WiFi adapter to one of the USB ports. If the USB ports are not enough, then use a powered USB hub. For Raspberry Pi Zero, we need to plug in this dongle to a USB OTG cable and then plug that into the micro-USB port of RPi Zero.

After plugging in the USB WiFi adapter, we need to open the **lterminal** utility. It is the terminal command-line utility of the RPi OS. We can find it as a small black icon in the RPi OS's taskbar, and we can also find it in **Accessories** in the RPi OS menu. Another way to invoke it is to press Ctrl+F2. The **Run** window in Figure 1-32 will appear. Here, you can type in **lterminal** and then press the Enter key on the keyboard or click the **OK** button.

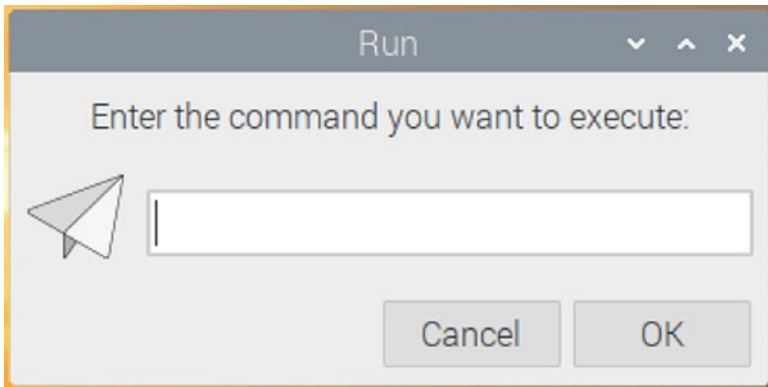


Figure 1-32. A Run window

The `lxterminal` is the terminal emulator for the Raspberry Pi OS, and Figure 1-33 is a screenshot of an instance of it.

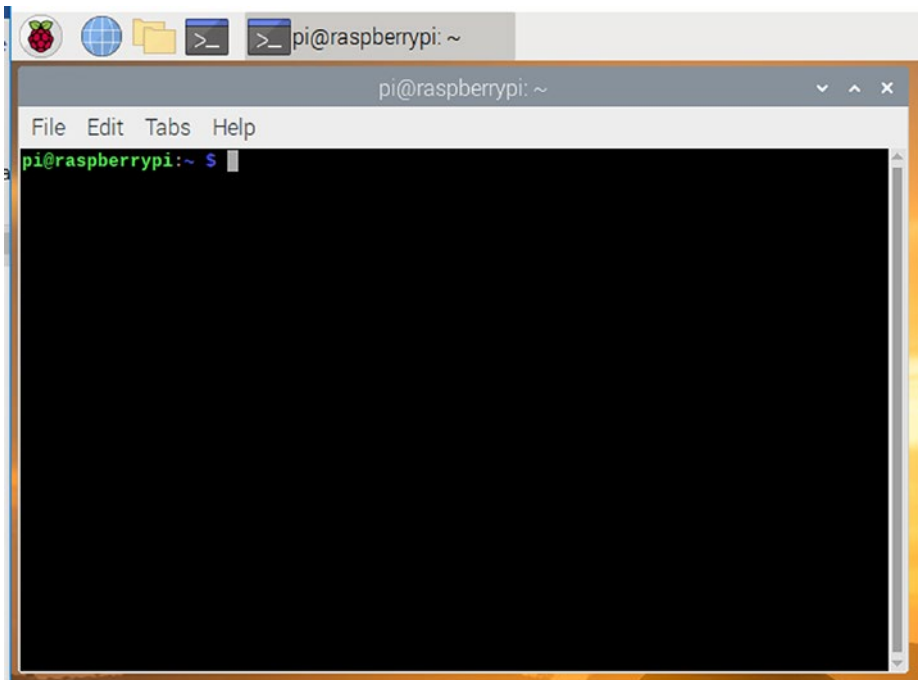


Figure 1-33. A screenshot of the `lxterminal`

Here, we can type in the Linux commands; and after typing in, we must press the Enter key to execute the current command. Let us manually configure the networking using this. This will also give you a decent practice to work with the lxterminal which we will be primarily using throughout the book.

All the network-related information is stored in a file at `/etc/network/interfaces`. Do not bother yourself too much at this stage. We will learn all these things in detail from the next chapter onward. To connect to WiFi after plugging in the USB WiFi dongle, we need to add a few entries to this file I mentioned. First, take the backup of the original file by executing the following command in the lxterminal:

```
mv /etc/network/interfaces /etc/network/interfaces.bkp
```

Then we can create the network interfaces file from scratch by running the following command:

```
sudo nano /etc/network/interfaces
```

The preceding command will open the network interfaces file with a plaintext editor known as the nano editor. It is a simple **WYSIWYG (What You See Is What You Get)** plaintext editor. Enter the following lines there:

```
source-directory /etc/network/interfaces.d
auto lo
iface lo inet loopback
auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-ssid "AshwinIon"
wpa-psk "internet1"
```

After entering the lines, press **Ctrl+X** and then enter **Y**. In the preceding settings, substitute **AshwinIon** with your own SSID and **internet1** with the password for the same WiFi network. Then run the following command on the command prompt:

```
sudo service networking restart
```

It will restart the networking service and will connect to the WiFi. In any case (Ethernet or WiFi), the RPi is assigned with a unique IP address. We can find it out by running the Linux networking command `ifconfig` at the **lxterminal**. The output of the command will have the IPV4 and the MAC addresses of the RPi board.

The other way of knowing the IP address of the RPi is by checking the active client list in the WiFi router or the managed switch to which the RPi board is connected. Figure 1-34 is a screenshot of my WiFi router's active client list where we can see an entry for the RPi connected to it.

The screenshot shows a web interface with a sidebar on the left containing 'Device Info', 'Active Client Table', 'Statistics', and 'IPv6'. The main content area has tabs for 'Setup', 'Wireless', 'Advanced', 'Maintenance', and 'Status'. Under the 'Wireless' tab, there are three sections: 'Active Client Table' (with a note that it shows IP and MAC addresses), 'Active Wired Client Table' (with a header table), and 'Active Wireless Client Table' (with a data table). A 'Refresh' button is at the bottom right.

Name	IP Address	MAC Address
realme-2-Pro	192.168.2.2	50:29:f5:9d:bf:c1
DESKTOP-FC0VL72	192.168.2.5	d4:6e:0e:11:b2:ea
raspberrypi	192.168.2.6	7c:dd:90:00:e2:1e

Figure 1-34. A screenshot of the active client list

The last entry corresponds to the RPi board connected to it.

Summary

In this chapter, we got started with the basics of Linux and the Raspberry Pi OS. Then we installed the Raspberry Pi OS on a microSD card and learned how to boot up various models of Raspberry Pi. We have also had a bit of hands-on with the terminal emulator **lxterminal**, and we will explore this in detail in the next chapter.

In the next chapter, we will learn the basics of the Linux filesystem and GUI (Graphical User Interface). We will learn what an OS shell is and how to communicate with it using the terminal emulator. We will also learn how to update the RPi OS with commands and how to remotely connect to it.

CHAPTER 2

Getting Ready

In the last chapter, we became familiar with single-board computers and Raspberry Pi. We also learned the basics of Linux and the Raspberry Pi OS. We learned how to prepare a SD card with the RPi OS and how to boot up a RPi board with the RPi OS. We also learned a few basics of working with the terminal emulator.

As a continuation of the last chapter, we will explore the following concepts in this chapter:

- Operating system shell
- Raspberry Pi OS GUI
- The command prompt
- Linux filesystem
- Remotely accessing the RPi

After completing this chapter, we will be very comfortable with the terminal emulator and shell of Linux. We will also be comfortable with the Linux filesystem.

I would like to note one more thing. I am using a RPi 4 B with 4 GB RAM for the demonstrations throughout the book. However, all the demonstrations in this book will work with any RPi board model. And the commands that are not specific to the RPi OS will run on any Debian Linux distribution or derivative. If any command is specific to the RPi OS and not compatible with Debian, then I will mention it in the description.

Operating System Shell

In any operating system, a shell is a user interface for accessing a system's services.

It takes input from the user and executes programs based on that input. When a program finishes execution, the shell displays the program's output.

All the operating systems have shells. An operating system can have multiple shells. A shell can use the Command-Line Interface (CLI) (like the Unix terminal emulator programs) or Graphical User Interface. In this chapter, we will explore both the concepts in detail.

Command-Line Interface (or CLI)-based shells need users to memorize the commands. We will explore a variety of commands throughout this and the remaining chapters of this book. Shell commands can be put together into scripts that can be used to perform a variety of tasks on a Unix-like computer.

Graphical User Interface (or GUI)-based shells are easier to use. Basically these use one of the varieties of the desktop environments for Unix-like operating systems. We can read more about the desktop environments in the article at <https://itsfoss.com/best-linux-desktop-environments/>.

Raspberry Pi OS GUI

Let us have an overview of the RPi OS GUI. We have a dedicated chapter near the end of this book for exploring a few GUI utilities of the Raspberry Pi OS. In this section, we are just going to have a very brief overview of the GUI of the RPi OS. The Raspberry Pi OS uses the PIXEL desktop which is a customized LXDE (**L**ightweight **X**11 **D**esktop **E**nvironment). Other popular desktop environments for Unix-like operating systems are KDE, GNOME, and XFCE. It is possible to use these with the RPi OS, but for the

sake of simplicity, we will stick to the default LXDE for our demonstrations throughout the book.

Let us have an overview of the GUI of the RPi OS. When we boot it up, we can see a desktop like any other Unix-like environment. The desktop has a taskbar where we can see various options (Figure 2-1).



Figure 2-1. *Raspberry Pi OS Desktop*

The Raspberry fruit symbol in the leftmost corner is the RPi OS menu where we can find all the GUI packages for making our life easier. The globe symbol next to that is the shortcut for a web browser. The folders symbol next to that is the shortcut for the **File Explorer** utility. Then the black icon next to it is the shortcut for the **lxterminal** which is the default command-line terminal emulator for the RPi OS. We have already used it briefly for changing the networking settings.

On the right-hand side, we can see the VNC server symbol. This is because I am accessing the desktop remotely using a Windows computer. We will learn about it in detail in the later part of this chapter. After that, we see a Bluetooth symbol. We can connect to a Bluetooth device of our choice using this. The next is the WiFi symbol. We can connect to a WiFi network of our choice. After that, we can see the audio meter for adjusting the sound and a clock. On the desktop, the only icon is the Trash, where you can find the recently deleted items, and they can be either recovered or removed permanently from here. If you ever have worked with any GUI-based operating system, you will find all this very familiar.

The Command Prompt

We can access the command prompt by using the terminal emulator, **lterminal**. Open the lterminal window and execute the following command:

```
echo $SHELL
```

It will return the following output:

```
/bin/bash
```

This is the default shell of the RPi OS. It is known as the **Bash** shell. The output is the location of the executable file for the program of the Bash shell. The RPi OS has other shells. We can see them with the following command:

```
ls -la /bin/*sh*
```

It will return the following output:

```
pi@raspberrypi:~ $ ls -la /bin/*sh*
-rwxr-xr-x 1 root root 925124 Apr 18 2019 /bin/bash
-rwxr-xr-x 1 root root 91896 Jan 18 2019 /bin/dash
lrwxrwxrwx 1 root root 4 Apr 18 2019 /bin/rbash -> bash
lrwxrwxrwx 1 root root 4 May 27 12:35 /bin/sh -> dash
```

There are four lines in the output. And as we can see, there are four shells, namely, bash, dash, rbash, and sh. rbash and sh are nothing but the symbolic links (represented by the -> symbol in the earlier output) to the bash and dash shells. So the RPi OS has two shells, and Bash is the default shell.

Do not worry too much about these commands as of now. We will learn them in detail in the upcoming chapters.

Updating the RPi OS

In the last chapter, we learned how to get the RPi board up and running with the RPi OS. We had skipped the process to update the RPi OS at the first-time configuration of the RPi. Let us update it from the command prompt now. All the latest packages are updated in the RPi OS repository, and we can update the RPi OS by referring to this repository using the Internet. The following command is used to download package information from all configured sources:

```
sudo apt-get update -y
```

This updates the information on the updated versions of packages or their dependencies. After this, we have to run the following command:

```
sudo apt-get dist-upgrade -y
```

This command upgrades all the packages and their dependencies and also removes all the obsolete packages. The parameter `-y` in both the commands means we are entering `y` whenever the execution prompts for Yes/No.

Finally, update the firmware with the following command:

```
sudo rpi-update
```

This is how we update the RPi OS and the firmware on the RPi board.

Linux Filesystem

In this section, we will briefly discuss the Linux filesystem. The Linux filesystem is modeled after the Unix filesystem. We can explore the filesystem using the File Explorer. When we open the File Explorer, it opens the visual view of the folder `/home/pi`. Folders are also known as directories. Figure 2-2 is a screenshot of the File Explorer showing the `/home/pi`.

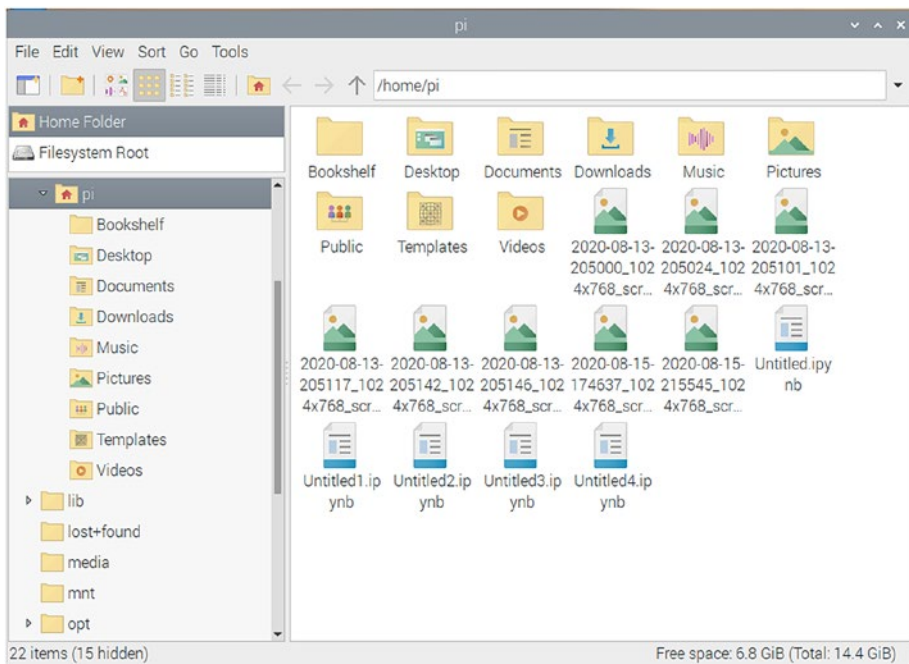


Figure 2-2. *File Explorer*

We can see many folders and files in the `/home/pi` folder. This folder is the home directory of the user **pi**. In the topmost part, under the menu bar, we can see an address bar. There, type in the character `/` and press Enter. This `/` is the root directory of the filesystem. The filesystems of most of the popular Unix-like operating systems are treelike structures, and the directory `/` is the root of that tree. Figure 2-3 is a File Explorer view of this directory.

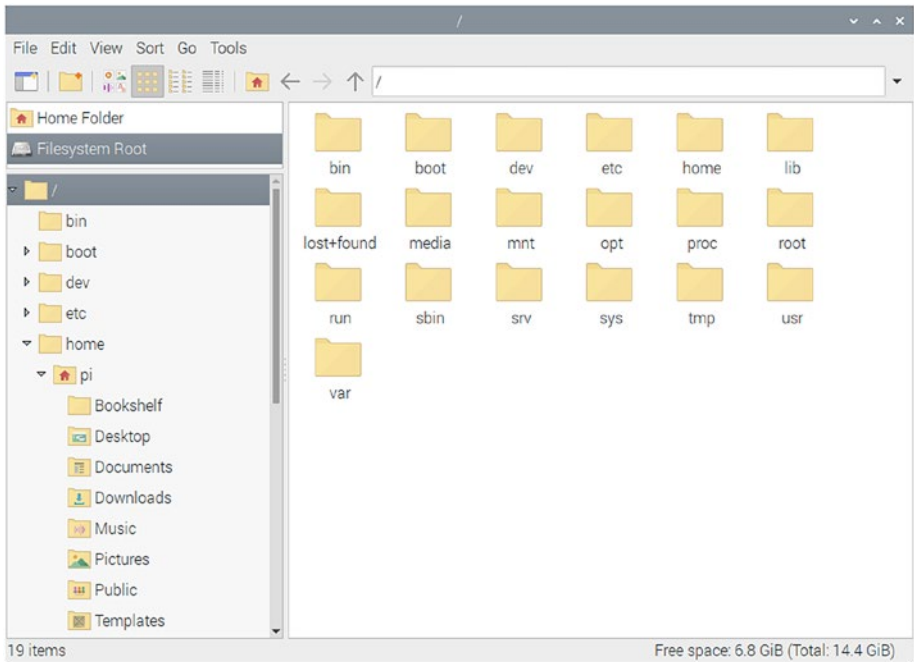


Figure 2-3. File Explorer view of the / (root) directory

Here, we can see many directories under the root directory. The following are brief descriptions of the important directories:

/bin: The /bin directory has many of the user executable files.

/boot: This directory has a bootloader, a kernel executable, and configuration files required to boot a Unix-like OS on a computer. In the RPi OS, the `config.txt` file has all the boot-related options.

/dev: This directory has the device files for all the hardware devices attached to the computer.

`/etc`: This directory contains the local system configuration files for the host computer.

`/home`: This is the home directory storage for user files. Each user has a subdirectory in this directory.

`/lib`: This directory has shared library files that are required to boot the system.

`/media`: Here, all the new storage devices are mounted. For example, when we attach a portable USB drive to the RPi, it will show up here.

`/mnt`: This is a temporary mount point for regular filesystems.

`/opt`: Optional files are located here. An example of optional files is the vendor-supplied programs.

`/root`: This is not the root directory of the (`/`) filesystem.

This is the home directory for the root user.

`/sbin`: These are the system binary files. These are the executable programs used for system administration.

`/tmp`: This is the temporary directory. It is used by the operating system and many programs to store temporary files. Users may also store files temporarily in this location. Note that files stored here may be purged by the OS without any warning.

`/usr`: These are shareable, read-only files, including executable binaries and libraries, man files, and other types of documentation.

/var: Variable data files are stored here. Examples are log files, MySQL and other database files, web server data files, email inboxes, and other program-specific files.

Remotely Accessing the RPi

We can remotely access the RPi's desktop and command prompt. For accessing the command prompt, we have already enabled the remote SSH while configuring the RPi after the OS setup. We can use any SSH client. However, I find the bitwise SSH client the most convenient. We can install it for the Windows OS by downloading it from www.bitvise.com/ssh-client-download. It is free of cost. Once we install it, we open it. We can find it by typing in SSH in the search bar of Microsoft Windows. The bitwise SSH connection window is as shown in Figure 2-4.

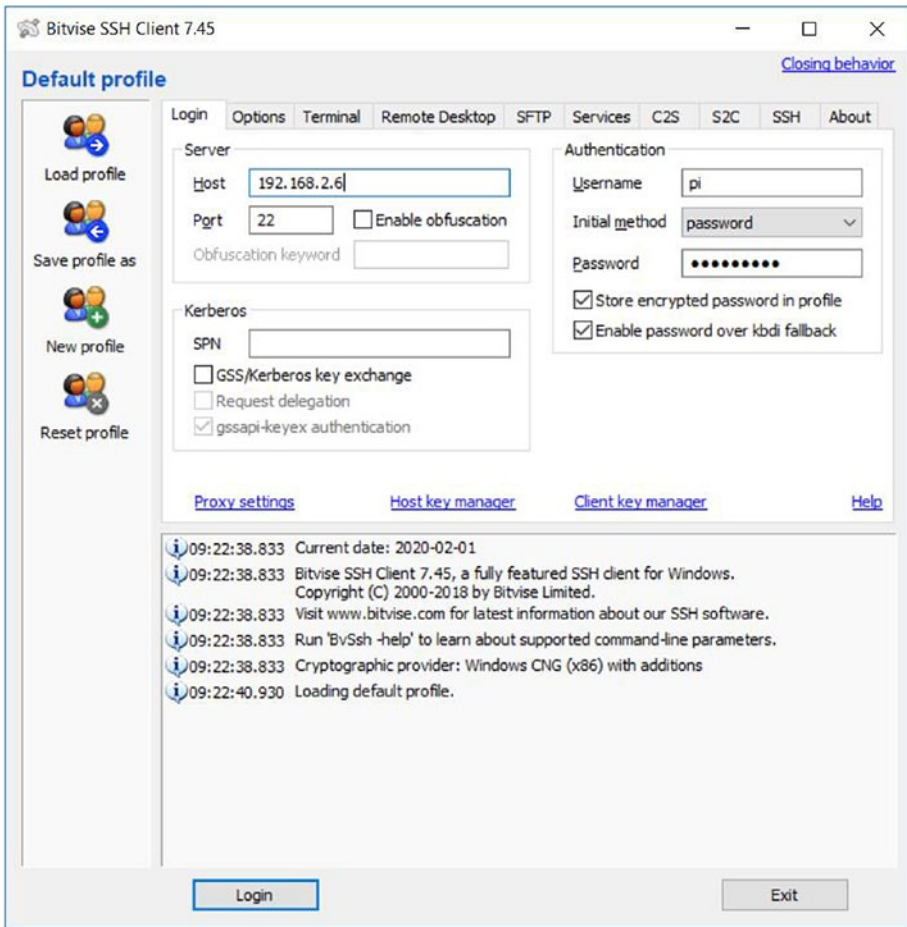


Figure 2-4. Bitwise SSH connection window

Fill in the details for the host, username, and password (pi and raspberry is the default combination, in case you forgot). Then click the **Login** button. When we log in to any new host the first time, it shows the message in Figure 2-5.

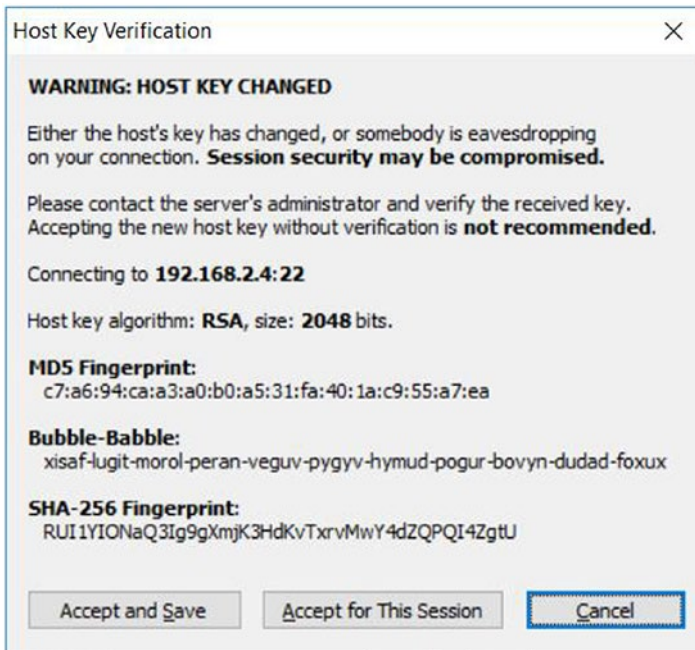


Figure 2-5. Host Key Verification Window

Click the **Accept and Save** button. This will save the host key of the RPi to the Windows computer, and this message will not be shown again for the same RPi when we make a fresh connection the next time. Once we connect, it will show us the RPi OS command prompt as shown in Figure 2-6.

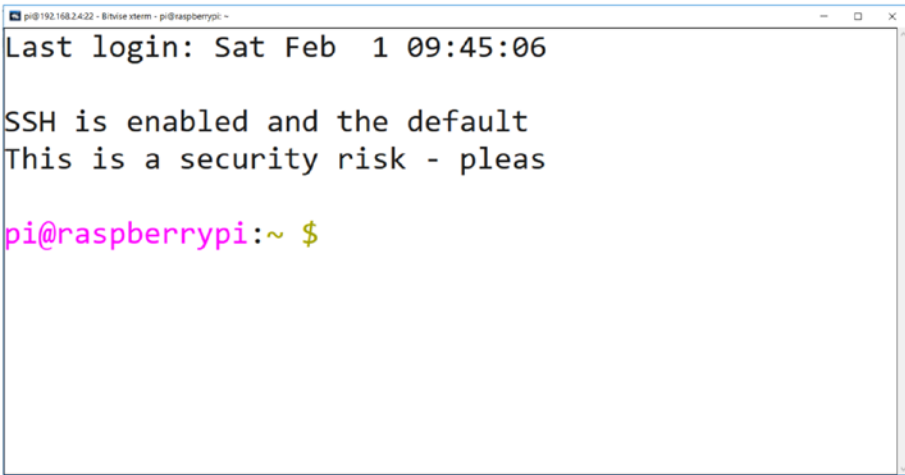


Figure 2-6. RPi OS command window remote access with SSH

It also opens a File Transfer window as shown in Figure 2-7.

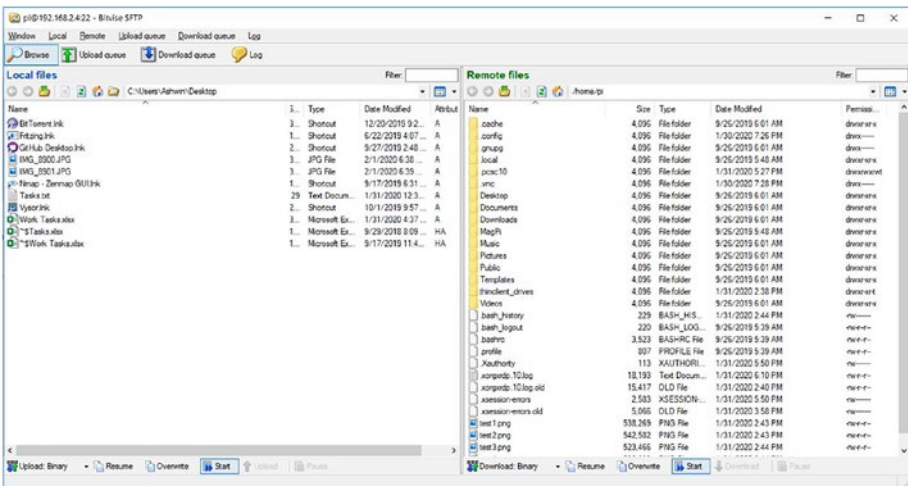


Figure 2-7. File Transfer Window

We can simply drag and drop files from Windows to RPi and vice versa. On the left, we have the Windows desktop for the current user, and on the right-hand side, we have the home directory for the user **pi** on the RPi OS.

This is how we can access the command prompt of the RPi OS and transfer files visually. Now, we will see how to remotely access the desktop. The RPi OS comes with the VNC server. We have already enabled the VNC server at the time of configuration after the installation. We just need to install a VNC viewer on our Windows PC. It can be done by downloading it from www.realvnc.com/en/connect/download/viewer/. We can search for it through Windows search by typing in VNC. The window for the application is as shown in Figure 2-8.

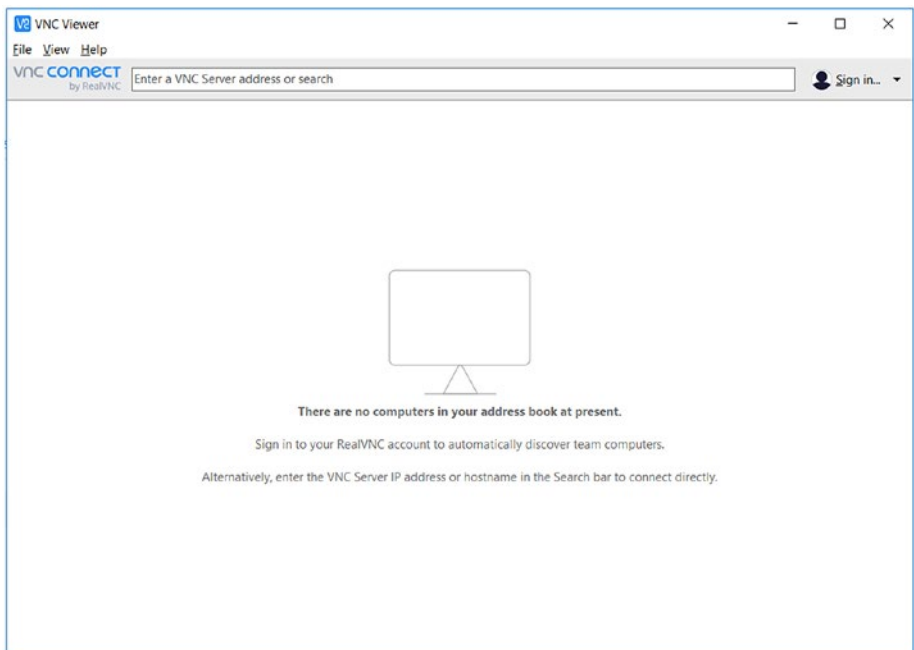


Figure 2-8. VNC viewer window

In the menu, click **File** ► **New Connection**. It opens a new connection window as shown in Figure 2-9.

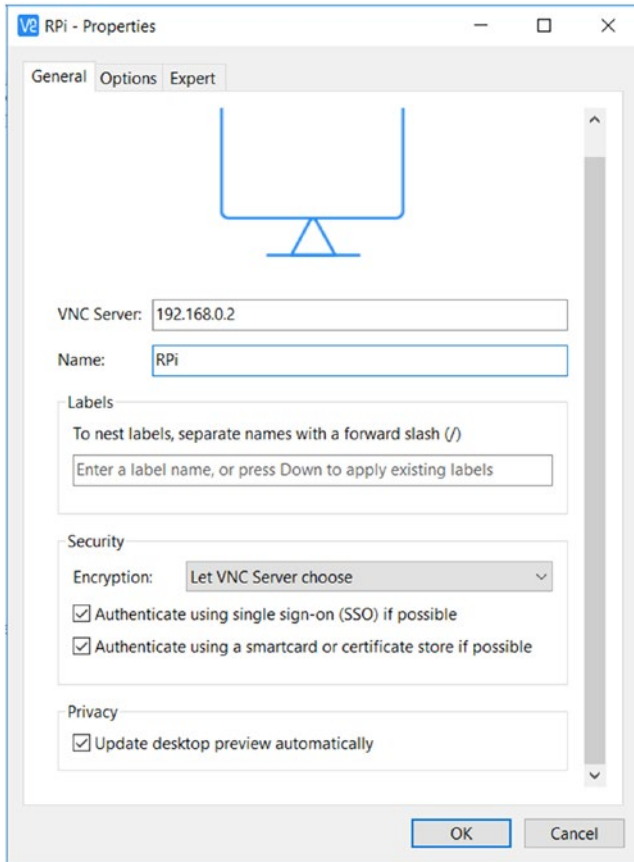


Figure 2-9. Connection details

Fill in the IP address and the name that you want to set for your connection, and click the **OK** button. It will create an icon corresponding to the connection in the VNC viewer application window. Double-click it to connect, and the window in Figure 2-10 will appear.

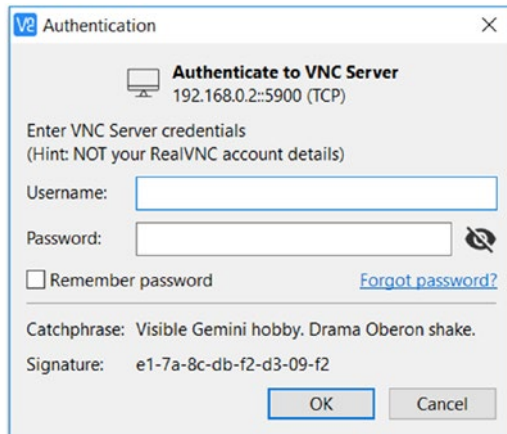


Figure 2-10. *Credentials*

Just key in the username and password. Click the checkbox **Remember password** so we won't be asked again for the credentials. Finally, click the **OK** button. It will open a remote desktop window as shown in Figure 2-11.

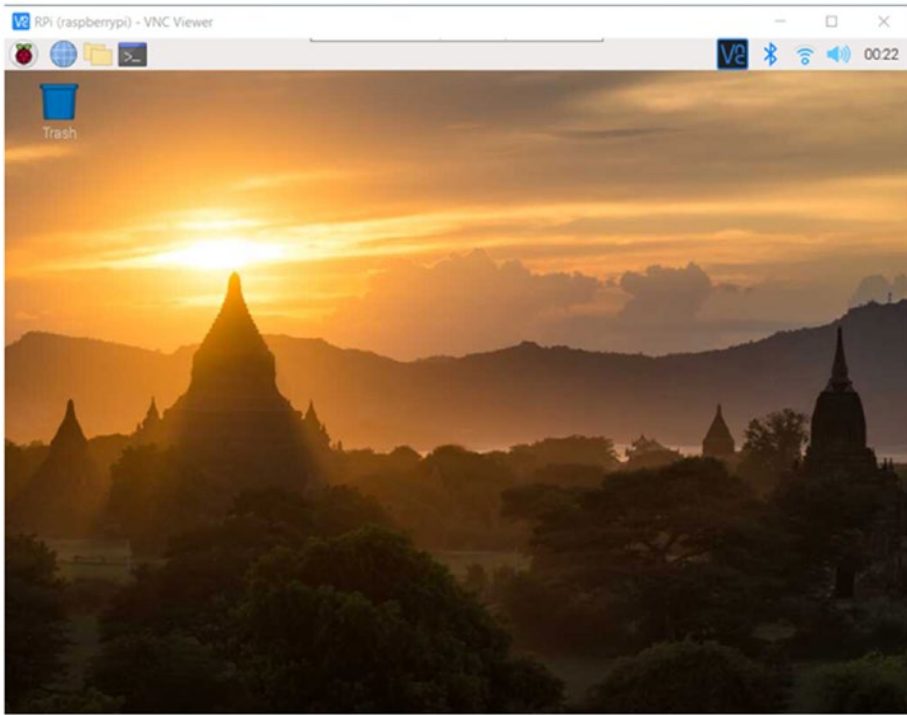


Figure 2-11. *Remote desktop with the VNC viewer*

Unless you are using very outdated networking equipment, the response is real time and smooth. We can carry out all the GUI-related tasks this way without needing an extra monitor.

So this is how we can access the RPi's command prompt and desktop remotely.

Summary

In this chapter, we learned the basics of the Linux shell, GUI, and command prompt. We also learned how to update the RPi OS and the firmware with commands. We finally learned how to access the command prompt and the desktop remotely. All these topics are needed to get started with learning the commands of the Linux operating system.

In the next chapter, we will continue our journey of exploring the Linux operating system. We will learn some simple file- and directory-related commands. We will also learn the basics of a few text editors.

CHAPTER 3

Directory Commands and Text Editors

In the last chapter, we learned how to work with the command prompt. We also updated our Pi's OS and the firmware. We learned how to access the Pi remotely. All these skills will be very helpful to us for all the demonstrations we will see in this and remaining chapters.

From this chapter onward, we will start learning the Linux commands. We will learn the commands related to files and directories.

We will explore the following concepts in this chapter:

- Absolute and Relative Paths
- Commands: `pwd`, `tree`, and `cd`
- Command: `ls`
- Command: `touch`
- Various Text Editors
- Create and Delete Directories
- Case-Sensitive Names of Directories and Files

After completing this chapter, we will be very comfortable with exploring the filesystem using the File Explorer as well as the command prompt.

Absolute and Relative Paths

When we are logged in as the user **pi**, if we open the **File Explorer** or **lterminal** utility, by default, they show us the **home** directory of the user **pi**. The path of this directory is `/home/pi`. This is the absolute path. We know that the `/` is the root of the filesystem. When we refer to any file or a directory in Unix-like operating systems, it can be referred with the absolute path. It means that it includes all the subdirectories, starting from the root directory `/`. `/home/pi` means that in the root directory `/`, there is a directory `home` that has a subdirectory `pi`. We usually write the absolute path in code or documentation in order to avoid any confusion. For example, `config.txt` is usually referred as `/boot/config.txt`.

The relative path of a file or a directory is the path in relation to a directory. For example, `/home/pi` is the absolute path. We can also say that just `/pi` is the path of the same directory relative to the `/home` directory.

We can paste the absolute path of a directory in the address bar of the File Explorer utility and press the Enter key to go to that directory. We can use the absolute path in the command prompt too to traverse to that directory. We will see that in the next section.

Commands: `pwd`, `tree`, and `cd`

We can check the name of the current directory with the command `pwd`. It means **p**resent **w**orking **d**irectory. Open the **lterminal**, and type in the command

```
pwd
```

Note that all Unix-like operating systems including all the distributions of Linux treat commands and filenames as case sensitive. Type all the commands and filenames mentioned in this book and any other documentation you come across as they are mentioned. A wrong case or mixed case will return an error.

The output of the command we executed is as follows:

```
pi@raspberrypi:~ $ pwd
/home/pi
```

In the preceding output, we can see `pi@raspberrypi`. It means that the user **pi** is logged in to the computer as **raspberrypi**. `$` is the prompt sign, and `pwd` is the command. We can see the output (the present working directory) in the next line. This is the absolute path. This is how we can see the path of the current directory.

We have learned that the Unix filesystem is like a tree structure and the root directory `/` is at the root of the filesystem. We can see this tree structure through a command `tree` that shows this. If the version of the RPi OS you are using does not have this command, then you can install it by running the following command:

```
sudo apt-get install tree
```

APT stands for **A**dvanced **P**ackage **T**ool. It is the tool to manage software installation, removal, and upgrade in Debian and derivatives. In order to install a new software from the repository, we have to mention its name after `sudo apt-get install`. Similarly, we can remove any software with `sudo apt-get remove` followed by the name of that software. For example, have a look at the following command:

```
sudo apt-get remove tree
```

In this case, you have removed the **tree** utility. Install it again with the command we learned earlier.

Execute the following command in the **lxterminal** in the **home** directory of the **pi** user:

```
tree
```

The results are shown in [Figure 3-1](#).

```

pi@192.168.0.222 - Bitvise xterm - pi@raspberrypi ~
pi@raspberrypi:~ $ tree
.
├── Bookshelf
│   └── 000_RPi_BeginnersGuide_DIGITAL.pdf
├── Desktop
├── Documents
├── Downloads
├── Music
├── Pictures
├── Public
├── Templates
└── Videos

9 directories, 1 file

```

Figure 3-1. Output of the `tree` command

The output shows all the directories, subdirectories, and files present in the current directory in the form of a tree as shown in Figure 3-1.

We can traverse to any directory by running the command `cd` as follows:

```
cd /
```

Executing the preceding command takes us to the root (`/`) directory of the filesystem. If the directory we want to switch to is in the current directory, we can use the relative path; otherwise, we must use the absolute path. We can directly go to the home directory of the current user by running the following command:

```
cd ~
```

We will frequently be using this command to switch working directories. We can go to the parent directory of the current directory with the following command:

```
cd ..
```

In Unix and derivatives, `..` refers to the parent directory of the current directory.

Command: ls

We have learned that we can use the command `tree` to show all the files, directories, and subdirectories under the current directory in a tree structure. There is another command, `ls`, that shows similar information. It means list, and it shows the files and directories (but not the subdirectories unless specified) in the form of a list. Run the command as follows:

```
ls
```

It shows the following output:

```
Bookshelf Documents Music Public Videos  
Desktop Downloads Pictures Templates
```

Many Unix commands can be run with options. We can pass one or more options by typing in the command followed by an empty space and then a hyphen sign followed by the options immediately. For example, if we want to see the list of files and directories in a long list format, then we can run the command as follows:

```
ls -l
```

The output is as follows:

```
pi@raspberrypi:~ $ ls -l
total 36
drwxr-xr-x 2 pi pi 4096 May 27 12:48 Bookshelf
drwxr-xr-x 2 pi pi 4096 May 27 13:16 Desktop
drwxr-xr-x 2 pi pi 4096 May 27 13:16 Documents
drwxr-xr-x 2 pi pi 4096 May 27 13:16 Downloads
drwxr-xr-x 2 pi pi 4096 May 27 13:16 Music
drwxr-xr-x 2 pi pi 4096 May 27 13:16 Pictures
drwxr-xr-x 2 pi pi 4096 May 27 13:16 Public
drwxr-xr-x 2 pi pi 4096 May 27 13:16 Templates
drwxr-xr-x 2 pi pi 4096 May 27 13:16 Videos
```

Let us see more options as follows:

- `-l`: It is the long listing format. This outputs file types, permissions, the number of hard links, owner, group, size, last-modified date, and filename.
- `-F`: It shows the nature of a file by appending a character after the filename. For example, `*` is for an executable and `/` for a directory. Regular files do not have a suffix.
- `-a`: It lists all the files in the given directory including the hidden files and directories (their names start with a `.` character in Unix).
- `-R`: This recursively lists all the subdirectories.
- `-t`: This sorts the list of files by modification time.
- `-h`: It print sizes of files in a human-readable format.
- `-1`: This forces the output to be one entry per line.

Let us try combining a few options:

```
ls -la
```

The output is as follows:

```
pi@raspberrypi:~ $ ls -la
total 104
drwxr-xr-x 19 pi  pi  4096 Aug 17 15:18 .
drwxr-xr-x  3 root root 4096 May 27 12:40 ..
-rw-----  1 pi  pi   979 Aug 17 15:56 .bash_history
-rw-r--r--  1 pi  pi   220 May 27 12:40 .bash_logout
-rw-r--r--  1 pi  pi  3523 May 27 12:40 .bashrc
drwxr-xr-x  2 pi  pi  4096 May 27 12:48 Bookshelf
drwxr-xr-x  8 pi  pi  4096 Jul 22 15:35 .cache
drwx-----  6 pi  pi  4096 Aug 17 12:23 .config
drwxr-xr-x  2 pi  pi  4096 May 27 13:16 Desktop
drwxr-xr-x  2 pi  pi  4096 May 27 13:16 Documents
drwxr-xr-x  2 pi  pi  4096 May 27 13:16 Downloads
drwx-----  3 pi  pi  4096 May 27 13:16 .gnupg
drwxr-xr-x  2 pi  pi  4096 Jul 26 11:58 .ipynb_checkpoints
drwxr-xr-x  5 pi  pi  4096 Jul 26 11:58 .ipython
drwxr-xr-x  6 pi  pi  4096 Jul 22 15:25 .local
drwxr-xr-x  2 pi  pi  4096 May 27 13:16 Music
drwxr-xr-x  2 pi  pi  4096 May 27 13:16 Pictures
drwx-----  3 pi  pi  4096 Jul 22 15:35 .pki
-rw-r--r--  1 pi  pi   807 May 27 12:40 .profile
drwxr-xr-x  2 pi  pi  4096 May 27 13:16 Public
drwxr-xr-x  2 pi  pi  4096 May 27 13:16 Templates
drwxr-xr-x  2 pi  pi  4096 May 27 13:16 Videos
drwx-----  3 pi  pi  4096 Jul 15 01:58 .vnc
-rw-----  1 pi  pi    56 Aug 15 08:32 .Xauthority
-rw-----  1 pi  pi  2624 Aug 15 08:32 .xsession-errors
-rw-----  1 pi  pi  2445 Aug 13 10:17 .xsession-errors.old
```


In the preceding output, `.` refers to the current directory, and `..` refers to the parent directory. All the files and directories that have `.` as the first character in their names are hidden. We cannot see them in the File Explorer too unless **Hidden items** is enabled. You may want to combine different options as an exercise for this section.

Command: touch

The command `touch` updates the modification date of an existing file passed to it as a parameter. If the file does not exist, then it creates a new empty file. For example, run the following command:

```
touch test.txt
```

It creates a file `test.txt` in the current directory. We can see it as follows:

```
ls -l touch.txt
```

The output is as follows:

```
pi@raspberrypi:~ $ ls -l test.txt
0 -rw-r--r-- 1 pi pi 0 Aug 17 16:48 test.txt
```

Let's run the command again after a minute:

```
ls -l touch.txt
```

And let's see its output again:

```
pi@raspberrypi:~ $ ls -l test.txt
-rw-r--r-- 1 pi pi 0 Aug 17 16:49 test.txt
```

We can observe that the last updated date has changed. We can run it with the existing filename in the parameter of the command to update the last access date without modifying the actual file.

Various Text Editors

We have already used the **nano** text editor to modify the network configuration. It is a WYSIWYG type of a plaintext editor for the command line. Other popular command line-based text editors for Unix are **vi** and **vim**. **vi** is included in the RPi OS by default. We need to install **vim** with the following command:

```
sudo apt-get install vim -y
```

This installs the **vim** editor. We can read more about **vi** and **vim** at the following URLs:

<https://vim.rtorr.com/>
<https://devhints.io/vim>

Both are a bit heavy for beginners. I prefer **nano**. If you want a GUI-based editor for editing the text files, you can opt for **Leafpad**. Install it with the following command:

```
sudo apt-get install leafpad -y
```

After the installation, we can find it in the **Accessories** option in the RPi OS menu. Figure 3-2 shows this in action:

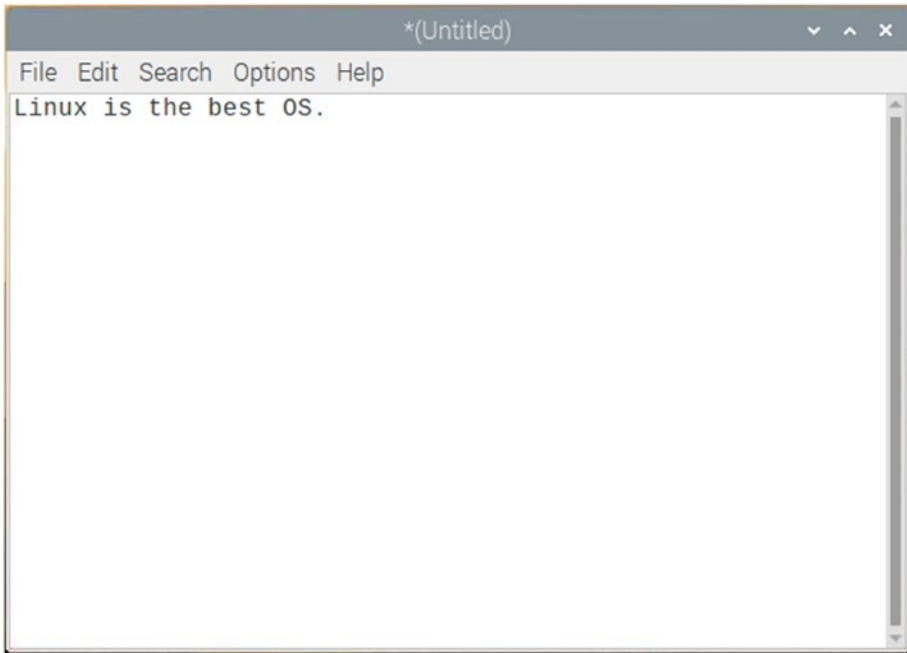


Figure 3-2. Leafpad in action

Create and Delete Directories

We can create and remove directories and files using the File Explorer. In the File Explorer window or on the desktop, if we right-click, we can see the menu shown in Figure 3-3.

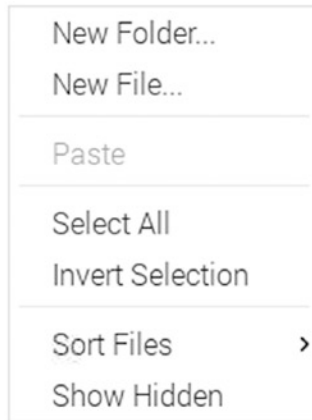


Figure 3-3. Options after right-clicking

We can create new directories (folders) and new files from this menu. Or we can use a command to create them. We have already seen how to use the command `touch` to create an empty file. Let us see how to create and remove a directory. We can use the command `mkdir` to create a directory as follows:

```
mkdir testdir
```

It creates a directory named `testdir` in the current directory. This is an empty directory. You can switch to it with the command `cd` and see it in the parent directory with the command `ls`. This is an empty directory, and an empty directory can be deleted by running any one of the following commands:

```
rm -d testdir
rmdir testdir
```

If you have created a few files in this directory, then you can use the command `rm` as follows to delete the files:

```
rm testfile1.txt
```

We can also delete a non-empty directory with all its contents as follows:

```
rm -r testdir
```

And of course, we can anytime use the File Explorer GUI to perform any of these operations.

Case-Sensitive Names of Directories and Files

Previously, I had mentioned that commands and names of directories and files are case sensitive in Linux. Let us demonstrate that now. Run the following commands in the **lxterminal** one by one:

```
mkdir test  
mkdir Test  
mkdir TEST
```

We can run the command `ls` to see these directories:

```
ls -lF
```

The output is as follows:

```
drwxr-xr-x 2 pi pi 4096 Aug 17 19:34 test/  
drwxr-xr-x 2 pi pi 4096 Aug 17 19:34 Test/  
drwxr-xr-x 2 pi pi 4096 Aug 17 19:34 TEST/
```

As we can see, it created all the directories with the same name. The case of characters is different in each name, so they are treated as different directories. In the Microsoft Windows OS, the names of files and directories are case insensitive.

We can also try to create files with the same name (with different cases for characters in names) with the command `touch` to see this in action for files. Try that as an exercise for this section.

Summary

In this chapter, we have started with a few basic yet important commands related to files and directories. These are very useful commands, and we will use them frequently throughout the book.

In the next chapter, we will continue our journey and learn more Unix commands.

CHAPTER 4

More Commands

In the last chapter, we learned the useful file- and directory-related commands. We also got ourselves acquainted with a few text editors. In that process, we also learned to use the APT utility to manage packages on Debian.

In this chapter, we will see more Linux commands. The following is the list of topics we will learn in this chapter:

- Configuring the RPi Board
- Getting Help on Commands
- Network-Related Commands
- Commands: File Operations
- Printing a String
- Control Operators
- Filename Globbing
- Command: History
- Pipes

After completing this chapter, we will be very comfortable with various useful commands in Linux. This chapter will instill more confidence in users about the command prompt.

Configuring the RPi Board

At the time of the installation of the RPi OS, we had seen the GUI tool for configuration of the RPi board. The command-line version of the same tool is known as the `raspi-config` utility. We can invoke it with the following command:

```
sudo raspi-config
```

The utility's main menu is as shown in Figure 4-1.

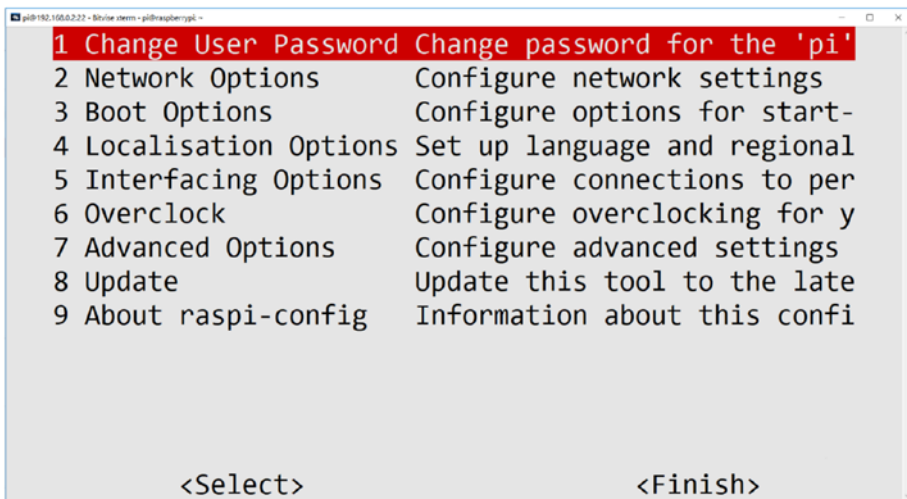


Figure 4-1. Raspberry Pi configuration utility at the command prompt

It has all the options we learned in the graphical tool. You may want to explore it further as an exercise for this section.

Note This command does not work in other distributions of Linux. It is specific to the RPi OS on RPi boards.

What Is sudo?

By this time, you must have noticed that we use the command `sudo` before a few commands. You also might have tried to run them without `sudo` and must have gotten the following error:

```
pi@raspberrypi:~ $ raspi-config
Script must be run as root. Try 'sudo raspi-config'
```

This is because a few commands and utilities need the security privileges of another user (usually `superuser` or the user `root` throughout this book). `sudo` is a program in Unix-like operating systems. It allows users to run programs with the security privileges of another user. By default, another user is the `superuser` (in our case, the user `root`). The command is expanded as "substitute user do" or "superuser do."

If any command needs `sudo` and we run it without `sudo`, it returns the error we learned in the preceding example.

Getting Help on Commands

We can get help on various commands with the commands `man` and `info`. We can use the command `man` with any other command as follows:

```
man ls
```

We will see a screen with the information of the command `ls`. This is known as a man page, and it is a form of documentation in the Unix-like systems. We can quit this documentation screen by pressing the key **Q** on the keyboard.

We can find similar information using the command `info` as follows:

```
info ls
```

It will show the information about the usage of the command. We can quit this information screen too by pressing the key **Q** on the keyboard.

Network-Related Commands

Let us have a look at a few network-related commands. The first command is `ifconfig`. It is a system administration utility and is run at the time of the boot. This command is used to set the IP address and netmask. If we run it without any parameters, then it shows the network details as follows:

```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether dc:a6:32:12:0c:e8 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 17 bytes 1004 (1004.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 17 bytes 1004 (1004.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.2 netmask 255.255.255.0 broadcast
    192.168.0.255
    inet6 fe80::7d45:b9a:284a:26bf prefixlen 64 scopeid
    0x20<link>
    ether dc:a6:32:12:0c:e9 txqueuelen 1000 (Ethernet)
    RX packets 4650 bytes 422988 (413.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4297 bytes 2465513 (2.3 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Currently, the RPi is connected to a WLAN network of my home. That is why the entries in the wlan0 section of the output (the last section) are enabled. For wired LAN, we can check the eth0 section (the first section) in the output.

Here, we can see important information like IPV4 and IPV6 addresses, netmask, broadcast address, and MAC settings. We also can see details like the number of received and sent packets.

Note The command `ifconfig` has many similarities to the command `ipconfig` in Windows and Mac.

Another command that shows similar information is `iwconfig`. It shows information about the currently connected WiFi as follows:

```
pi@raspberrypi:~ $ iwconfig
eth0      no wireless extensions.

lo        no wireless extensions.

wlan0     IEEE 802.11  ESSID:"Ashwin_Ion"
          Mode:Managed  Frequency:2.432 GHz  Access Point:
          6C:72:20:43:89:31
          Bit Rate=81 Mb/s   Tx-Power=31 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:on
          Link Quality=50/70  Signal level=-60 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:13  Invalid misc:0  Missed
          beacon:0
```

CHAPTER 4 MORE COMMANDS

We can test the reachability to a host in the internal or external network with the command `ping` as follows:

```
pi@raspberrypi:~ $ ping -c4 www.google.com
PING www.google.com (172.217.27.196) 56(84) bytes of data.
64 bytes from bom07s15-in-f4.1e100.net (172.217.27.196): icmp_
seq=1 ttl=119 time=8.80 ms
64 bytes from bom07s15-in-f4.1e100.net (172.217.27.196): icmp_
seq=2 ttl=119 time=8.15 ms
64 bytes from bom07s15-in-f4.1e100.net (172.217.27.196): icmp_
seq=3 ttl=119 time=7.86 ms
64 bytes from bom07s15-in-f4.1e100.net (172.217.27.196): icmp_
seq=4 ttl=119 time=8.03 ms

--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 8ms
```

In the example, `-c4` means we are sending four packets to the target host. It is optional, and in its absence, the command will run indefinitely.

We can download a file from the Internet with the command `wget` as follows:

```
wget ftp://ftp.gnu.org/pub/gnu/wget/wget-latest.tar.gz
```

This will download the mentioned file into the current directory. We can see the downloaded file with the command `ls` as follows:

```
pi@raspberrypi:~ $ ls *.gz
wget-latest.tar.gz
```

These are a few examples of very frequently used network-related commands in Linux.

Commands: File Operations

We can perform a variety of operations on files. Create an empty file and an empty directory in the current directory with the following commands:

```
touch abc
mkdir practice
```

Let us learn a few useful commands related to files. Let us see how to copy the file. We can copy it in the same location with a different name as follows:

```
cp abc abc1
```

We can see the original and the copy with the command `ls` as follows:

```
pi@raspberrypi:~ $ ls abc* -la
-rw-r--r-- 1 pi pi 0 Aug 22 15:33 abc
-rw-r--r-- 1 pi pi 0 Aug 22 15:33 abc1
```

In the output, we can see the file attributes too. We will learn about them later in the book.

We can copy it into a folder as follows:

```
cp abc ./practice/
```

The folder (or directory) `practice` is in the same folder. So we can provide the relative path. If it is not in the same folder, then we must provide the absolute path. Also, we can copy a file from any source to any target by providing the absolute paths.

We can rename the original file with the command `mv` as follows:

```
mv abc1 abc2
```

The original file will be renamed to another name. We can do this operation between directories too just as the command `cp`.

CHAPTER 4 MORE COMMANDS

Let us see a few more commands. Open the created file in a text editor and add 15–20 lines and save it. Then run the following command:

```
cat abc
```

It will show the contents of the file:

```
head abc
```

It shows the first ten lines. The head command shows the top lines in any source fed to it. Here, we are working with files. We can customize how many lines we want to see as follows:

```
head -5 abc
```

We can see the bottom lines with another command tail as follows:

```
tail abc
```

```
tail -5 abc
```

Let us study another file-related command cut in detail. It is used to extract the sections of each line in the output. A great example is extraction of data from a comma-separated value (CSV) file. In a CSV file, the data is arranged in columns, and they are separated by a comma (or some other delimiter like :). Data can be extracted by bytes, characters, or fields separated by a delimiter. The following command extracts the first two characters from the file:

```
cut -c 1-2 abc
```

We can use -f to choose the fields separated by a delimiter specified by -d. We can also use -b for bytes.

Printing a String

We can print a string with the command `echo`. The following are examples:

```
pi@raspberrypi:~ $ echo test
test
pi@raspberrypi:~ $ echo 'test'
test
pi@raspberrypi:~ $ echo "test"
test
```

Control Operators

Let us see a few control operators. Unix and derivatives have many control operators. Let us learn them one by one.

Run the following commands in sequence:

```
ls
echo $?
```

The last command returns 0. This is because `$?` stores the exit code of execution of the last command. If it is a success, it stores 0 and otherwise other code.

We can separate two commands with a semicolon (`;`) as follows:

```
echo test1 ; echo test2
```

Let us see the usage of the operator `&`. When a line ends with it, the shell does not wait for the command to finish execution. We get the shell prompt back.

Open the **lxterminal** program in GUI or using VNC. Then run the command `leafpad` to open the text editor. You will notice that as long as the editor is running, the command prompt is locked and not running typed-in commands. Once we close the editor, it will run those commands one by one (they are actually stored in a buffer). If we run the following command

```
leafpad &
```

it prints the PID (Process ID) of the program in the prompt, and the prompt is available for us to use. It does not wait for the editor to be closed.

Let us see the usage of the operator `&&`. It is a logical operator. Let us see an example as follows:

```
echo abc && echo xyz
```

When it is used between two commands, if the first command succeeds, then the second one is executed. If the first command fails, the second one is not executed. In the preceding example, both the commands run fine. Let us see another example:

```
fecho abc && echo xyz
```

In this case, both the commands are not executed.

Another logical operator is `||`. It is the logical OR. When placed between two commands, if the first command succeeds, the second one is not executed. The second command executes only if the first one fails. Check yourself by running the following examples:

```
echo abc || echo xyz  
fecho abc || echo xyz
```

We can combine both the operators in such a way that it prints a success message if the command succeeds; otherwise, it prints a failure message. The following is an example:

```
rm file1 && echo SUCCESS || echo FAIL
```


Finally, we can use the backslash operator `\` as an escape character. We need to print `;` on the command prompt, but the shell interprets it as end of the command. We can avoid that using a backslash as follows:

```
pi@raspberrypi:~ $ echo We want to print \;  
We want to print ;
```

Filename Globbing

Filename globbing is a feature of the UNIX shell. It means representing multiple filenames by using special characters called wildcards with a single filename. A wildcard is a symbol which is used to substitute for one or more characters. We can use wildcards to create a string that represents multiple filenames:

- `*` represents zero or more characters.
- `?` represents exactly one character.

Let us see a few examples. Run the following command:

```
ls a?c
```

It lists the file `abc`. As of now, in the **home** location, there is only one file that matches this criterion. The first and the last characters in the filename are `a` and `c`.

Let us see another example. Let us list all the files starting with character `a` in the filename:

```
ls a*
```

We can also list a file with the extension `txt` as follows:

```
ls *.txt
```

This is how we can use filename globbing with the command `ls`.

Command: History

Operating systems maintain the history of commands executed. We can find out the sequence of the commands executed in the shell with the command history. The following is a sample output of the command:

```
125 tail -5 abc
126 cut cut -c 1-2 abc
127 echo test
128 echo 'test'
129 echo "test"
130 echo abc && echo xyz
131 fecho abc && echo xyz
132 echo abc || echo xyz
133 fecho abc || echo xyz
134 rm file1 && echo SUCCESS && echo FAIL
135 rm file1 && echo SUCCESS || echo FAIL
136 history
```

I have shown only the ending part of the entire output as it will fill up several pages to show the whole output. As we can see, it shows the recent commands executed in the command prompt.

Pipes

Piping is a form of redirection. Using this, we can redirect the output of one command to another. Suppose I wish to see only the history of the last ten commands executed. Then I must use piping as follows:

```
history | tail -10
```

In the preceding command, `|` is the pipe operator. We are feeding the output of the command `history` to the command `tail`. The output is as follows:

```
131 fecho abc && echo xyz
132 echo abc || echo xyz
133 fecho abc || echo xyz
134 rm file1 && echo SUCCESS && echo FAIL
135 rm file1 && echo SUCCESS || echo FAIL
136 history
137 hist
138 history
139 ls -l
140 history | tail -10
```

While writing shell scripts, pipes are usually used in a creative way to filter the output of the commands executed.

Summary

In this chapter, we have started with a few commands of intermediate difficulty. These commands will be useful in writing the shell scripts which we will study later in this book. Now we all are very comfortable with the basic and intermediate-level use of the command prompt.

In the next chapter, we will study a few more useful commands and more complex concepts.

CHAPTER 5

Useful Unix Commands and Tools

In the last chapter, we learned a few more commands of intermediate difficulty. We are very comfortable now with the command prompt and can navigate the filesystem of Linux and other Unix-like OSs. We can use simple file and directory commands. We are also comfortable with various operators and piping.

In this chapter, we will learn advanced commands and tools in Unix. The following is the list of topics we will learn in this chapter:

- Shell and environment variables
- Useful Linux commands
- Useful Unix tools

After this chapter, we will be very comfortable with advanced tools in Unix. We will find these commands and concepts useful for learning shell scripting in the next chapter.

Shell and Environment Variables

Let us see how we can define variables in the shell. We can define numeric and string types of variables as follows:

```
a=2  
str1='ASHWIN'
```

We do not have to declare them as we do in programming languages like C or Java. These variables are known as shell variables. We can access them by prefixing the variable names with a \$ symbol as follows:

```
echo $a
echo $str1
```

The preceding statements will print the values stored in the variables. We can assign values belonging to any data type to a variable. Thus, the variables in the shell are not confined to storing values of any single data type.

An environment variable is a variable whose value is set with the functionality built into the operating system or shell. An environment variable is made up of a name and value pair. All system-related information is stored in the environment variable. We can see the list of environment variables by running the following command:

```
env
```

It will print a very long list of variables, and it will consume several pages. We have already seen that the variable SHELL stores the name of the executable file of the current shell. So we will have a look at the important environment variables. Run the following command to know the Bash shell version:

```
echo $BASH_VERSION
```

Run the following command to know the hostname of your RPi:

```
echo $HOSTNAME
```

Run the following command to know the location of the history file:

```
echo $HISTFILE
```

The following command returns the location of the **home** directory of the current logged user (in our case, the user **pi**):

```
echo $HOME
```

To know the directory location the shell searches for the executable files when we run any command, use the following command:

```
echo $PATH
```

Useful Linux Commands

Let us see a few useful commands in Linux. The command `w` shows who are logged in and what they are doing. Run the command and see the output.

The command `uptime` shows for how long the system is running:

```
pi@raspberrypi:~ $ uptime
17:05:24 up 19:10, 3 users, load average: 0.24, 0.22, 0.18
```

The command `who` shows who is logged in:

```
pi@raspberrypi:~ $ who
pi      tty1      2020-08-24 21:54
pi      pts/0    2020-08-25 16:42 (192.168.0.100)
pi      pts/1    2020-08-25 16:59 (192.168.0.100)
```

The command `whoami` prints the ID of the current user as follows:

```
pi@raspberrypi:~ $ whoami
pi
```

We can get information about the system with the following command:

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.4.51-v7l+ #1333 SMP Mon Aug 10 16:51:40 BST
2020 armv7l GNU/Linux
```

We can get information about the current processes and utilization of resources using the commands `htop` and `top`. Run them to see the output.

We can see a snapshot of current processes with the command `ps`:

```
pi@raspberrypi:~ $ ps -ef
UID          PID  PPID  C  STIME TTY          TIME CMD
root          1     0  0  Aug24 ?           00:00:04 /sbin/init
root          2     0  0  Aug24 ?           00:00:00 [kthreadd
root          3     2  0  Aug24 ?           00:00:00 [rcu_gp]
```

This is the partial output of the execution of the command.

The command `df` reports the details of the filesystem:

```
pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        15G  6.5G  7.3G  47% /
devtmpfs         1.8G   0  1.8G   0% /dev
tmpfs            1.9G   0  1.9G   0% /dev/shm
tmpfs            1.9G  8.7M  1.9G   1% /run
tmpfs            5.0M  4.0K  5.0M   1% /run/lock
tmpfs            1.9G   0  1.9G   0% /sys/fs/cgroup
/dev/mmcblk0p1  253M   54M  199M  22% /boot
tmpfs            378M  4.0K  378M   1% /run/user/1000
```

We can see the list of connected USB devices as follows with the command `lsusb`:

```
pi@raspberrypi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 005: ID 046d:081b Logitech, Inc. Webcam C310
Bus 001 Device 004: ID 046d:c077 Logitech, Inc. M105 Optical Mouse
Bus 001 Device 003: ID 1c4f:0002 SiGma Micro Keyboard TRACER
Gamma Ivory
```

Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub

Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

We can see processor information with the commands `lscpu` and `cat /proc/cpuinfo`. Run both the commands to see the output.

We can use the following commands to see information related to memory:

```
pi@raspberrypi:~ $ free -m
      total        used         free   shared  buff/cache   available
Mem:    3776         121        3320         36         334        3491
Swap:    99           0           99
pi@raspberrypi:~ $ cat /proc/meminfo
MemTotal:        3867184 kB
MemFree:         3399896 kB
MemAvailable:    3575016 kB
```

Unix commands are binary executable files. We can locate them with the commands `which` and `whereis`. The command `which` tells us the location of a binary executable:

```
pi@raspberrypi:~ $ which python3
/usr/bin/python3
```

We can retrieve information about the man page and documentation about the command with the command `whereis` as follows:

```
pi@raspberrypi:~ $ whereis python3
python3: /usr/bin/python3.7m /usr/bin/python3
/usr/bin/python3.7-config /usr/bin/python3.7
/usr/bin/python3.7m-config /usr/lib/python3 /usr/lib/python3.7
/etc/python3 /etc/python3.7 /usr/local/lib/python3.7
/usr/include/python3.7m /usr/include/python3.7
/usr/share/python3 /usr/share/man/man1/python3.1.gz
```


There is another Raspberry Pi OS-specific utility that can retrieve a lot of system information. It is `vcgencmd`. We can learn more about it at www.raspberrypi.org/documentation/raspbian/applications/vcgencmd.md.

The following are the examples of the execution:

```
pi@raspberrypi:~ $ vcgencmd measure_temp
temp=35.0'C
pi@raspberrypi:~ $ vcgencmd get_mem arm && vcgencmd get_mem gpu
arm=896M
gpu=128M
```

The first example shows the CPU temperature, and the second example shows the memory split (in megabytes) between CPU and GPU.

Useful Unix Tools

Let us study a few useful UNIX tools. These useful UNIX commands are found in all the distributions of Linux and other Unix-like operating systems. Let us create a simple CSV file for the demonstration. I have created a small CSV file for the demo, and its contents are as follows:

```
pi@raspberrypi:~ $ cat abc.csv
ASHWIN, 20k, INDIA
THOR, 10k, Asgard
JANE, 15k, UK
IRON MAN, 100k, USA
```

We can check the statistics of the file (number of words, lines, and characters including blank spaces) with the command `wc` as follows:

```
pi@raspberrypi:~ $ wc abc.csv
 4 13 71 abc.csv
pi@raspberrypi:~ $ wc -c abc.csv
71 abc.csv
```

```
pi@raspberrypi:~ $ wc -w abc.csv
13 abc.csv
pi@raspberrypi:~ $ wc -l abc.csv
4 abc.csv
```

The first example shows all the statistics of a file. The next three examples show the counts of characters, words, and lines, respectively. We can also use the command `cut` on this file for more practice. Have a look at the following examples:

```
pi@raspberrypi:~ $ cut -c 2-5 abc.csv
SHWI
HOR,
ANE,
RON
pi@raspberrypi:~ $ cut -d "," -f 2- abc.csv
20k, INDIA
10k, Asgard
15k, UK
100k, USA
```

We can use the command `grep` to find patterns of texts. For example, if I want to find my name in a text file, I can use the command `grep` as follows:

```
pi@raspberrypi:~ $ grep ASHWIN abc.csv
ASHWIN, 20k, INDIA
```

If I want the search to be case insensitive, then I can use it the following way:

```
pi@raspberrypi:~ $ grep -i asgard abc.csv
THOR, 10k, Asgard
```

CHAPTER 5 USEFUL UNIX COMMANDS AND TOOLS

We can sort data with the command `sort` as follows:

```
pi@raspberrypi:~ $ sort abc.csv
ASHWIN, 20k, INDIA
IRON MAN, 100k, USA
JANE, 15k, UK
THOR, 10k, Asgard
```

We can find out unique data items as follows:

```
pi@raspberrypi:~ $ sort abc.csv | uniq
ASHWIN, 20k, INDIA
IRON MAN, 100k, USA
JANE, 15k, UK
THOR, 10k, Asgard
```

To see the command in action, before running it, insert a duplicate line in the file `abc.csv`:

The following are the date and calendar commands in action:

```
pi@raspberrypi:~ $ date
Tue 25 Aug 2020 09:03:01 PM IST
pi@raspberrypi:~ $ cal
    August 2020
Su Mo Tu We Th Fr Sa
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Finally, if we want to find a file, then we can use the command `find` as follows:

```
pi@raspberrypi:~ $ find . -name "*.conf"
./config/lxterminal/lxterminal.conf
./config/lxsession/LXDE/desktop.conf
./config/pcmanfm/LXDE/desktop-items-0.conf
```

The command is followed by the path (in our case, it is the current directory, hence `.`) and criteria for the search. Here, we are searching for the configuration files in the current directory.

Summary

In this chapter, we learned many advanced Linux commands. We will use all the commands we learned in this and previous chapters to prepare shell scripts in the next chapter.

The next chapter will have detailed instructions on how to prepare and execute shell scripts on Linux.

CHAPTER 6

Shell Scripting

In the last chapter, we learned a few more useful commands in Unix and the RPi OS. These commands and tools we learned are extremely useful in writing shell scripts.

Continuing the theme of Unix and RPi OS commands of the last chapter, we will get started with shell scripting in this chapter and continue it to the next chapter. The following is the list of topics we will learn in this chapter:

- Unix File Permissions
- Command: nohup
- Beginning Shell Scripting
- User Input
- Expressions in the Shell
- If Statement
- Switch Case
- Length of a Shell Variable
- Command-Line Arguments
- Function
- Loops in the Shell
- Comparing Strings
- File Operations

After this chapter, we will be comfortable writing shell scripts on all Unix-like operating systems.

Unix File Permissions

Let us understand this concept by example. Run the following command in the lxterminal:

```
ls -l
```

The output is as follows:

```
pi@raspberrypi:~ $ ls -l
total 76
drwxr-xr-x 2 pi pi 4096 Aug 20 16:10 Bookshelf
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Desktop
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Documents
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Downloads
drwxr-xr-x 3 pi pi 4096 Aug 28 08:40 gnomeforpi
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Music
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Pictures
-rwxr-xr-x 1 pi pi 7980 Aug 29 17:36 prog00
-rw-r--r-- 1 pi pi 76 Aug 29 17:36 prog00.c
-rw-r--r-- 1 pi pi 22 Aug 29 18:38 prog00.py
-rwxr-xr-x 1 pi pi 8740 Aug 29 17:49 prog01
-rw-r--r-- 1 pi pi 93 Aug 29 17:48 prog01.cpp
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Public
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Templates
-rw-r--r-- 1 pi pi 3 Sep 3 16:06 test1.sh
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Videos
```

This is the Unix long list format. We have already seen this. Let us understand the meaning of all the terms in the output. Let us see the following two lines:

```
drwxr-xr-x 2 pi pi 4096 Aug 20 16:40 Pictures
-rwxr-xr-x 1 pi pi 7980 Aug 29 17:36 prog00
```

As we can see, there are nine columns in the output. The first column (drwxr-xr-x and -rwxr-xr-x) tells us about the file type and permissions. The first character is d if it is a directory and - if it is a file. The rest of the nine characters tell us about the permissions. We will see them in detail soon.

The second column has a number. It shows the number of links. The third and fourth columns are the owner and group names (pi and pi in this case). The fifth column shows the size in bytes. For directories, it is always 4 k. If you are interested, you can read more about it at <https://askubuntu.com/questions/186813/why-does-every-directory-have-a-size-4096-bytes-4-k>.

The next three columns are the last modification time. The last column shows us the name of the directory or file.

Let us discuss permissions in detail. You must have noticed the first column has a string like drwxr-xr-x. As we discussed, the first character denotes the file type. The next nine characters can be divided into three groups of three characters each. They are permissions to the user who created the file/directory, user's group, and others, respectively.

- r means read permission.
- w means write permission.
- x means execute permission.

If we encounter the - symbol anywhere, it means that permission is not granted. So the string -rwxr-xr-x means that it is a file. The creator/owner of the file has all (read, write, and execute) permissions. The owner's own group has read and execute permissions. And others have read and

execute permissions. We can manually alter permissions. We usually use numerical representation of the strings for the permissions:

- r means 4.
- w means 2.
- x means 1.

So, when we need to set permissions for a file/directory, we compute the sum for the owner, group, and others, respectively. For example, if I want to enable all permissions for all, then it will be `rxwxrwxrwx`. The numerical representation will be $(4+2+1\ 4+2+1\ 4+2+1)$ which can be written as `777`. For reasons of security, we rarely use this. The most common permission is `rxwxr--r--`. It can be represented as $(4+2+1\ 4+0+0\ 4+0+0)$ and written as `744`. Another common permission format is `rxwxr-xr-x`. It can be represented as $(4+2+1\ 4+0+1\ 4+0+1)$ and written as `755`. We will soon see how to grant these permissions on files and directories. We needed to understand this concept as it is very important to know it for executing programs on Unix-like operating systems.

Command: `nohup`

`nohup` means **No Hangup**. We use this command on the command prompt in cases where we do not want the program invoked to be terminated if we close the command prompt. We usually use it along with the `&` operator. To demonstrate this, we need to use the desktop of the RPi OS (either directly or remotely through VNC). Open the **lxterminal**. And then run the following command:

```
nohup idle &
```

It will open the IDLE (Integrated Development and Learning Environment) editor (it is an IDE for Python 3; we will see that in the later part of the book). Now if we close the **lxterminal** window, it will not

close the IDLE editor. However, if we invoke the IDLE with the command `idle` and nothing else, the IDLE will be closed if we close the **lxterminal** window used to invoke it.

Beginning Shell Scripting

Run the following command in the lxterminal:

```
pi@raspberrypi:~ $ echo "Hello World!"
Hello World!
```

As we can see, it immediately prints the output. We can create a script that has a collection of these statements. The statements in the script are fed to the shell interpreter and executed one by one. Such a script is known as **shell script**. Let us create one. Open any text editor and paste the command we executed there and save the file with the name **prog00.sh**. Generally, we use **.sh** as an extension for the shell script files. It is not necessary to have an extension. It just helps us to identify that these are shell scripts. We can easily list all the shell scripts in a directory with the following command:

```
ls *.sh
```

Once you create the file for the shell script, we can run it as follows:

```
bash prog00.sh
```

or

```
sh prog00.sh
```

Add the command `date` after the first line and run it again:

```
pi@raspberrypi:~ $ sh prog00.sh
Hello World!
Fri Sep 4 10:37:50 IST 2020
```

This is how we can write and execute simple shell scripts.

There is another way we can run the shell scripts. We need to change the permissions of the shell script with the following command:

```
chmod 755 prog00.sh
```

We have already discussed the meaning of the permissions represented by 755. Now we can directly run the script using `./` as follows:

```
./prog00.sh
```

It will execute the program and print the output. We can explicitly specify the interpreter with something known as **shebang** or **sha-bang**. Just add `#!/bin/bash` as the first line of the script shown in Listing 6-1.

Listing 6-1. prog00.sh

```
#!/bin/bash
echo "Hello World!"
date
```

This way the script uses the **Bash** shell to run when invoked directly.

User Input

We can accept input from a user in our shell script. In Listing 6-2, we are using the statement `read` to read the user input. We are reading the input into a variable and displaying its value. This is how we can read user input.

Listing 6-2. prog01.sh

```
#!/bin/bash
echo 'Who am I talking to?'
read name
echo 'Nice to meet you' $name
```

Expressions in the Shell

Just like any programming language, we can write expressions in the shell. The following shell script (Listing 6-3) shows various ways to write mathematical expressions in the shell.

Listing 6-3. prog02.sh

```
#!/bin/bash

let a=5+4
echo $a

let "a=5+4"
echo $a

let a++
echo $a

let "a=4*5"
echo $a
```

In this script, the first, second, and fourth expressions show us an assignment operation, and the third expression shows us an increment operation. We will frequently use expressions to assign values to variables in the shell. We can also use the `expr` statement to evaluate the arithmetic operations as shown in Listing 6-4.

Listing 6-4. prog03.sh

```
#!/bin/bash

expr 3 + 4

expr "3 + 4"
```

```

expr 11 % 2
a=$( expr 10 - 3 )
echo $a

```

The first and the third `expr` statements are arithmetic, and the second one is a string as the operand is enclosed by double quotes. The fourth statement is an assignment statement. Run the script and see the output.

Finally, Listing 6-5 shows a way to write expressions without `let` or `expr`.

Listing 6-5. prog04.sh

```

#!/bin/bash

a=$(( 4 + 5 ))
echo $a

a=$((3+5))
echo $a

b=$(( a + 3 ))
echo $b

b=$(( $a + 4 ))
echo $b

(( b++ ))
echo $b

(( b += 3 ))
echo $b

a=$(( 4 * 5 ))
echo $a

```

Run the script to see the output.

If Statement

We can have conditional statements using `if`. They use comparison operators. The following are the comparison operators in Bash:

- `-eq` is equal to.
- `-ne` is not equal to.
- `-lt` is less than.
- `-le` is less than or equal to.
- `-gt` is greater than.
- `-ge` is greater than or equal to.

Listing 6-6 shows usage of the `-eq` operator with an `if` statement.

Listing 6-6. prog05.sh

```
#!/bin/bash
echo 'Please enter an integer: '
read a
if [ $a -gt 100 ]
then
echo "The number is greater than 100."
fi
```

Run the script and see the output.

We can even have a nested `if` (Listing 6-7).

Listing 6-7. prog06.sh

```
#!/bin/bash
echo 'Please enter an integer:'
read a
if [ $a -gt 100 ]
then
echo 'It is greater than 100.'
if (( $a % 2 == 0 ))
then
echo 'It is an even number.'
fi
fi
```

Run the script and see the output.

We can write an if-else block (Listing 6-8).

Listing 6-8. prog07.sh

```
#!/bin.bash
echo 'Please enter an integer:'
read a
if [ $a -gt 50 ]
then
echo 'The number is greater than 50.'
else
echo 'The number is less than or equal to 50.'
fi
```

Run it to see the output. We can even write the if-elif-else statement (Listing 6-9).

Listing 6-9. prog08.sh

```
#!/bin/bash

echo 'Please enter an integer:'
read a

if [ $a -gt 100 ]
then
echo 'The number is greater than 100.'
elif [ $a -eq 100 ]
then
echo 'The number is equal to 100.'
else
echo 'The number is less than 100.'
fi
```

Switch Case

We can have a switch case construct (like C, C++, and Java) in the shell.

Listing 6-10 is a sample of such construct.

Listing 6-10. prog09.sh

```
#!/bin/bash

echo 'Please input an integer:'

read a

case $a in
10)
echo Ten
;;
20)
```

```
echo Twenty
;;
100)
echo Hundred
;;
*)
echo 'Default Case'
esac
```

Run the shell script to see the output.

Length of a Shell Variable

We can use the # symbol to compute the length of a variable (Listing 6-11).

Listing 6-11. prog10.sh

```
#!/bin/bash

a='Hello World!'
echo ${#a}

b=12345
echo ${#b}
```

Execute the script to know the length of variables.

Command-Line Arguments

Command Line Arguments are the arguments passed to any program (in this context, a shell script) at the time of invoking that script from the command line or any other script. Most of the modern programming (C, C++, and Java) and scripting (shell, Perl, Python) languages have provision for handling command-line arguments.

In a shell script, we can use \$# and \$@ to handle the command-line arguments as shown in Listing 6-12.

Listing 6-12. prog11.sh

```
#!/bin/bash

echo 'Total number of arguments:' $#
echo 'All argument values:' $@

echo 'Name of the script:' $0
echo 'First Argument ->' $1
echo 'Second Argument ->' $2
```

Run it as follows:

```
pi@raspberrypi:~ $ bash prog11.sh 1 "Hello World" 3.14 ASH
Total number of arguments: 4
All argument values: 1 Hello World 3.14 ASH
Name of the script: prog11.sh
First Argument -> 1
Second Argument -> Hello World
```

Function

We can even write functions in shell scripts. Functions are reusable blocks of code that are called frequently in a program. If you have a block of code that needs to be used frequently, it makes sense to write a function around it. Listing 6-13 shows a demonstration of a simple function.

Listing 6-13. prog12.sh

```
#!/bin/bash
print_message ()
{
    echo 'Hello from function'
}

print_message
```

In the script, `print_message()` is a function, and we are calling it exactly one time as shown. We are first defining its body and then calling it (when we call, the brackets are not used). We can even have arguments in functions, and they work exactly the same as command-line arguments. You can read more about it at https://bash.cyberciti.biz/guide/Pass_arguments_into_a_function.

Loops in the Shell

We can write a loop using the `until` statement and comparison operator as shown in Listing 6-14.

Listing 6-14. prog13.sh

```
#!/bin/bash

counter=0

until [ $counter -gt 5 ]
do
    echo 'Counter:' $counter
    ((counter++))
done
```

Run the script and it shows the following output:

```
pi@raspberrypi:~ $ bash prog13.sh
Counter: 0
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
```

We can even have for loops as shown in Listing 6-15.

Listing 6-15. prog14.sh

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo 'Looping ... number' $i
done
```

The output is as follows:

```
pi@raspberrypi:~ $ bash prog14.sh
Looping ... number 1
Looping ... number 2
Looping ... number 3
Looping ... number 4
Looping ... number 5
```

We can compute a factorial with a for loop as shown in Listing 6-16.

Listing 6-16. prog15.sh

```
#!/bin/bash
echo 'Enter a number'
read num
```

```

fact=1
for((i=2;i<=num;i++))
{
    fact=$((fact * i))
}
echo $fact

```

We can write the same program with a while loop as shown in Listing 6-17.

Listing 6-17. prog16.sh

```

#!/bin/bash
echo 'Enter a number:'
read num

fact=1

while [ $num -gt 1 ]
do
    fact=$((fact * num))
    num=$((num - 1))
done

echo $fact

```

Run all these programs to see the loops in action.

Comparing Strings

We can compare strings in the shell with string comparison operators. Listing 6-18 shows string comparison in action.

Listing 6-18. prog17.sh

```
#!/bin/bash

a='GNU'
b='Linux'

if [ $a = $b ]
then
    echo "$a = $b : a is equal to b"
else
    echo "$a = $b : a is not equal to b"
fi

if [ $a != $b ]
then
    echo "$a = $b : a is not equal to b"
else
    echo "$a = $b : a is equal to b"
fi

if [ -z $a ]
then
    echo "-z $a : length of a is zero"
else
    echo "-z $a : length of a is not zero"
fi

if [ -n $a ]
then
    echo "-n $a : length of a is not zero"
else
    echo "-n $a : length of a is zero"
fi
```

```
if [ $a ]
then
    echo "$a : string is not empty"
else
    echo "$a : string is empty"
fi
```

We can change the values of strings stored in variables `a` and `b` to experiment with different outcomes.

File Operations

We can check files for various conditions with the following operators:

- `-r` checks if the file is readable.
- `-w` checks if the file is writeable.
- `-x` checks if the file is executable.
- `-f` checks if the file is an ordinary file.
- `-d` checks if the file is a directory.
- `-s` checks if the file size is zero.
- `-e` checks if the file exists.

Listing 6-19 shows the usage of all these operators.

Listing 6-19. prog18.sh

```
#!/bin/bash

file='prog12.sh'

if [ -r $file ]
then
    echo 'File has read access.'
```

```
else
    echo 'File does read access.'
fi
if [ -w $file ]
then
    echo 'File has write permission.'
else
    echo 'File does not have write permission.'
fi
if [ -x $file ]
then
    echo 'File has execute permission.'
else
    echo 'File does not have execute permission.'
fi
if [ -f $file ]
then
    echo 'File is an ordinary file.'
else
    echo 'File is not an ordinary file.'
fi
if [ -d $file ]
then
    echo 'File is a directory.'
else
    echo 'File is not a directory.'
fi
if [ -s $file ]
then
    echo 'File size is not zero.'
```

```
else
    echo 'File size is zero.'
fi

if [ -e $file ]
then
    echo 'File exists.'
else
    echo 'File does not exist.'
fi
```

Run the script to see the output.

Summary

In this chapter, we have gotten started with shell scripts in the shell. As discussed earlier, shell scripts can be very useful tools for programmers who are tasked to complete various activities. We have explored shell scripts in quite detail. However, there is more to shell scripting than we have covered. If you are entrusted with tasks related to Unix-like systems, then you can use shell scripts at your work to get the desired results. Mastery in scripting requires a lot of practice in real-life tasks.

In the next chapter, we will explore I/O redirection techniques and learn about a useful utility known as **crontab**. We will demonstrate that with a real-life project.

CHAPTER 7

I/O Redirection and Cron

In the last chapter, we got started with shell scripting for Unix-like OSs. We are comfortable with shell scripts now.

Continuing the momentum, we will learn a couple of useful features in Unix-like operating systems' shell. The following is the list of topics we will learn in this chapter:

- I/O redirection
- Crontab

After this chapter, we will be comfortable using the above-mentioned advanced features in the shell. This is a very short chapter.

I/O Redirection

In Unix-like operating systems, standard streams are interconnected input and output channels. The three standard interconnected channels are standard input (stdin), standard output (stdout), and standard error (stderr). They are attached to file handles (or file descriptors) 0, 1, and 2, respectively. Almost all the programming and scripting environments come with provisions to handle these streams. Let us see them one by one.

stdin

This is the standard input. In most of the cases, it is the input from the keyboard. Let us see an example. Create a file known as `New.txt` in the current location and add some characters to the file. We know that we can show the contents of a file with the command `cat` as follows:

```
cat New.txt
```

We can use `stdin` to feed data to the `cat` command as follows:

```
cat < New.txt
```

It will show us the contents of the file mentioned in the command. We know that Unix associates 0 with `stdin`. So the preceding command can also be written as follows:

```
cat 0< New.txt
```

stdout

The standard output is usually the visual display console (it could also be the remote terminal). It is represented by the file handle 1. Suppose I wish to redirect it to a file. Then I can write a command as follows:

```
ls -l 1> output.log
```

When we execute the preceding command, it will not show the output in the terminal. It will redirect the output to the file mentioned in the command. We can see the contents of that file with the `cat` command as follows:

```
cat output.log
```

We can also omit 1 and write the command `ls -l 1> output.log` as follows:

```
ls -l > output.log
```

The `>` redirection operator overwrites previous data in the file mentioned in the command if the file exists already. We can append to the existing data with the `>>` redirection operator as follows:

```
ls -l 1>> output.log
ls -l >> output.log
```

Both the commands perform the same function.

Stderr

We can redirect the standard error. It is represented by the file handle 2. We can use it as follows:

```
ls -l 2>error.txt
```

It will redirect the error in the command (if any) to the file mentioned in the command. We can also redirect it and overwrite the error log as follows:

```
ls -l 2>>error.txt
```

We can redirect the output and error to the same file as follows:

```
ls -l 1>output.txt 2>&1
```

We can also write it as follows:

```
ls -l >output.txt 2>&1
```

A device file that rejects all information redirected to it is a null device. In Unix-like operating systems, it is `/dev/null`. It is also known as the Unix Black Hole (informally). We usually redirect error to this file. It is used as follows:

```
ls -l 2>/dev/null
```

Crontab

Cron is a time-based scheduler in Unix-like operating systems. We can run a program or a script at a set time at regular intervals. We have to use a file known as **crontab** to set cron jobs. We need to make entries for our scheduled jobs to run. The format is as follows:

(minute) (hour) (day of the month) (month) (day of the week)
<Program or script to run>

- **Minute** can have values from 0 to 59.
- **Hour** can have values from 0 to 23.
- **Day of the month** can have values from 1 to 31.
- **Month** can have values from 1 to 12.
- **Day of the week** can have values from 0 to 6.

Let us see a few examples. The following entry runs the specified script every day at midnight:

```
0 0 * * * /home/pi/backup.sh
```

We can run a script or a program at the time of every reboot as follows:

```
@reboot /home/pi/test2.sh
```

We can run a program every five minutes as follows:

```
*/1 * * * * /home/pi/test3.sh
```

We can use the following command to see the current entries in the crontab:

```
crontab -l
```

We can use the following command to edit the entries in the crontab:

```
crontab -e
```

It will ask you for the choice of editor when you run it the very first time. Choose the **nano** editor.

Let us see an example of a simple shell script and usage of crontab. Let us create a small shell script as shown in Listing 7-1.

Listing 7-1. reboot_hist.sh

```
#!/bin/bash
logfile=reboot_hist.log
echo "System rebooted at : " >> $logfile
date >> $logfile
```

Change its permissions with the following command:

```
chmod 755 reboot_hist.sh
```

Finally, add the following entry to the **crontab**:

```
@reboot /home/pi/reboot_hist.sh
```

Then when we reboot, it will write the reboot time to the specified log file. I rebooted a couple of times and then checked the log file. The contents are as follows:

```
pi@raspberrypi:~ $ cat reboot_hist.log
System rebooted at :
Thu Sep 10 20:04:42 IST 2020
System rebooted at :
Thu Sep 10 20:07:01 IST 2020
```

Summary

In this short chapter, we learned two advanced features of the shell: I/O redirection and usage of crontab. We also saw a small example of the crontab entry. Both these features are quite useful while writing shell scripts.

With this chapter, we conclude our journey of shell scripts. In the next chapter, we will learn how to program with high-level programming languages like C, C++, and Python 3 on Linux.

CHAPTER 8

Introduction to High-Level Programming Languages

In the last chapter, we concluded Linux shell scripting. In this chapter, we will learn how to write programs with high-level programming languages like C, C++, and Python 3 with the RPi OS. The following is the list of topics we will learn in detail in this chapter:

- C and C++ programming
- Python programming language
- Python 3 on Debian derivatives

After this chapter, we will be comfortable with writing programs using modern programming languages on the Linux platform.

C and C++ Programming

C and C++ are very versatile programming languages. As of now, almost all the operating systems (to be very specific, OS kernel and device drivers) are written in the C programming language. I have been writing C and C++ programs on the Linux platform since 2003. I mostly use GCC (GNU Compiler Collection) for compiling C programs using any of the Linux distributions. Also, I use the g++ compiler for C++ on Linux. So, in this section, I will give you a brief overview of the process of compiling using GCC and g++ compilers. We will also see how to execute the compiled bytecode.

Both these compilers are preinstalled on the RPi OS. In case you want to make sure, just run the following command:

```
sudo apt-get install build-essential
```

It will return a message that has the following string:

```
build-essential is already the newest version (12.6).
```

It means that GCC and g++ are already installed.

We can verify the versions with the following commands:

```
gcc -v
```

```
g++ -v
```

We can use any text editor to create and save a C program named as prog00.c as follows:

```
#include<stdio.h>
int main( void )
{
    printf("Hello, World\n");
    return 0;
}
```

Compile it with the following command:

```
gcc prog00.c -o prog00
```


Here, the string following `-o` is the name of the executable file for our program. If we do not use `-o` followed by the filename, it creates a `.out` executable file by default. Run the executable file with the following command:

```
./prog00
```

It will print the output as follows:

```
Hello, World
```

We can also write a C++ program, `prog01.cpp`, as follows:

```
#include<iostream>
using namespace std;
int main(void)
{
    cout<<"Hello, World\n";
    return 0;
}
```

We can compile it as follows:

```
g++ prog01.cpp -o prog01
```

We can run it as follows:

```
./prog01
```

The output will be as follows:

```
Hello, World
```

This is how we can compile and run C and C++ programs on the Linux platform. This section is no way a comprehensive guide for GCC and g++. You may be interested in finding out more information about GCC. You can visit the homepage of the **GCC** project here: <https://gcc.gnu.org/>.

Python Programming Language

Python 3 is a high-level and interpreted programming language. It is a general-purpose programming language. In this section, we will have in-depth general discussion about the Python programming language and its philosophy.

History of the Python Programming Language

Python is a successor to the **ABC** programming language which itself is inspired by **ALGOL 68** and **SETL** programming languages. It was created by **Guido van Rossum** as a personal side project during vacations in the late 1980s while he was working at Centrum Wiskunde & Informatica (CWI) (English: “National Research Institute for Mathematics and Computer Science”) in the Netherlands. From the initial release of the Python programming language till July 12, 2018, Guido has been the lead developer and *Benevolent Dictator for Life (BDFL)* for this project. After July 12, he has gone into **permanent vacation** and now works in a steering committee for Python. The following are the important milestones in Python’s release timeline:

- February 1991: van Rossum published the code (labeled version 0.9.0) to alt.sources.
- January 1994: Version 1.0 was released.
- October 2000: Python 2.0 was released.
- December 2006: Python 3.0 was released.
- December 2019: Python 2.x was officially retired and is no longer supported by the Python Software Foundation.

As we can see in the timeline, Python 2.x versions are no longer supported, and Python 3 is the most recent. Python 3 is not backward compatible with Python 2. Python 3 is the latest and supported version of the Python programming language. So we will use Python 3 throughout the book to demonstrate programs for data visualization. Unless explicitly mentioned, Python means Python 3 throughout this book.

Python Enhancement Proposals

For steering the development, maintenance, and support of Python, the Python leadership came up with the concept of Python Enhancement Proposals (PEPs). They are the primary mechanism for suggesting new features and fixing issues in the Python project. We can read more about the PEPs at the following URLs:

www.python.org/dev/peps/

www.python.org/dev/peps/pep-0001/

Philosophy of the Python Programming Language

The philosophy of Python is detailed in PEP20. It is known as **the Zen of Python** which can be found at www.python.org/dev/peps/pep-0020/. The following are the points from that PEP. A few are funny:

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.

8. Special cases aren't special enough to break the rules.
9. However, practicality beats purity.
10. Errors should never pass silently.
11. Point 10 is so unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one – and preferably only one – obvious way to do it.
14. However, that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. However, never is often better than “right” now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea – let's do more of those!

These are general philosophical guidelines that influenced the development of the Python programming language and continue to do so.

Applications of Python

As we have seen that Python is a general-purpose programming language, it has numerous applications in the following areas:

1. Web development
2. GUI development
3. Scientific and numerical computing
4. Software development
5. System administration

We can read case studies of Python for various projects at the www.python.org/success-stories/.

Python 3 on Debian Derivatives

Python 3 comes preinstalled on Debian and all the derivatives like Ubuntu or the Raspberry Pi OS. So we do not have to install it separately. Both the major Python versions, Python 2 and Python 3, come preinstalled on Debian derivatives. Python 2 and Python 3's executables are named as **python** and **python3**, respectively. We must use the executable file **python3** for our demonstrations. To know the version and location of the needed binary executable file, run the following commands one by one:

```
python3 -V
which python3
```

Python Modes

Python has various modes. Let us discuss them one by one. Before we get started with that discussion, we will see what IDLE is. IDLE stands for Integrated Development and Learning Environment, and it is an IDE (Integrated Development Environment) developed by the Python Software Foundation for Python programming.

All the Linux distributions may not come with IDLE preinstalled. We can install it on Debian and its derivatives (including the Raspberry Pi OS) by running the following commands in sequence:

```
sudo apt-get update --fix-missing
sudo apt-get install idle3 -y
```

Once installation is completed, we can find the IDLE in the menu (in this case, the Raspberry Pi OS menu) as shown in Figure 8-1.

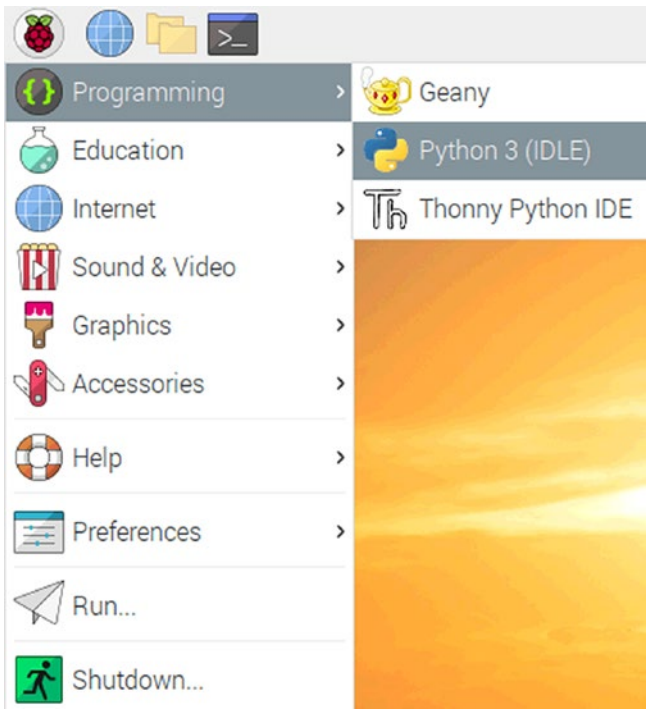


Figure 8-1. IDLE in the Raspberry Pi OS menu

We can also launch IDLE on Linux by running the following command:

```
idle
```

It will launch a window as shown in Figure 8-2.

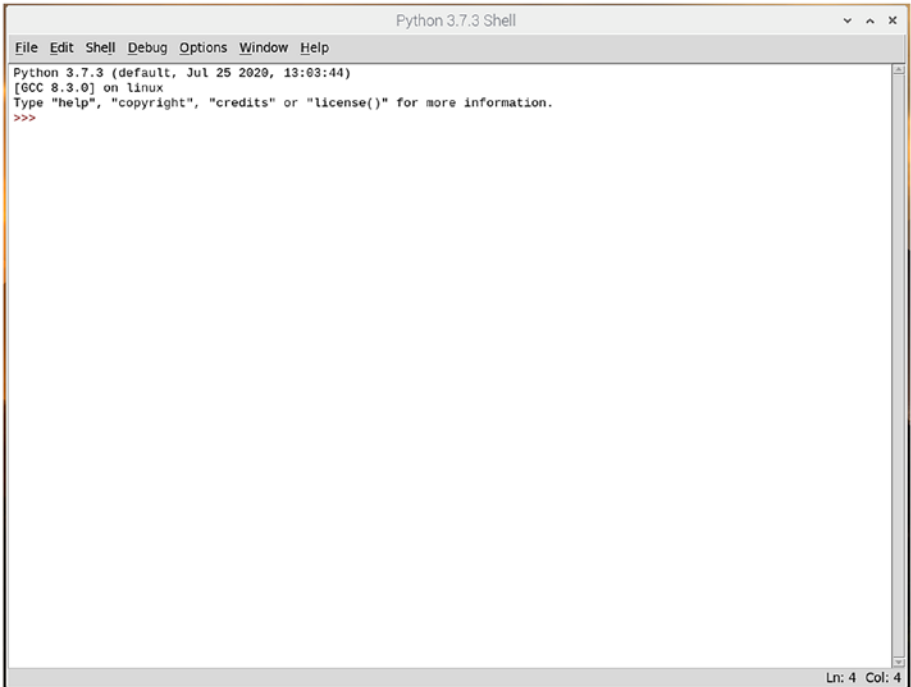


Figure 8-2. IDLE window

Before we proceed, we need to configure it for the comfort of our eyes. We can change the font by clicking **Options ► Configure IDLE** as shown in Figure 8-3.

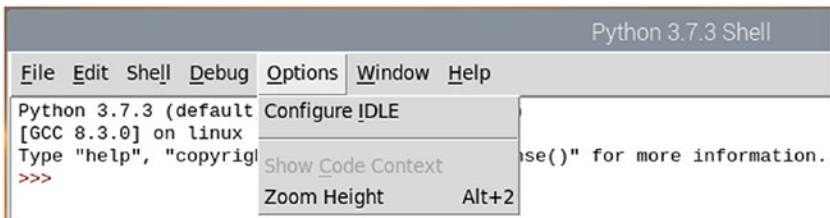


Figure 8-3. Configure IDLE

The following window opens (as shown in Figure 8-4) where you can change the font and size of the characters in the IDLE.

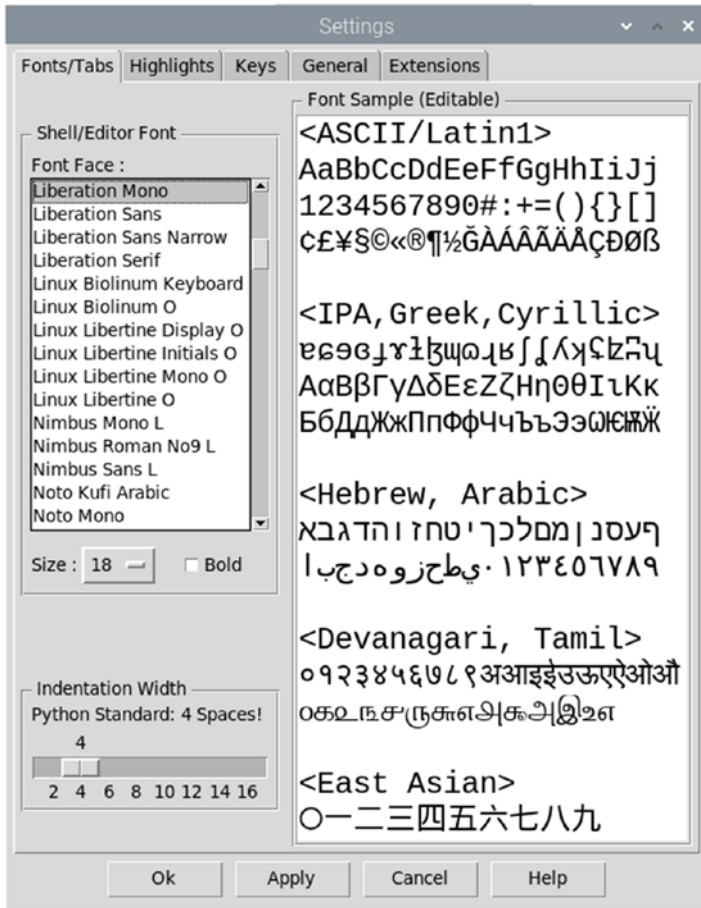


Figure 8-4. IDLE configuration window

Now, let us discuss various modes of Python.

Interactive Mode

Python's interactive mode is a command line–type of shell which executes the current statement and gives immediate feedback on the console. It runs the previously fed statements in active memory. As new statements are fed into and executed by the interpreter, the fed code is evaluated. When we open the IDLE, we see a command-line prompt. It is nothing but Python's interactive mode. Let's see a simple example. Let us type in the customary **Hello World** program in the interactive prompt as follows:

```
print('Hello World!')
```

Press the Enter key to feed the line to the interpreter and execute it. Figure 8-5 is a screenshot of the output.

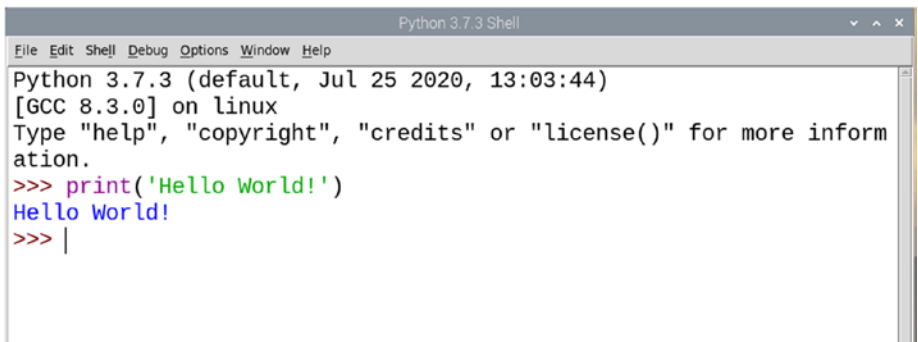
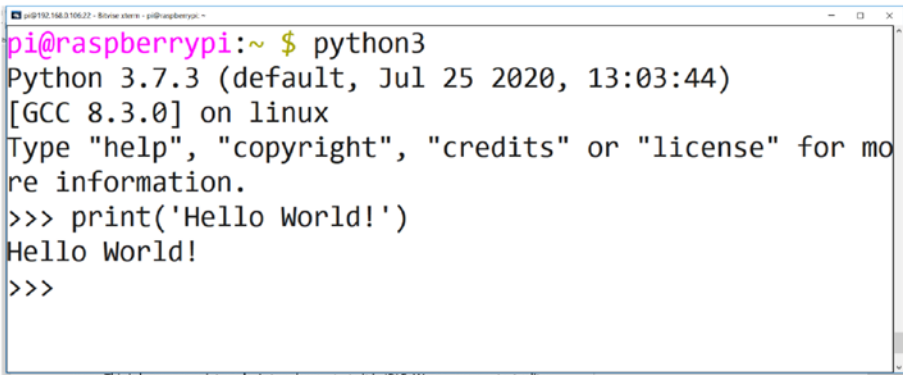


Figure 8-5. Python interactive mode on IDLE

We can launch Python's interactive mode from the command prompt too. In the Linux command prompt (e.g., `lxterminal`), we must run the command **python3** to launch it. Figure 8-6 is a screenshot of the interactive mode in the RPi OS command prompt (accessed remotely).



```
pi@raspberrypi:~ $ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World!')
Hello World!
>>>
```

Figure 8-6. Python interactive mode in the Linux command prompt

Script Mode

We can write a Python program and save it on the disk. Then we can launch it in multiple ways. This is known as script mode. Let us demonstrate it in IDLE. We can use any text editor to write the Python program. However, as IDLE is an IDE, it is convenient to write and run Python programs with IDLE. Let's see that first. In the IDLE, click **File** ➤ **New File**. It will create a blank new file. Add the following code to that:

```
print('Hello World!')
```

Then save it with the name **prog00.py** on the disk as shown in Figure 8-7.

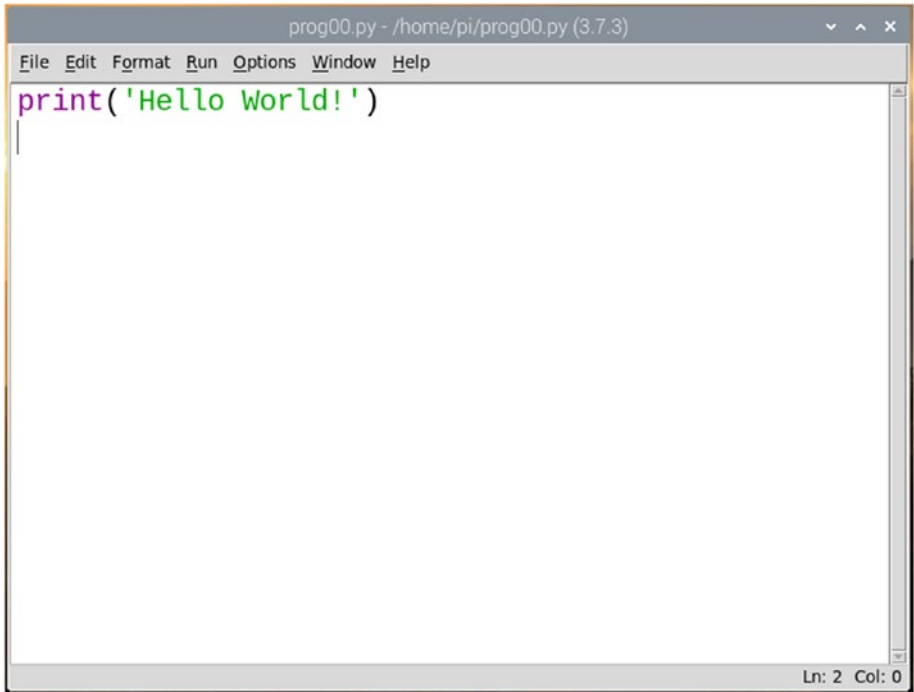


Figure 8-7. A Python program in the IDLE code editor

In the menu, click **Run ▶ Run Module**. It will execute the program on the IDLE's prompt as shown in Figure 8-8.

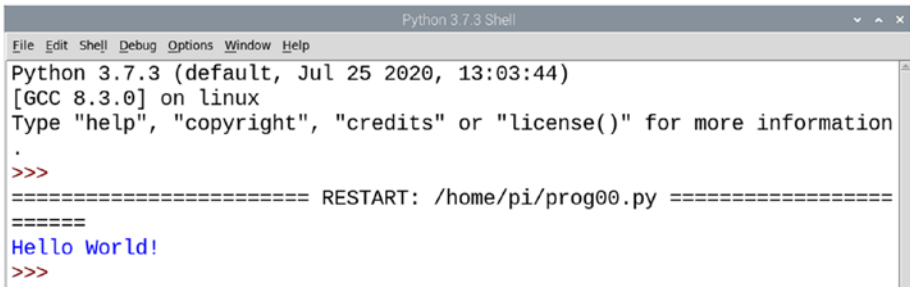


Figure 8-8. A Python program under execution in the IDLE prompt

We can even launch the program with Python's interpreter in the command prompt of the OS too. Open the command prompt of the OS and navigate to the directory where the program is stored.

In the Linux terminal, we must run the following command in the command prompt:

```
python3 prog00.py
```

Then the interpreter will run the program in the command prompt, and the output (if any) will appear there.

In Linux, there is another way we can run the program without explicitly using the interpreter. We must add the **shebang** line to the beginning of the code file. For example, our code file looks like the following:

```
#!/usr/bin/python3  
print('Hello World!')
```

The first line is known as the **shebang** line. It tells the shell what interpreter to use and its location. Then run the following command to change the file permissions to make it executable for the owner as follows:

```
chmod 755 prog00.py
```

Then we can directly launch our Python program file like any other executable with `./` as follows:

```
./prog00.py
```

The shell will execute the program and print the output in the terminal. Note that this is applicable only for Unix-like systems as they support executing programs with **shebang**.

Summary

In this chapter, we got started with C and C++ programming on Linux. Then we explored the basics of the Python programming language. We learned how to write basic Python programs and how to execute them in various ways. We also learned various modes of the Python programming language and how to launch it from the command prompt.

Now we are very comfortable with high-level programming languages on the RPi OS (and Debian derivatives). We can write and execute programs with C, C++, and Python 3. You may want to explore Java programming on the Raspberry Pi OS. The RPi OS comes with Java and related IDEs preinstalled.

In the next chapter, we will continue our wonderful journey of programming with Python 3 and write programs for GPIO (General-Purpose Input/Output) programming. We will use LEDs, resistors, breadboards, and other electronic components for that. We will also have a brief introduction to various types of buses in digital electronics.

CHAPTER 9

Programming with RPi GPIO

In the last chapter, we got introduced to high-level programming languages on Unix-like platforms. We learned to write programs with C and C++ with the GCC compiler. We also learned how to write and execute programs in the Python 3 programming language.

Continuing where we left off at the last chapter, in this chapter, we will explore GPIO programming with RPi and Python 3. The following is the list of topics we will explore in this chapter:

- GPIO pins
- Programming with GPIO

We will be comfortable with GPIO programming and usage of basic electronic components with Raspberry Pi after this chapter.

General-Purpose Input/Output Pins

The RPi board has General-Purpose Input/Output header pins. All the versions of the RPi board have this feature. This feature sets single-board computers apart from other small computers. The GPIO pins give SBCs the ability to directly interface with low-level electronic components and various data transfer buses.

I am using an RPi 4 B board with 8 GB RAM for this chapter. It is the most recent board in the RPi family. We can see the meanings of pins on any RPi board by running the following RPi OS-specific command:

```
pinout
```

It will show us the layout of the board as shown in Figure 9-1.

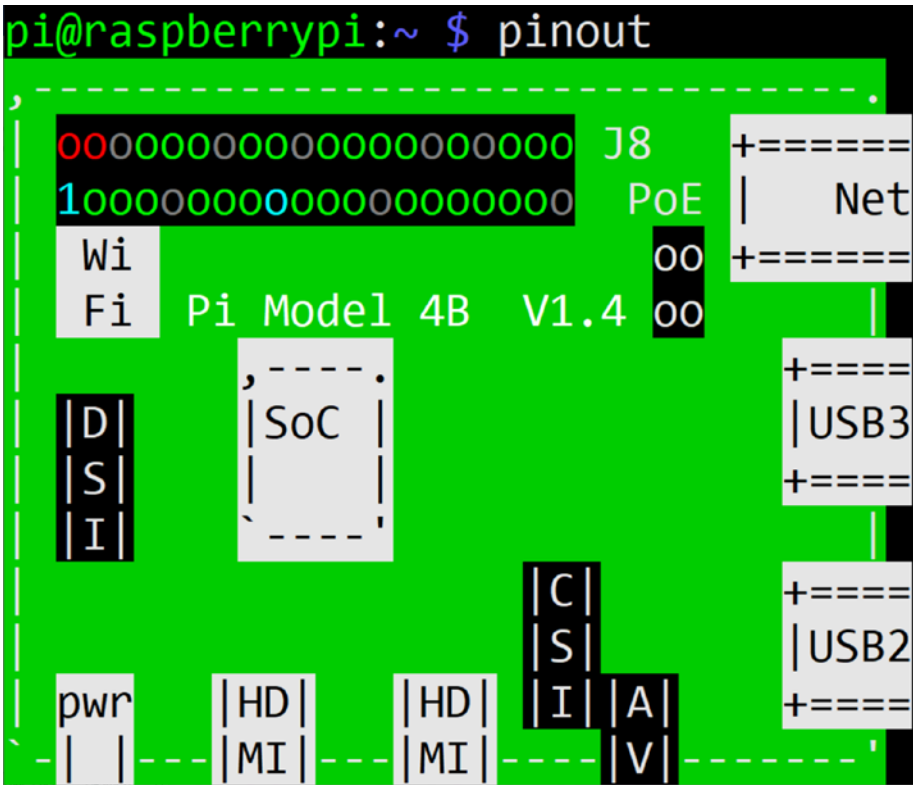


Figure 9-1. Board layout of RPi 4 B

In the top-left part of the figure, we can see 40 GPIO pins. A few earlier models of RPi had 26 pins. But they have long been out of production, so we will not discuss them here. The programs we will demonstrate in this

chapter are compatible with all the models of Pi. Figure 9-1 is the first part of the output shown on the command line. If we scroll down, we can see more. The last part of the output shows us the meanings of the pins as shown in Figure 9-2.

```

3V3 (1) (2) 5V
GPIO2 (3) (4) 5V
GPIO3 (5) (6) GND
GPIO4 (7) (8) GPIO14
GND (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
GND (25) (26) GPIO7
GPIO0 (27) (28) GPIO1
GPIO5 (29) (30) GND
GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
GND (39) (40) GPIO21

```

Figure 9-2. Pinout of RPi 4 B

This output shows us the power pins (5V, 3V3, and GND) and digital IO pins. GND stands for ground and 3V3 means 3.3 volts. We can see that there are two numbering schemes shown in the output. The physical pin numbers (also known as board pin numbers) are mentioned in the brackets, and BCM names are mentioned outside. For our convenience, we will use board (physical) pin numbering in the programs that we will write.

We can get an idea about the orientation of the pins by comparing both Figures 9-1 and 9-2. The pins are color-coded in both the figures.

Now, take a LED, a resistor, and a few jumper cables and prepare the circuit shown in Figure 9-3.

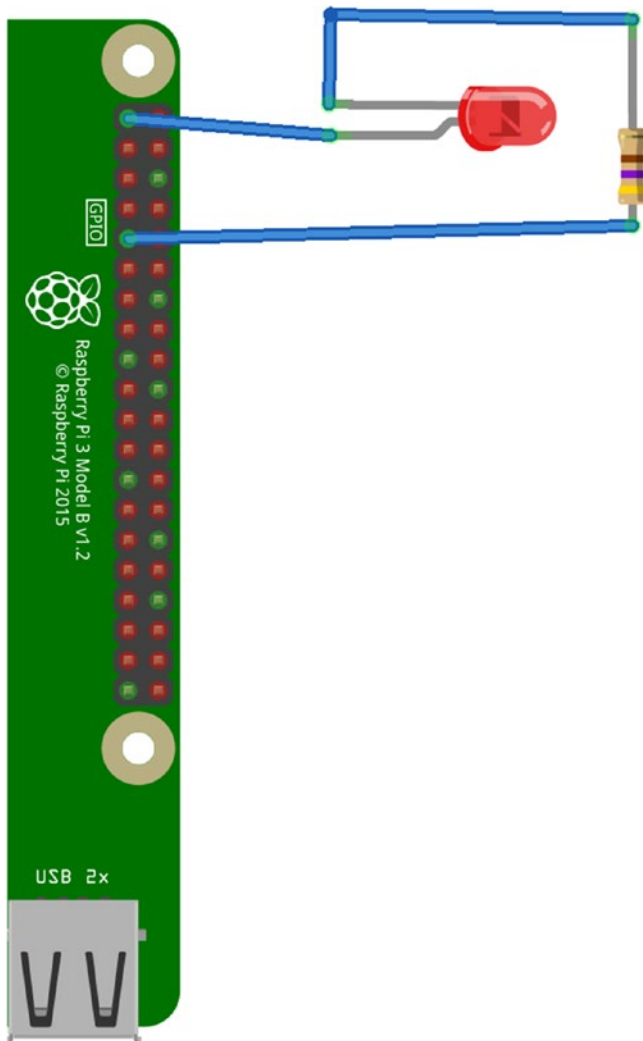


Figure 9-3. Simple LED circuit

The LED will glow as long as the RPi board is on. This is because we are connecting the anode of the LED (the longer pin) to the 3V3 pin and the cathode to a GND pin through a resistor of 470 Ohms. This is the simplest LED circuit we can make with this. You may want to try to connect the

anode of the LED to the 5V pin, and it will glow more. We have chosen the resistor with the appropriate value so it will not burn the LED. In the next section, we will see how we can write programs with GPIO.

Programming with GPIO

In this section, we will see simple circuits with LEDs and pushbuttons. We will use the Python 3 GPIO library for that. It comes preinstalled with the RPi OS. If it is not preinstalled, then install it with the following command:

```
sudo apt-get install python3-rpi.gpio -y
```

Prepare a circuit as shown in [Figure 9-4](#).

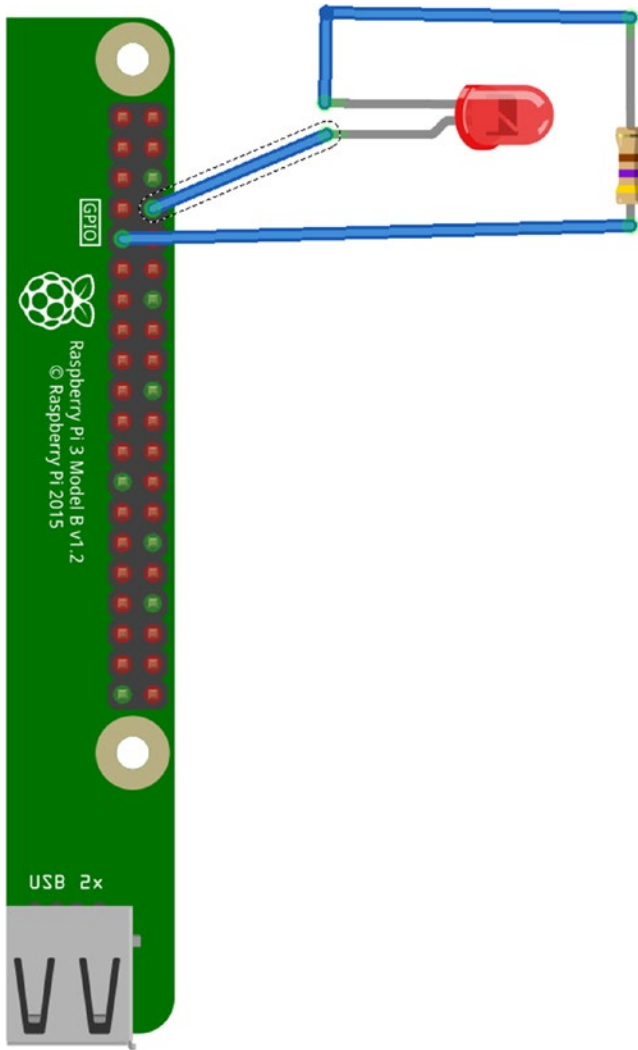


Figure 9-4. Programmable LED circuit

We are connecting the anode of the LED to the pin which is physically numbered as 8 in Figure 9-2. That is the only change we have from the earlier circuit. Check out Listing 9-1.

Listing 9-1. LED_Blink.py

```
from time import sleep
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW)

while True:
    GPIO.output(8, GPIO.HIGH)
    sleep(1)
    GPIO.output(8, GPIO.LOW)
    sleep(1)
```

Let us discuss it in detail. The first two lines import required libraries. Then we are instructing the RPi board to use the board (also called as, as we have seen earlier, physical) pin numbering with the statement `GPIO.setmode()`. Then we are disabling warnings. We are using the function `GPIO.setup()` to set board pin 8 to output mode. We are also setting its initial state as LOW. Then in an indefinite loop, we are alternately sending HIGH and LOW signals to pin 8. The call to the function `sleep()` adds an interval of 1 second between the statements. When pin 8 is HIGH, the LED is on; and when pin 8 is LOW, the LED is off. Run the script with the following command:

```
python3 LED_Blink.py
```

The LED will start blinking. To terminate the program, press Ctrl+C on the keyboard.

This is the basic GPIO programming. We can use this creatively to create various patterns of blinking LEDs if we use a breadboard to connect multiple LEDs to the digital GPIO pins.

Summary

We have continued our journey of Python 3 programming in this chapter and studied a program that makes a LED blink. We have also learned how to build a basic circuit with a LED and a resistor. Raspberry Pi has many more things to offer through its GPIO programming. You can explore this vast topic further at your own convenience.

In the next chapter, we will study the GUI of the RPi OS in detail. We will also see how to install various desktop environments for the RPi OS in detail.

CHAPTER 10

Explore the RPi OS GUI

In the last chapter, we explored Raspberry Pi GPIO in detail. We wrote programs in Python to demonstrate the GPIO functionality of RPi. We also had a brief overview of other types of buses in RPi.

This chapter will change the direction of our focus back to the software part of the RPi OS. In this chapter, we will explore various desktop environments and GUI utilities on the RPi OS. The following is the list of topics covered in this chapter:

- GUI Utilities on the RPi OS
- Other Desktop Environments

After this chapter, we will be very comfortable various GUI environments and utilities on the RPi OS.

GUI Utilities on the RPi OS

There are many GUI utilities on the RPi OS. They are grouped as per their usage. Figure 10-1 shows the RPi OS menu.

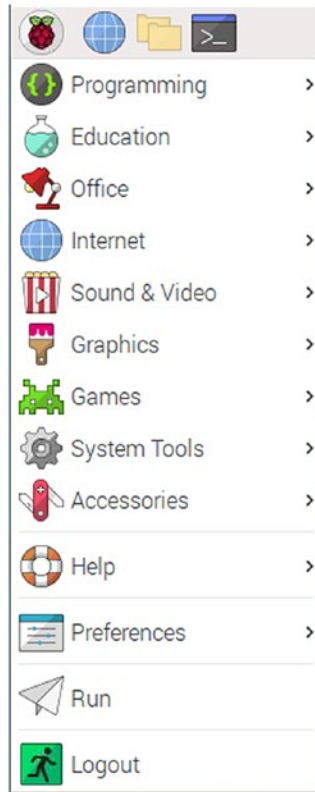


Figure 10-1. RPi OS menu

Let us have an overview of the menu items. In the **Programming** section, all the utilities for programming can be found. The RPi OS comes with the IDEs for Java (BlueJ and Greenfoot), Python (Thonny, Geany, Mu), Scratch, Wolfram, and Mathematica. We have also seen how to install IDLE for Python 3 earlier. You can find that here. There are interesting utilities like Node-RED, Sense HAT Emulator, and Sonic Pi. It is worth exploring them.

The **Education** section has SmartSim which is a software for learning digital electronics.

The **Office** section is very handy for people who want to use office applications like a word processor and spreadsheet.

The **Internet** section has a web browser, VNC, and an email client.

The **Sound & Video** section has the VLC media player. The **Graphics** section has an image viewer. You also may want to play a few games in the **Games** section.

The next four sections have various tools to manage the RPi and a few useful programs. I recommend exploring them on your own. The **Run** section, when clicked, brings up a window to launch the programs by typing in their names. We had seen the demonstration in Chapter 1. If you know the names of the programs, they can be launched from here by typing in their names. We had also seen the last option **Logout** already.

This is all about the modified LXDE desktop environment. In the next section, we will explore other desktop environments for the RPi OS.

Other Desktop Environments

Let us install other desktop environments one by one.

XFCE

Installing this one is easy. Just run the following commands one by one in sequence:

```
sudo apt-get update --fix-missing
sudo apt-get install xfce4 -y
sudo apt-get install slim -y
```

Now, when we reboot, we are presented with a screen as shown in Figure 10-2.



Figure 10-2. Login screen

We need to press the F1 key on the keyboard to choose the desktop environment. If you press F1 once, it shows **Session: Default Xsession** as shown in Figure 10-2. This is the default modified LXDE that comes with the RPi OS. We have seen this already.

The next option comes up here when we press F1 again. It reads **Xfce Session**. If we press F1 again, we see **Session: Openbox**; and after pressing F1 one more time, it shows **LXDE**. Now if we press F1 again, it cycles us back through the first option. So choose the **Xfce Session**, and key in the username and password (I have not changed the default username/password combination of **pi/raspberry**).

After login, it shows a nice XFCE desktop like the one shown in Figure 10-3.

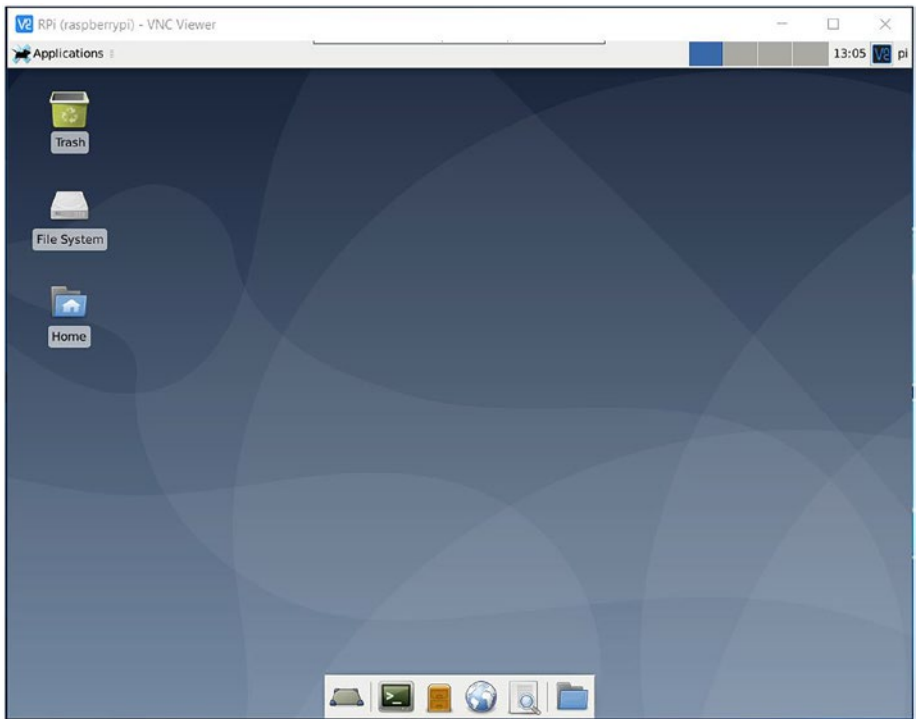


Figure 10-3. XFCE desktop

You can explore it. It is very much user-friendly.

Let us reboot and this time choose the option **Session: Openbox**. Log in into that, and you will find a blank desktop environment. This is because Openbox is a free, stacking window manager for the X Window System. We can right-click and see menu options. You can explore this style of desktop further. You can see the option to exit in the menu shown after right-clicking. Choose that to come back to the login screen. And now choose **Session: LXDE**. It will also show a very nice, pleasant, and visually appealing desktop environment as shown in Figure 10-4.

This is the unmodified LXDE desktop environment. The menu is not labeled and can be difficult to find at first. It is at the bottom-left corner as shown in Figure 10-4.

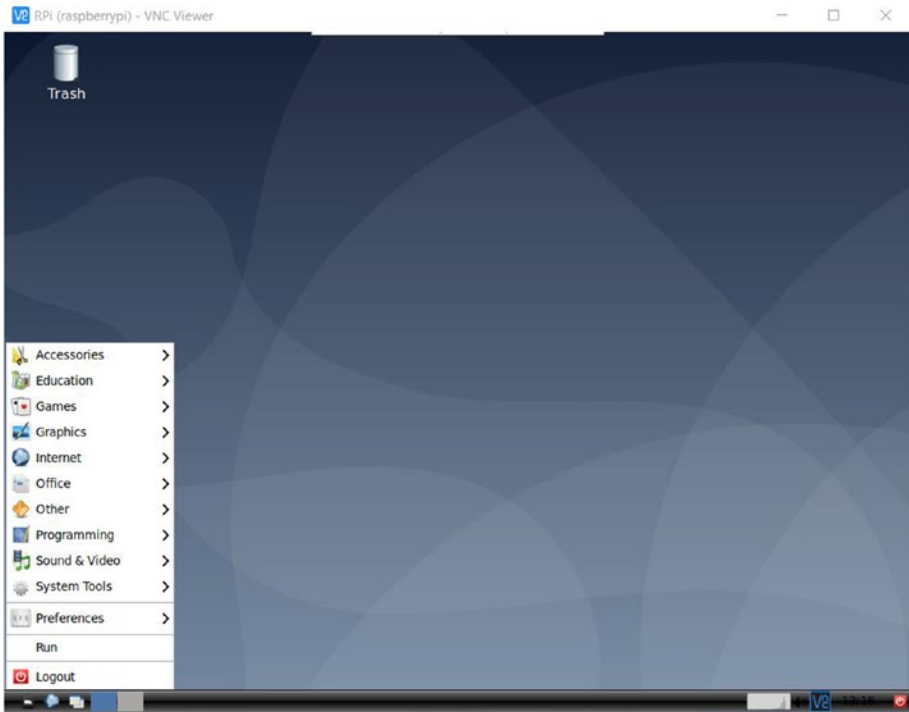


Figure 10-4. LXDE desktop

KDE Plasma

KDE is an international community for developing free software for Unix-like operating systems. One of the software developed by KDE is the KDE Plasma desktop environment. Let us install it on our Pi. Open the terminal program of your current desktop environment and run the following commands:

```
sudo apt-get update --fix-missing
sudo apt-get dist-upgrade -y
sudo apt-get install kde-full -y
```

We can see the other installation options at <https://wiki.debian.org/KDE>. During the installation, you will be prompted to choose a desktop environment. Choose **slim**. Once the installation is completed, reboot the system. We will find the **KDE Plasma** as an option in the login screen (Figure 10-5).

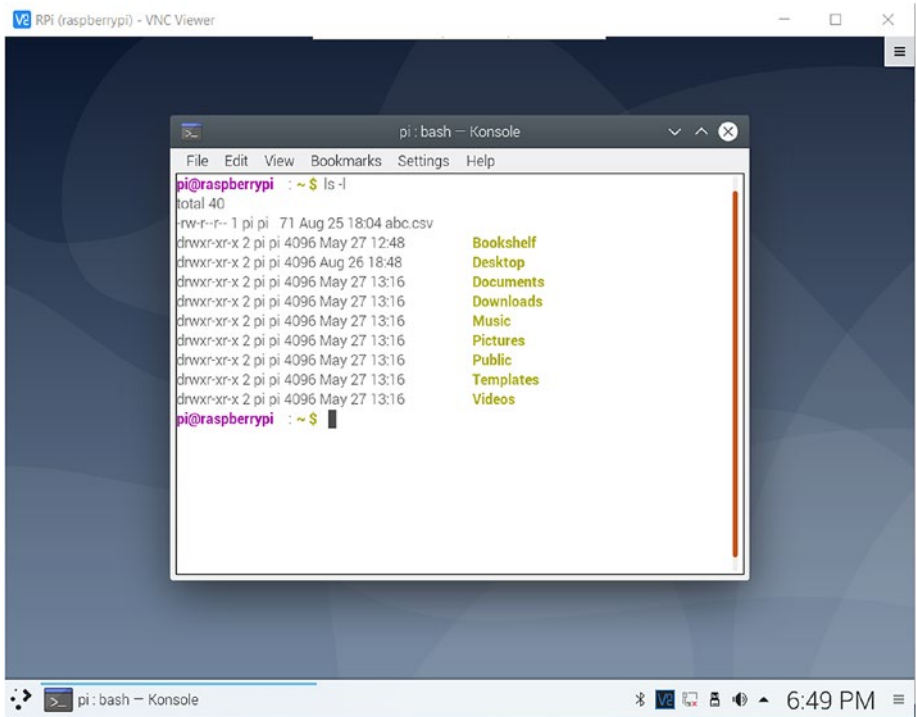


Figure 10-5. KDE Plasma desktop

KDE Plasma, XFCE, and LXDE environments will come with their own set of utilities. If you choose any of the desktop environments, you will find all the utilities listed under the menu. Feel free to explore the utilities on your own.

Summary

In this chapter, we learned various important utilities. We also learned to install other desktop environments on the RPi OS for RPi. For practice, as suggested earlier, work with all the desktop environments and explore the utilities on your own.

Next, we will have the Appendix of this book, which contains assorted topics with tips and tricks related to the Raspberry Pi OS.

APPENDIX

Additional Tools

In the last chapter, we explored various GUI tools that come with the RPi OS and other desktop environments. In this Appendix, we'll learn about Raspberry Pi Imager and other related topics.

Raspberry Pi Imager

Raspberry Pi Imager is a versatile software. We can use it for other purposes too. If you read a RPi OS-formatted card with Windows or any other OS, then you will find that it has two partitions. The first partition is the **boot** partition (it is 256 MB usually) which stores the `config.txt` file, and the other partition has the rest of the Linux filesystem. We cannot use the card for other uses (Android phone or video camera) with the RPi OS or any other OS for SBCs on it. For that, we have to format the card. RPi Imager provides an option for that. When we click **CHOOSE OS**, we can see different options as shown in Figure A-1.

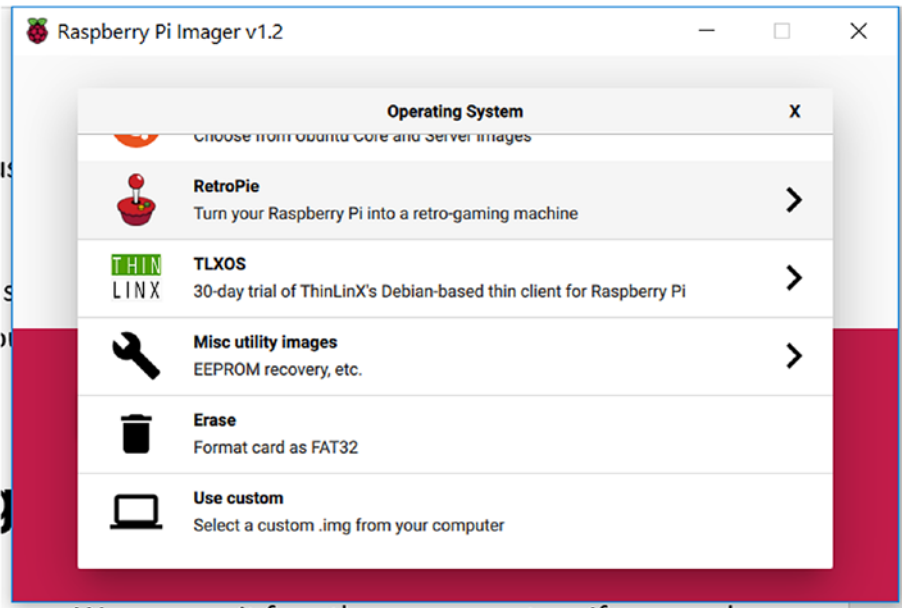


Figure A-1. Erase option

Choose the option **Erase**. Then choose the SD card in the main screen and finally click the button **WRITE**. It will format the card as a FAT32 storage device. Now, we can reuse it for any other purpose.

Additional Utilities

We can also use utilities like the built-in Windows Disk Management or SD Card Formatter (www.sdcard.org/downloads/formatter/) to erase the partitions made by RPi OS.

We can also write operating systems other than the RPi OS to the microSD card. We can see the list of other operating systems in the same menu (CHOOSE OS). Raspberry Pi supports more operating systems than the one mentioned here.

We can download the OS images from www.raspberrypi.org/downloads/ and then choose the option **Use custom** to write the image files to the microSD card. These image files have the ***.img** extension.

Manjaro Linux

We can install Manjaro Linux with the KDE Plasma desktop from <https://manjaro.org/downloads/arm/raspberry-pi-4/arm8-raspberry-pi-4-kde-plasma/>.

FreeBSD

We can also install FreeBSD from <https://wiki.freebsd.org/action/show/arm/Raspberry%20Pi>.

Additional OSs

Many other operating system projects create images for RPi. Most of these images will be zipped and can be extracted using a free utility called 7-Zip.

Many of these OS projects come with torrent files, and we can download the image files by opening these torrent files with torrent software like BitTorrent or uTorrent. We can also directly download the image file, but I usually download it with a download manager like Download Accelerator Plus (www.speedbit.com/). In case Internet connectivity is lost, torrent and download managers save the checkpoint, so we can resume the download when the connectivity is restored.

Index

A

Advanced Package Tool (APT), 55
Absolute path, 54–56, 73

B

Bash shell, 38, 82, 96
Basic Input/Output
 System (BIOS), 16
BitTorrent/uTorrent, 153
Boot partition, 151

C

cd command, 56
Centrum Wiskunde &
 Informatica (CWI), 120
command history, 78, 79
Command-Line Interface (CLI), 36
Command prompt, 38, 39
Comma-separated value (CSV), 74
Control operators, 75
Crontab, 114, 115

D

Debian, 6, 35, 55, 67
df command, 84

Directories, 41, 42

 case sensitive, 64, 65
 create and
 remove, 62–64

Dynamic Host Configuration
 Protocol (DHCP), 29

E

echo command, 75

F

Filename globbing, 77
File operations, 73
File Transfer Window, 46
FreeBSD, 153

G

General Purpose Input
 Output (GPIO), 131
GPIO.setmode(), 140
GPIO.setup(), 140
Graphical User Interface
 (GUI), 36

H

Hidden items, [60](#)

High-level programming languages

 C and C++, [118](#), [119](#)

 python++, [120](#)

I, J, K

I/O redirection

 LUnix-like operating systems, [111](#)

 stderr, [113](#)

 stdin, [112](#)

 stdout, [112](#)

Is command, [57–59](#)

L

Lightweight X11 Desktop

 Environment (LXDE), [36](#)

Leafpad, [62](#)

Linux and distributions, [5](#), [6](#)

LINUX commands, [83–86](#)

Linux filesystem, [39](#)

M

Manjaro Linux, [153](#)

microSD card, [152](#)

N

nano text editor, [61](#)

Network-related commands, [70](#), [71](#)

nohup, [94](#)

O

Operating system shell, [36](#)

OS setup

 HDMI converters, [12](#)

 HDMI male pin, [11](#)

 microSD card, [10](#)

 preparing SD card

 boot, [16](#)

 changes, [16](#), [17](#)

 CHOOSE OS, [14](#), [15](#)

 config.txt., [16](#)

 Raspberry Pi Imager, [14](#)

 writing OS, [16](#)

 RPi board, [17](#), [18](#)

 SD card converter, [10](#)

 SD card reader, [13](#)

 types of HDMI male pin, [11](#), [12](#)

 USB-C male pin, [7](#), [8](#)

 USB keyboard, [8](#), [9](#)

 USB OTG converter, [9](#)

 VGA monitor, [13](#)

P, Q

Present working directory (pwd),

[54](#), [55](#)

ping command, [72](#)

Piping, [78](#)

Process ID (PID), [76](#)

Python

 applications, [122](#)

 definition, [120](#)

 enhancement proposals, [121](#)

 history, [120](#)

- IDLE, [124–126](#)
- interactive mode, [127, 128](#)
- philosophy, [121, 122](#)
- script mode, [128–130](#)
- Python 3, Debian derivatives, [123](#)
- Python Enhancement
 - Proposals (PEPs), [121](#)

R

- Raspberry Pi
 - 4 Model B
 - components, [4, 5](#)
 - photograph, [4](#)
 - specifications, [3](#)
 - OS, [6](#)
- Raspberry Pi Imager, [151](#)
- Raspberry Pi OS, [36](#)
 - Bluetooth symbol, [37](#)
 - desktop, [37](#)
 - File Explorer, [37](#)
 - lxterminal, [37](#)
 - WiFi symbol, [37](#)
- raspi-config utility, [68](#)
- Relative path, [54](#)
- Remote desktop, [50](#)
- rm command, [63](#)
- RPi board
 - booting up, [17, 18](#)
 - configuration
 - change password, [20](#)
 - country/language, [19](#)
 - interfaces, [25, 26](#)
 - localisation, [27, 28](#)
 - message, [29](#)
 - OS menu, [23, 24](#)
 - performance, [26, 27](#)
 - set up completion, [23](#)
 - set up screen, [20, 21](#)
 - shutdown options, [28](#)
 - update software, [22](#)
 - WiFi, [21, 22](#)
 - window, [18, 19, 24, 25](#)
- internet
 - accessories, [30](#)
 - active client list, [33](#)
 - backup, [32](#)
 - ifconfig command, [33](#)
 - Linux commands, [32](#)
 - lxterminal utility, [30](#)
 - lxterminal screenshot, [31](#)
 - nano editor, [32](#)
 - network interfaces, [32](#)
 - network-related
 - information, [32](#)
 - plaintext editor, [32](#)
 - restart, [33](#)
 - Run window, [30, 31](#)
 - USB OTG cable, [30](#)
 - USB WiFi dongle, [29, 30](#)
- RPi GPIO
 - pins, [133, 135–137](#)
 - programming, [138–140](#)
- RPi OS GUI
 - desktop environment
 - KDE plasma, [148, 149](#)
 - XFCE, [145–148](#)
 - utilities, [143, 144](#)

INDEX

S

- shebang or shebang, 96
- Shell/environment variables, 81, 82
- Shell scripts
 - command-line
 - arguments, 103
 - compare strings, 106–108
 - expressions, 97, 98
 - file operations, 108–110
 - functions, 103, 104
 - if statement, 99–101
 - loops, 104–106
 - lterminal, 95
 - nohup, 94, 95
 - switch case, 101
 - Unix file permissions, 92–94
 - user input, 96
 - variable length, 102
- Single-board computers (SBCs), 2
- SSH connection window, 44, 45
- sudo command, 69
- Superuser, 69

T

- touch command, 60
- tree command, 56
- tree utility, 55

U

- Unix commands, 57
- Unix-like operating systems, 40
- UNIX tools, 86–88

V

- vim editor, 61
- VNC viewer connection
 - window, 48
- VNC viewer window, 47

W, X, Y, Z

- wget command, 72