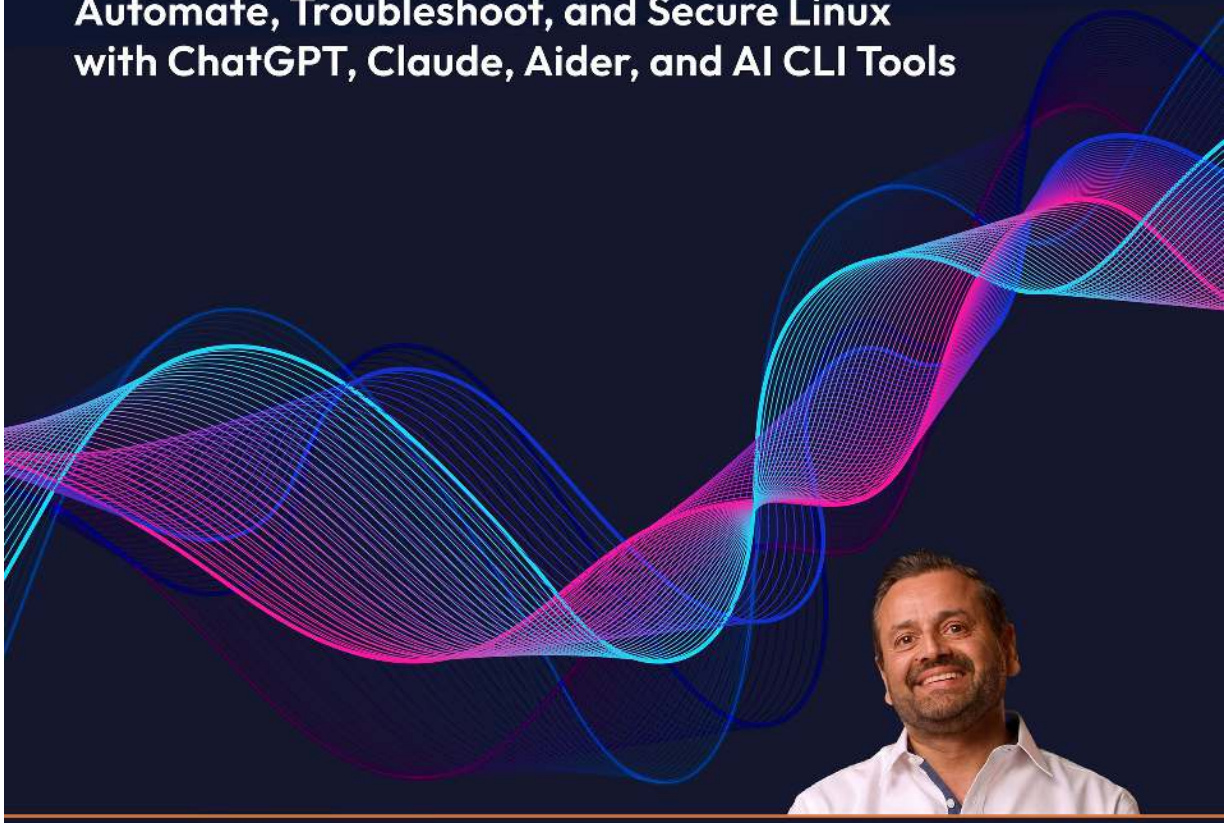
EXPERT INSIGHT

Early Access

Agentic Linux Administration

Automate, Troubleshoot, and Secure Linux with ChatGPT, Claude, Aider, and AI CLI Tools



Imran Afzal

packt

Agentic Linux Administration

Automate, Troubleshoot, and Secure Linux with ChatGPT, Claude, Aider, and AI CLI Tools

Imran Afzal

Agentic Linux Administration

Copyright © 2026 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot

guarantee the accuracy of this information.

Early Access Publication: Agentic Linux Administration

Early Access Production reference: B41362

Published by Packt Publishing Ltd.
Grosvenor House
11 St Paul's Square
Birmingham
B3 1RB, UK.
ISBN 978-1-80872-479-4

www.packtpub.com

Table of Contents

Welcome to Packt Early Access

Agentic Linux Administration: Automate, Troubleshoot, and Secure Linux with ChatGPT, Claude, Aider, and AI CLI Tools

Chapter 1: Foundations of AI-Driven Linux System Management

Understanding AI and Linux

How AI can help system administrators

How AI is transforming IT and DevOps

- Netflix •
- Amazon •
- Google •

Summary

Knowledge check

- Answers •

Chapter 2: Preparing Your System Workspace

Technical requirements

Installing VirtualBox

Creating and installing the CentOS virtual machine

- Downloading the CentOS Stream ISO •
- Creating the virtual machine •
- Configuring the virtual machine settings •
- Installing CentOS Stream •

Connecting through PuTTY

Exploring the Linux environment
Checking the user and system identity •
Exploring the Linux filesystem •
Navigating between directories •
Creating and reading a file •
Checking basic internet connectivity •
Reading system logs •
Working with elevated privileges •
Inspecting and modifying permissions •
Verifying network connectivity
Testing DNS resolution •
Restarting and checking NetworkManager •
Reviewing AI tools for system administrators
Summary
Knowledge check
Answers •

Chapter 3: Using ChatGPT/Claude from the Linux Command Line

Technical requirements
Understanding ChatGPT CLI
Why a command-line AI workflow is useful •
Installing Python, pip, and the OpenAI SDK
Creating an OpenAI API key
Exporting the API key in Linux •
For those using Claude instead of ChatGPT •
Reviewing the OpenAI Python example •
Sending the first request from the terminal

Making the program interactive
Summary
Knowledge check
Answers •

Chapter 4: Using ChatGPT CLI for Linux Administration

Technical requirements
Generating a system-health script
Troubleshooting a permission error
Creating and scheduling a disk-monitoring job
Interpreting system logs with AI
Reviewing security-hardening recommendations
Trying one focused control with AIDE •
Creating a user-account script safely
Summary
Knowledge check
Answers •

Chapter 5: Generating Linux Commands with Shell Genie

Technical requirements
Understanding Shell Genie
How Shell Genie complements ChatGPT CLI •
Installing and configuring Shell Genie on CentOS
Checking Python and pip •
Installing pipx and Shell Genie •
Resolving the Click compatibility error •
Selecting the backend and saving the configuration •

Testing Shell Genie by installing Apache •
Generating commands from English
Compressing the Documents directory •
Copying an archive before deleting the original •
Changing the user password •
Creating scripts and interacting with the system
Creating and running an EPEL installation script •
Creating and running a system-check script •
Summary
Knowledge check
Answers •

Chapter 6: Using Aider for Bash Scripting and Linux Automation

Technical requirements
Understanding Aider and when to use it
Using Aider with Git •
Working without Git •
Installing and configuring Aider
Installing Aider with pipx •
Providing the OpenAI API key •
Starting Aider without Git •
Testing the connection •
Creating and running a Bash snippet
Approving the file and shell commands •
Generating loops, cron jobs, and script templates
Verifying the cron entry and script •
Summary

Knowledge check

Answers •

Chapter 7: Bringing It All Together

Technical requirements

Recapping the AI-assisted Linux toolkit

ChatGPT CLI as the central assistant •

Shell Genie for quick commands •

Aider for working inside scripts •

Mini project: Creating and scheduling a user-audit script

Starting ChatGPT CLI as root •

Reviewing the generated script •

Creating user_audit.sh •

Scheduling the script •

Verifying the cron output •

Removing the every-minute test schedule •

Summary

Knowledge check

Answers •

Welcome to Packt Early Access

Agentic Linux Administration: Automate, Troubleshoot, and Secure Linux with ChatGPT, Claude, Aider, and AI CLI Tools

We're giving you an exclusive preview of this book before it goes on sale. It can take many months to write a book, but our authors have cutting-edge information to share with you today. Early Access gives you an insight into the latest developments by making chapter drafts available. The chapters may be a little rough around the edges right now, but our authors will update them over time.

You can dip in and out of this book or follow along from start to finish; Early Access is designed to be flexible. We hope you enjoy getting to know more about the process of writing a Packt book.

1. Chapter 1: Foundations of AI-Driven Linux System Management
2. Chapter 2: Preparing Your System Workspace
3. Chapter 3: Using ChatGPT/Claude from the Linux Command Line
4. Chapter 4: Using ChatGPT CLI for Linux Administration
5. Chapter 5: Generating Linux Commands with Shell Genie
6. Chapter 6: Using Aider for Bash Scripting and Linux Automation
7. Chapter 7: Bringing It All Together

1

Foundations of AI-Driven Linux System Management

When you first come across terms such as **artificial intelligence** (**AI**) and Linux, they can sound like two large and complicated subjects. If you are new to technology, don't worry. We are going to keep things simple and build the foundation step by step.

We will first look at what Linux is, then explore what artificial intelligence means, and finally see how the two can work together. We will also examine how AI supports system administrators and how it is already changing IT and DevOps work in the real world.

In this chapter, we'll cover the following topics:

- Understanding AI and Linux
- How AI can help system administrators
- How AI is transforming IT and DevOps

Let's start with Linux.

NOTE

Download the PDF version of this book

Your purchase includes a DRM-free PDF copy of this book, the code bundle, and a range of exclusive benefits. To unlock everything, follow the *Free benefits with your book* section in the *Preface*.

Understanding AI and Linux

Linux is an operating system, just like Microsoft Windows or Apple macOS. However, it is used extensively for servers, cloud systems, security tools, data

centers, websites, and enterprise IT systems.

Many organizations choose Linux because it is stable, fast, powerful, flexible, and free to use. It gives administrators a great deal of control over how a system is configured and managed.

Linux is also open source, which means that its design and source code are publicly available. A large global community helps keep it updated, secure, and reliable. Linux is known for running smoothly even when it is handling heavy workloads.

The best part is that once you learn Linux, you can apply those skills to many different systems and working environments.

Now, let's talk about **AI**, which stands for **artificial intelligence**.

AI is a technology that helps computers process information, identify patterns, and make decisions in intelligent ways. You may already interact with AI without realizing it. For example, AI may be involved when a phone suggests a reply, Google Maps recommends a route, or a website recommends products.

AI learns patterns from data and uses that knowledge to help solve problems. When we bring AI and Linux together, something very powerful happens. Linux manages the underlying system, while AI helps make working with that system easier and smarter.

Instead of manually typing long commands, interpreting complex error messages, or searching through large amounts of documentation, you can use tools such as ChatGPT CLI, Shell Genie, and Aider to help understand and resolve problems.

AI can help you create scripts, automate tasks, analyze logs, improve security, and receive recommendations in clear language. This means that you do not need to remember every Linux command before beginning to perform useful work.

In this relationship, AI becomes your assistant in Linux. For example, if a Linux service is not working, you can ask an AI tool to help identify the issue and explain it in understandable terms.

The combination of AI and Linux is therefore not just about learning commands. It is about working smarter, faster, and with greater confidence. If you are not from an IT background, AI can help guide you through unfamiliar tasks one step at a time.

So, take a deep breath and believe in yourself. We will build these skills gradually and turn what may initially look complicated into something simple, practical, and meaningful for your future.

How AI can help system administrators

Before looking at how AI can help, we first need to understand the role of a system administrator.

A system administrator, often shortened to system admin or sysadmin, manages an organization's computer systems and helps ensure that they continue to operate smoothly.

Common system administration responsibilities include:

- Creating user accounts
- Assigning permissions
- Installing software
- Updating systems
- Checking available storage
- Monitoring logs for problems
- Managing backups
- Restarting and maintaining services
- Helping keep systems secure

These tasks may sound straightforward, but as a company grows, many of them must be repeated again and again. This can take a considerable amount of time. This is where AI becomes helpful.

For example, imagine that a system administrator needs to create five new users with similar permissions. Instead of creating every user individually, the administrator can ask an AI tool to generate a script that handles all five accounts.

AI can also recommend commands for setting password policies, configuring firewall rules, or restricting access. Just as importantly, it can explain why those commands may be needed.

Troubleshooting is another area in which AI can help. Linux sometimes produces long and confusing error messages that can be difficult to interpret. Instead of immediately spending time searching online or reading lengthy documentation, a system administrator can ask an AI tool to analyze the error.

Depending on how the tool is being used, it can examine supplied system messages, logs, or terminal output and explain the issue in simpler language. It may also suggest possible solutions, commands, or next steps. In the practical examples later in this book, the relevant command output or log excerpt is supplied to the AI tool manually.

AI can also assist with documentation. Many administrators perform their tasks correctly but forget to document the steps afterward. An administrator can provide a short description of the work completed, and AI can help turn that description into a clearer and more professional record.

Now, let's talk about automation, which is one of the most useful applications of AI for system administrators.

Instead of manually completing the same tasks every day, an administrator can use AI to help create scripts that perform those tasks automatically. These tasks might include checking disk space, cleaning temporary files, verifying the status of a service, or sending alerts.

For security-related work, AI can offer suggestions for improving system safety, identifying weak points, and creating audit or compliance checklists. Rather than spending hours trying to determine the correct command or method, an administrator can ask AI for guidance and receive a response in clear language.

This can make system administration work smoother, faster, and less stressful. With AI support, even someone who is new to Linux can approach unfamiliar tasks with greater confidence.

With AI by your side, you are not alone while learning, solving problems, and developing into a stronger and more capable system administrator.

How AI is transforming IT and DevOps

AI is already helping professionals in real working environments, particularly in areas such as IT and DevOps.

DevOps is a collaborative approach in which development and operations teams work together to build, test, update, and maintain applications. Its goals include faster delivery, continuous improvement, and reliable system performance.

In the past, DevOps teams had to perform many activities manually. These included testing code, reading logs, monitoring performance, and resolving system issues. This often required significant time, effort, and attention.

Today, many companies use AI to make DevOps work easier, faster, and less stressful. AI can monitor systems, identify patterns, and help teams respond to changing conditions.

Alright, let's look at a few examples.

Netflix

Netflix (https://netflixtechblog.com/cloud-efficiency-at-netflix-f2a142955f83) provides an example of how data and AI can support large-scale services. Netflix's engineering teams use centralized platform data to understand resource inventory, ownership, usage, and cost. This data helps them identify usage and cost patterns and make informed decisions about cloud efficiency.

Looking ahead, Netflix has also described plans to use predictive analytics and machine learning to optimize usage and detect cost anomalies. In simple terms, this can help teams spot inefficient resource use or unexpected costs earlier and decide where action is needed.

By giving engineering teams clearer information about resource use, efficiency, and cost, this approach supports better decisions about how cloud infrastructure is managed. The goal is smoother, more efficient operation at scale.

Amazon

Amazon Web Services (AWS) (https://aws.amazon.com/devops-guru/features/) provides another practical example. Amazon DevOps Guru uses machine learning to analyze operational data, application metrics, and events, identify behavior that differs from normal patterns, and present recommendations for current or predicted operational issues.

AI-assisted detection does not automatically stop every problematic release. It gives administrators insights and recommendations so they can decide what action to take. For workloads with recurring demand patterns, Amazon EC2 Auto Scaling can also use predictive scaling to forecast demand and add capacity before the expected load arrives.

Increasing resources during high-traffic periods helps maintain performance. Reducing them when traffic is lower can help control costs.

Google

Google Cloud (https://docs.cloud.google.com/cloud-assist/investigations) offers similar AI-assisted operations through Gemini Cloud Assist. It can correlate logs, metrics, traces, and configuration data, summarize complex log entries, explain error messages, and provide troubleshooting recommendations.

In simple terms, AI can help operations teams identify and investigate issues more quickly. Gemini Cloud Assist can support investigations and recommend actions, but it acts only with explicit user authorization rather than resolving issues on its own.

These examples show that AI is not simply a future idea in DevOps. It is already being used to support daily IT operations.

With AI handling or assisting with repetitive tasks, DevOps teams can spend more time planning, improving systems, and creating new solutions. AI and DevOps are becoming a strong team, helping organizations deliver reliable services with fewer errors and greater confidence.

Summary

In this chapter, you learned the foundations of Linux and artificial intelligence and explored what happens when they are brought together. Linux provides a

stable, flexible, and widely used operating environment, while AI can help make working with that environment simpler and more efficient.

You also learned about the responsibilities of a system administrator and how AI can assist with scripting, automation, troubleshooting, documentation, and security-related tasks. Finally, we looked at how AI is already supporting IT and DevOps operations in large-scale environments.

Knowledge check

Test your understanding of the concepts covered in this chapter.

1. What is Linux?
    A. An AI-powered monitoring tool
    B. An operating system
    C. A programming language
    D. A cloud storage service
2. What does it mean when Linux is described as open source?
    A. It can only be used on public servers
    B. It does not require security updates
    C. Its source code is publicly available
    D. It can only be managed through the command line
3. Which of the following is a common responsibility of a system administrator?
    A. Designing marketing campaigns
    B. Creating user accounts and assigning permissions
    C. Producing product recommendations for customers
    D. Developing mobile games
4. How can AI assist with Linux troubleshooting?
    A. By preventing every possible system failure
    B. By replacing the operating system
    C. By analyzing supplied errors or logs and explaining them in simpler language
    D. By removing the need to monitor systems

5. Which task is a suitable candidate for AI-assisted automation?
   A. Checking disk space every day
   B. Choosing a company name
   C. Designing an office floor plan
   D. Conducting an employee interview
6. What is DevOps?
   A. A Linux distribution designed for developers
   B. A security tool for managing passwords
   C. A collaborative approach in which development and operations teams work together
   D. An AI model that generates Linux commands
7. What do the Netflix, Amazon, and Google examples illustrate?
   A. AI is only being considered for future IT systems
   B. AI is already being used to support large-scale IT operations
   C. AI removes the need for development and operations teams
   D. AI is used only to create user accounts

Answers

1. **b. An operating system**. Linux is an operating system used across servers, cloud environments, security tools, and many other systems.
2. **c. Its source code is publicly available**. Linux is open source, allowing a global community to examine, maintain, and improve it.
3. **b. Creating user accounts and assigning permissions**. User and permission management is one of the common responsibilities of a system administrator.
4. **c. By analyzing supplied errors or logs and explaining them in simpler language**. AI can help interpret complex information and suggest possible next steps.
5. **a. Checking disk space every day**. This is a repetitive task that can be performed automatically through a script.
6. **c. A collaborative approach in which development and operations teams work together**. DevOps helps teams build, test, update, and maintain applications more efficiently.

7. **b. AI is already being used to support large-scale IT operations**. The examples demonstrate how AI can assist with monitoring, resource management, software delivery, and troubleshooting.

2

Preparing Your System Workspace

Before working with AI tools inside Linux, we need a complete and reliable workspace. In this chapter, we will install VirtualBox on Windows, create a CentOS Stream virtual machine, explore the Linux environment, connect to it remotely through SSH, and verify that its network connection is working correctly.

We will also look at some of the AI tools available to system administrators. By the end of the chapter, the Linux environment should be ready for the practical AI-assisted tasks covered later in the book.

In this chapter, we will cover the following topics:

- Installing VirtualBox
- Creating and configuring a CentOS virtual machine
- Connecting through PuTTY
- Exploring the Linux environment
- Verifying network connectivity
- Reviewing AI tools for system administrators

Let's get started!

Technical requirements

To follow the hands-on procedures in this chapter, make sure the following are available:

- A 64-bit Windows host supported by the selected VirtualBox release, with administrator access and hardware virtualization enabled

- A stable internet connection for downloading VirtualBox, PuTTY, and the CentOS Stream installation image
- The latest available Oracle VirtualBox release for Windows hosts
- The latest available CentOS Stream x86_64 DVD ISO
- At least 4 GB of memory, one CPU core, and 30 GB of storage allocated to the virtual machine, in addition to the resources required by the Windows host
- PuTTY for SSH connection if connecting from Windows OS

Installing VirtualBox

We will begin by downloading and installing VirtualBox, the virtualization software used to create and run the Linux virtual machine.

1. Open a web browser and go to the official VirtualBox downloads page at `https://www.virtualbox.org/wiki/Downloads`. On the home page, select the download option to open the VirtualBox downloads page.

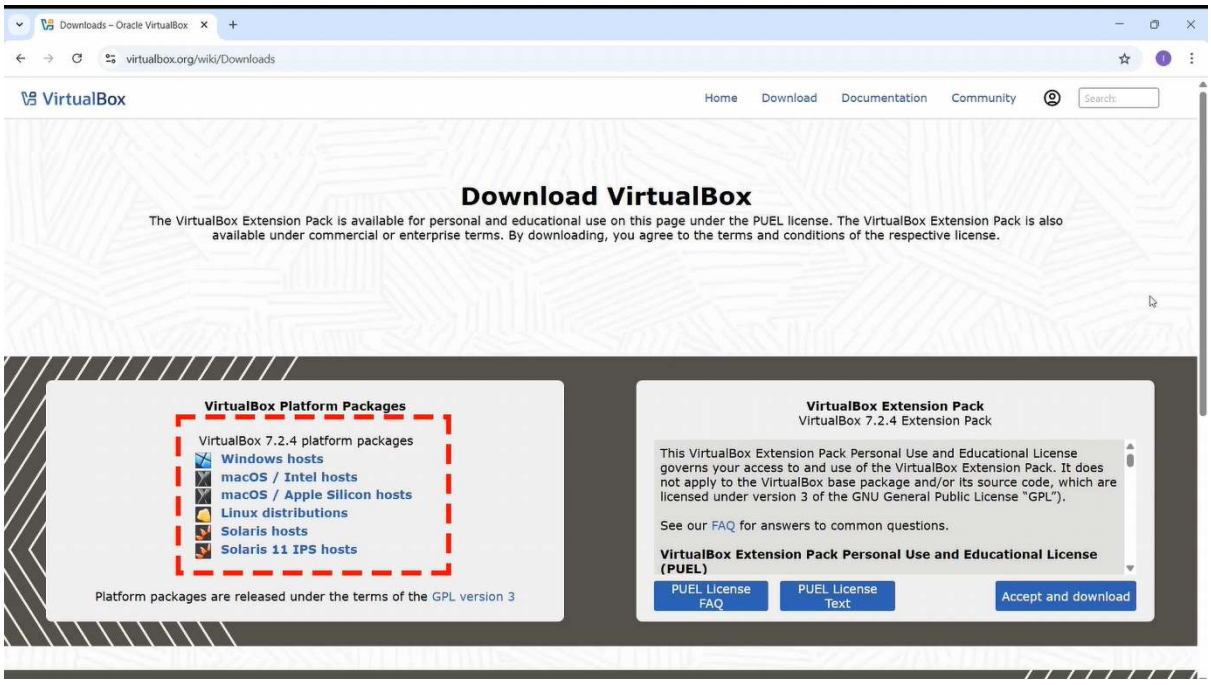


Figure 2.1: VirtualBox downloads page showing the available platform packages

2. VirtualBox provides different installation packages for each host operating system. This chapter demonstrates the Windows workflow, so select Windows hosts. The installer will begin downloading.

3. Once the download is complete, open the installer. Wait for the VirtualBox setup window to appear, and then select **Next**.
4. Accept the license agreement and select **Next** to continue.
5. The following screen presents the installation settings. Retain the default selections unless the host system requires a different configuration and select **Next** to continue.
6. The installer may warn that the network connection could be interrupted temporarily while the VirtualBox networking components are installed. This is expected during the installation. Select **Yes** to continue.

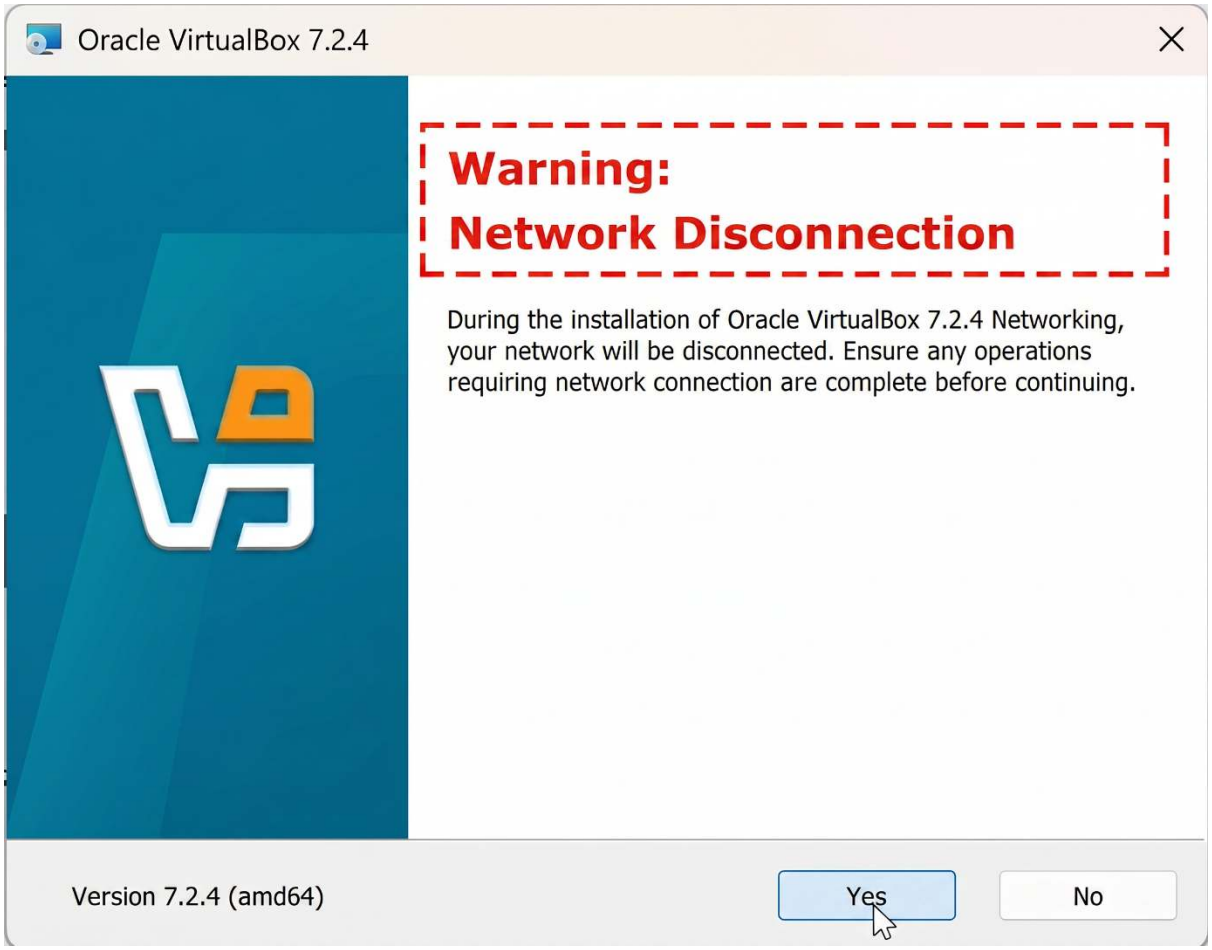


Figure 2.2: VirtualBox installer warning that the network connection may be temporarily interrupted

7. The installer then presents optional settings such as adding VirtualBox to the Start menu and creating a desktop shortcut. These options are selected by default and can be retained. Select **Next** to proceed.

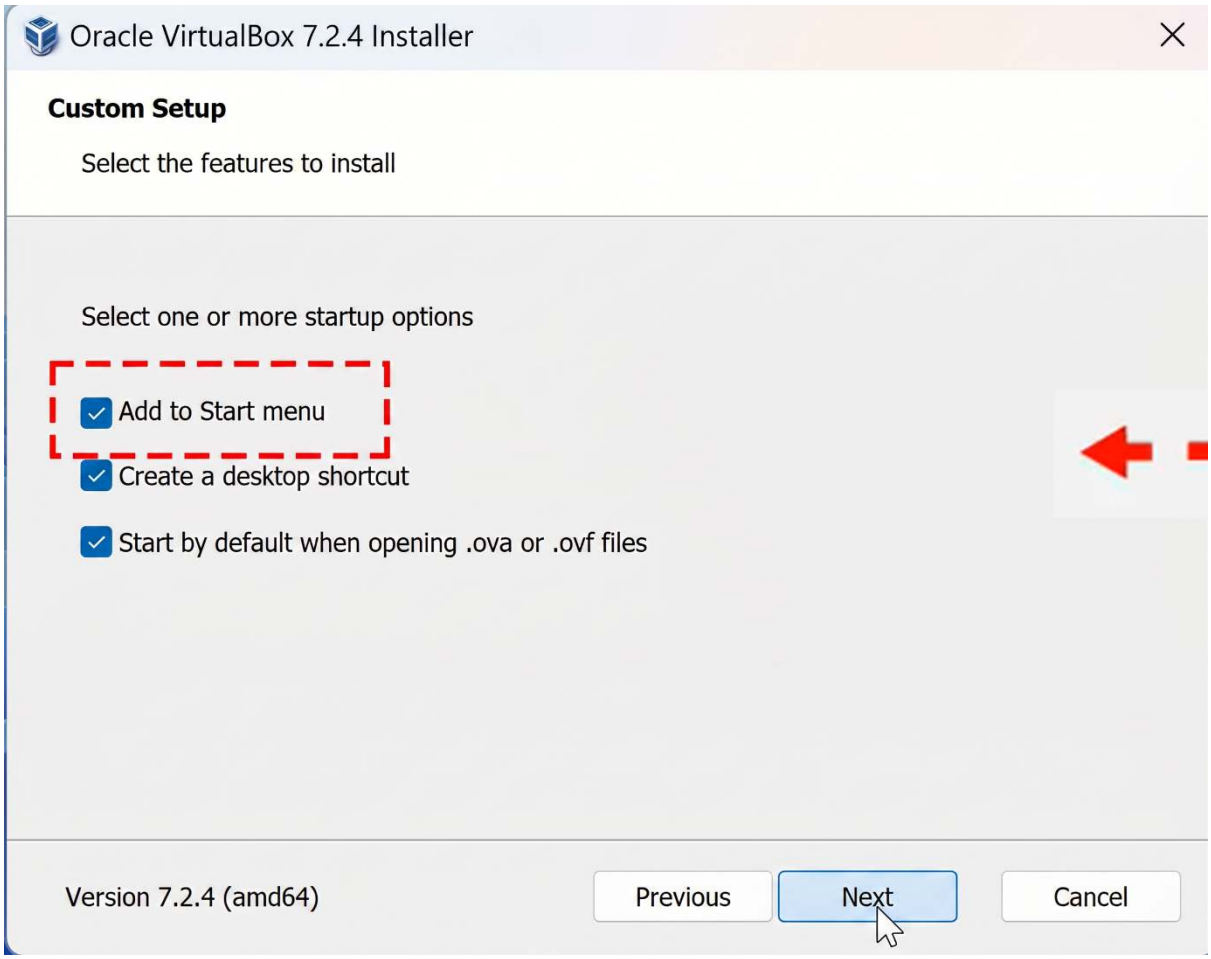


Figure 2.3: VirtualBox installer showing the default shortcut and file-association options

8. When the **Ready to Install** screen appears, select **Install** and wait for the installation to finish.
9. On the final screen, keep **Start Oracle VirtualBox** selected and click **Finish**. VirtualBox should open automatically.

The VirtualBox interface confirms that the application has been installed successfully and is ready to create the Linux virtual machine.

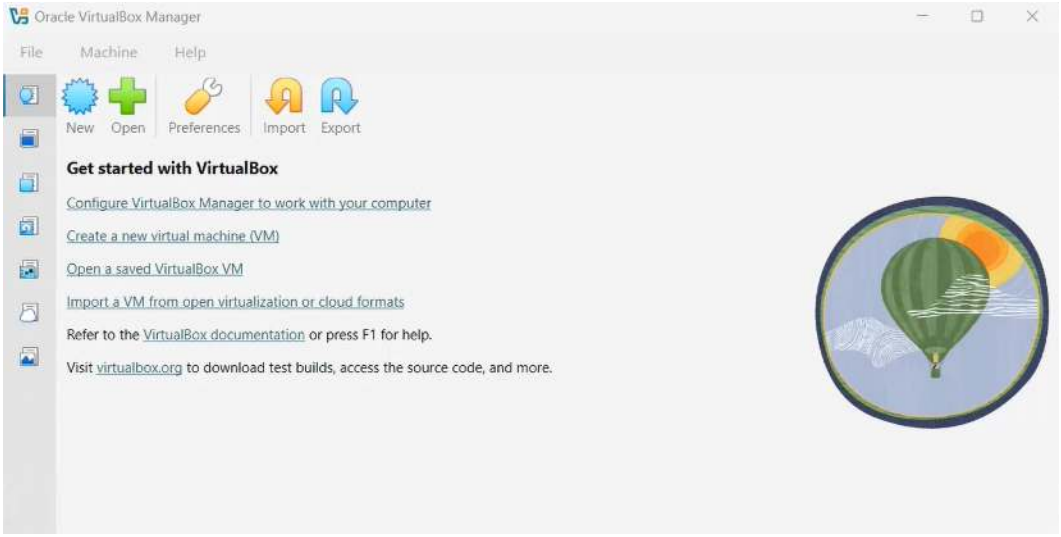


Figure 2.4: Main Oracle VirtualBox interface displayed after installation

So this is a simple explanation of how we download and install VirtualBox step-by-step in a clear and easy way.

Creating and installing the CentOS virtual machine

With VirtualBox installed, we can download CentOS Stream, create the virtual machine, and install the operating system.

Downloading the CentOS Stream ISO

Before creating the virtual machine, we need to download the CentOS Stream ISO file.

1. Open a web browser and go to the official CentOS downloads page at `https://www.centos.org/download/index.html`. Select the **Download** option on the CentOS home page.
2. The **Download** page displays the available CentOS Stream versions. Select CentOS Stream 10 or the latest version available, and locate the ISO download.
3. Select **Mirrors** under the **ISOs** column and download the `x86_64-dvd1` ISO.

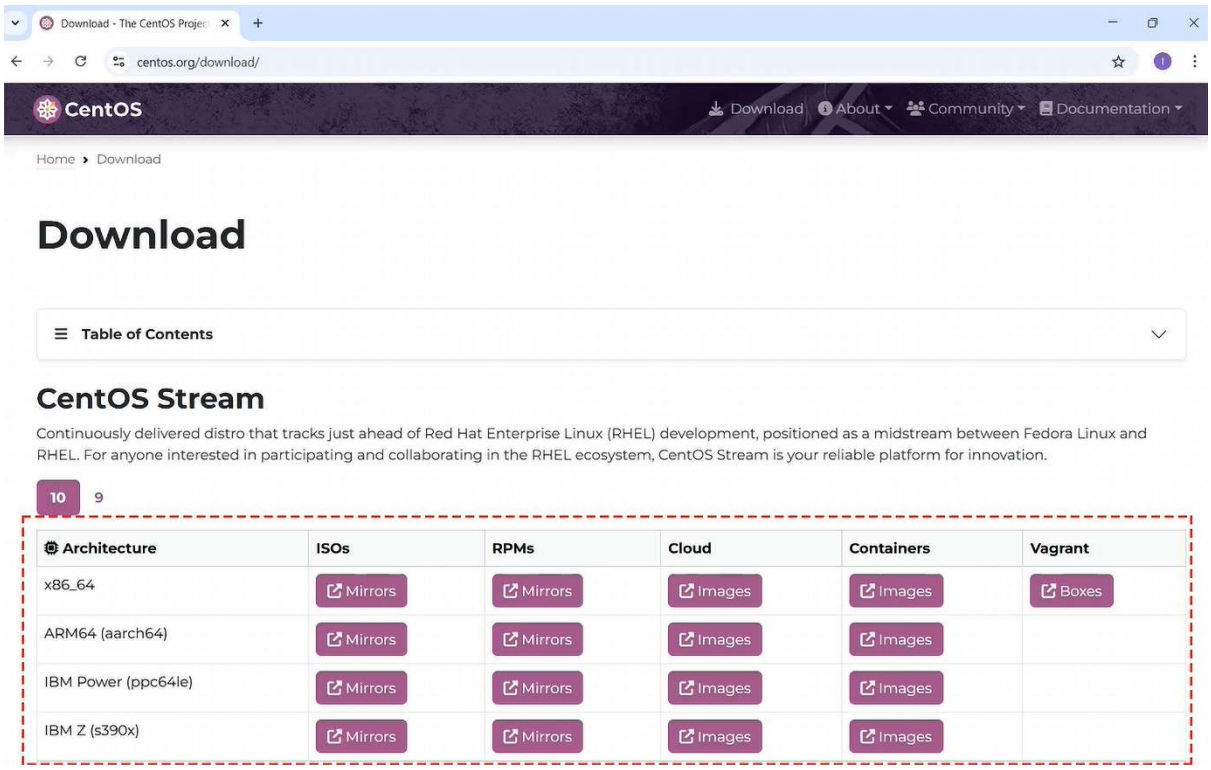


Figure 2.5: CentOS download page showing the available CentOS Stream release and the x86_64 DVD ISO option

4. The DVD ISO is a large download, and its filename reflects the selected CentOS Stream release and x86_64 architecture.

> **NOTE**
>
> Because the precise filename and size may change, confirm the downloaded ISO in the system's **Downloads** folder before continuing.

Creating the virtual machine

Now it is time to create the virtual machine and attach the ISO that will install CentOS. We will use the VirtualBox application installed in the previous section.

1. Open VirtualBox and select **New**. A window appears in which we can enter the basic information for the virtual machine.

2. Enter a recognizable name, such as `CentOS-Linux`. Retain the default folder location unless the virtual machine files need to be stored elsewhere.
3. Set the operating system type to **Linux**, select **Red Hat** as the distribution, and confirm that the version is **Red Hat (64-bit)**.

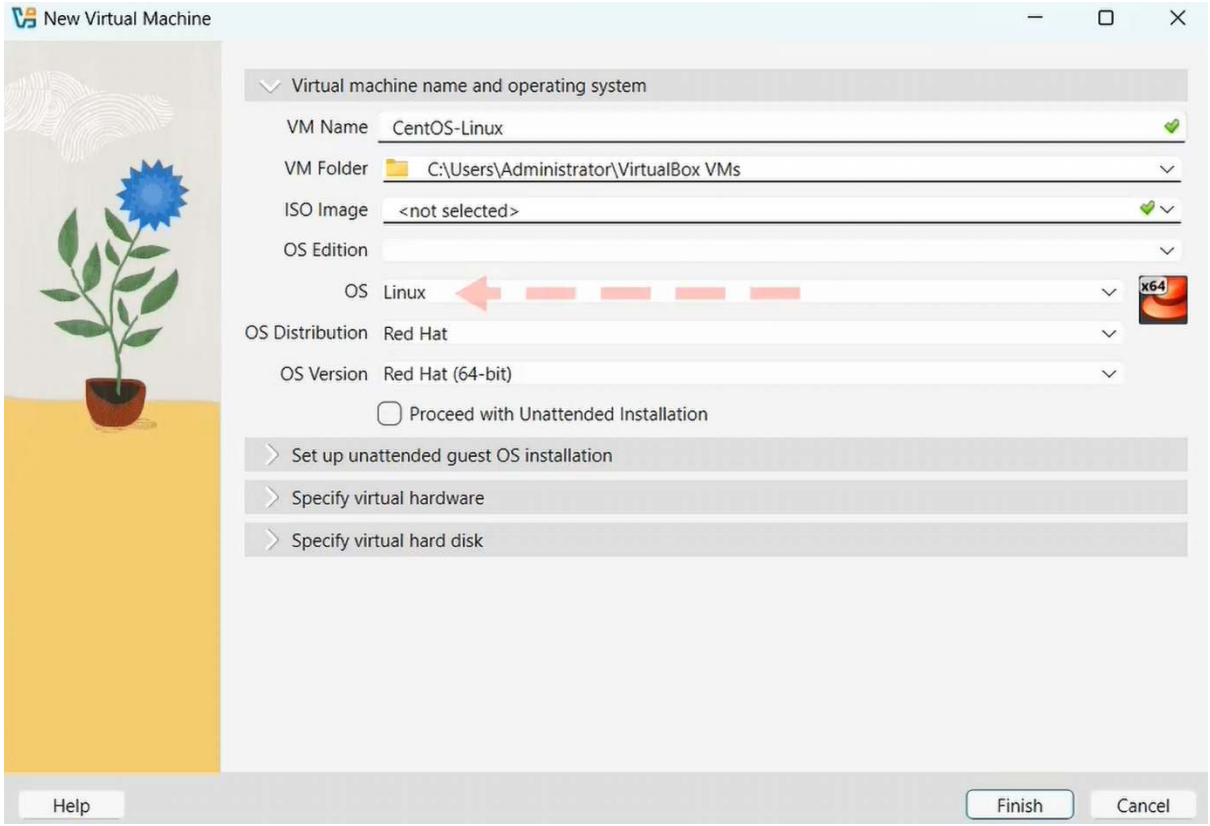


Figure 2.6: VirtualBox new virtual machine window

4. Next, configure the virtual hardware. On this screen, we adjust the memory and CPU values according to what our system can support. The screenshot reflects the values used in the original demonstration. For this book's setup, use 4 GB of memory and one CPU core. These options can still be adjusted according to what the host computer can support.

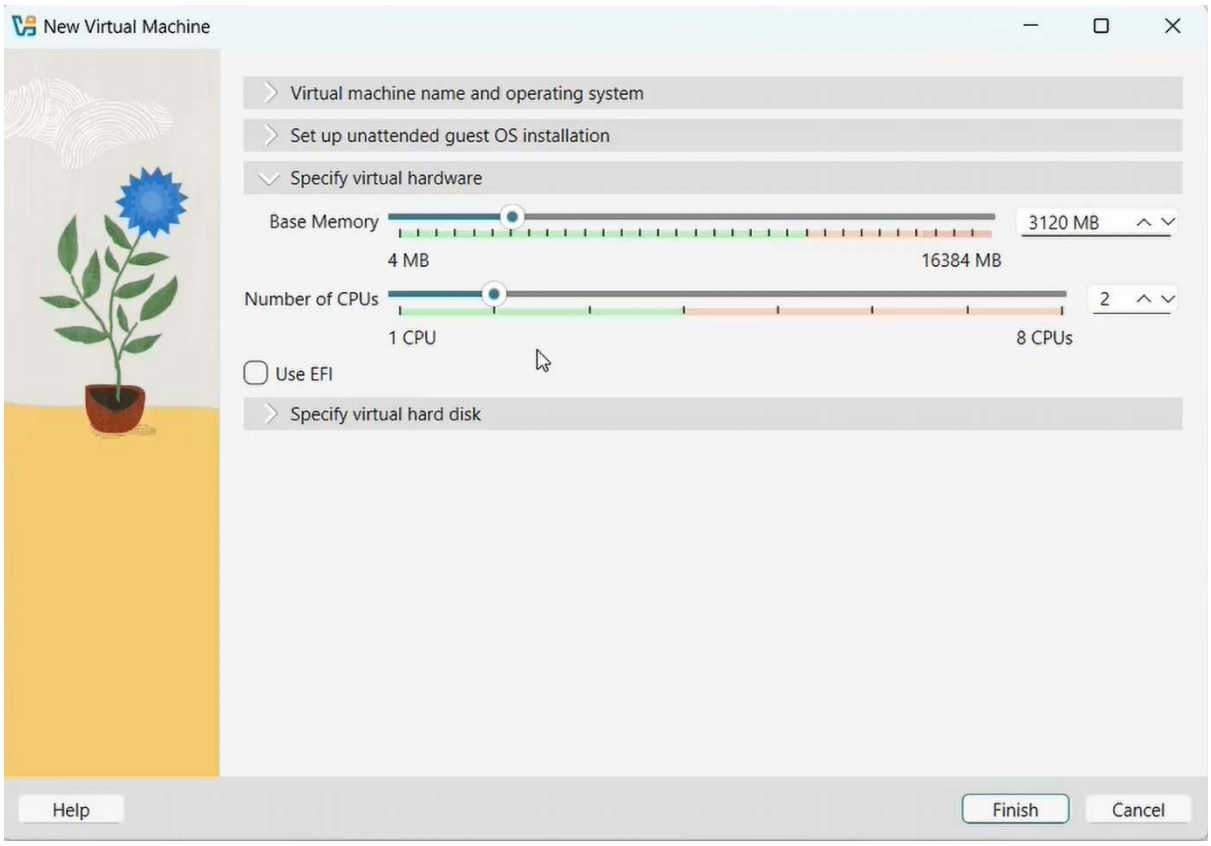


Figure 2.7: VirtualBox virtual hardware settings showing memory and CPU allocation

5. Then we scroll down to the next section, where we create the virtual hard disk. Here, we keep the default settings and change the storage size to 30 GB, which is enough for the installation, updates, and tools we'll use later.
6. After confirming the settings, select **Finish**.

The new virtual machine appears in the VirtualBox list with a summary of its memory, processor, storage, and network configuration.

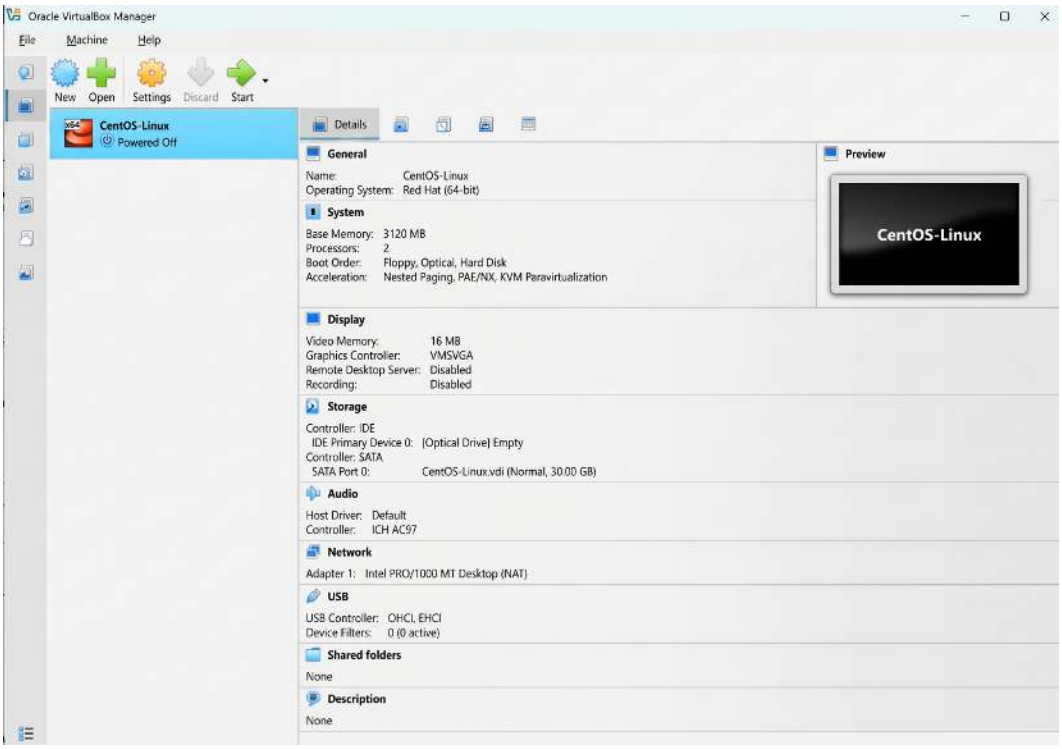


Figure 2.8: VirtualBox Manager showing the newly created CentOS Linux VM and its hardware summary

This means our virtual machine has now been successfully created and is ready for the operating system installation in the next steps.

Configuring the virtual machine settings

Now we have a few more settings to adjust before starting our virtual machine.

1. Click the Settings button at the top, and a new window appears with different options on the left side.
2. In the **System** tab, locate **Pointing Device** and select **USB Tablet** from the drop-down list. This setting provides smoother mouse movement inside the Linux desktop.
3. Next, open the **Network** tab. Adapter 1 is normally configured to use Network Address Translation, or **NAT**. Change it to **Bridged Adapter**. With bridged networking, the virtual machine behaves like another device on the same network and receives its own IP address from the router. This will make it easier to connect to the machine through SSH later.

NOTE

If bridged networking is unavailable on a corporate, public Wi-Fi, or managed network, keep **Adapter 1** set to NAT. For PuTTY access from the same Windows host, add a TCP port-forwarding rule in VirtualBox with host IP `127.0.0.1`, host port `2222`, and guest port `22`. Then connect PuTTY to `127.0.0.1` on port `2222` instead of using the virtual machine's bridged IP address.

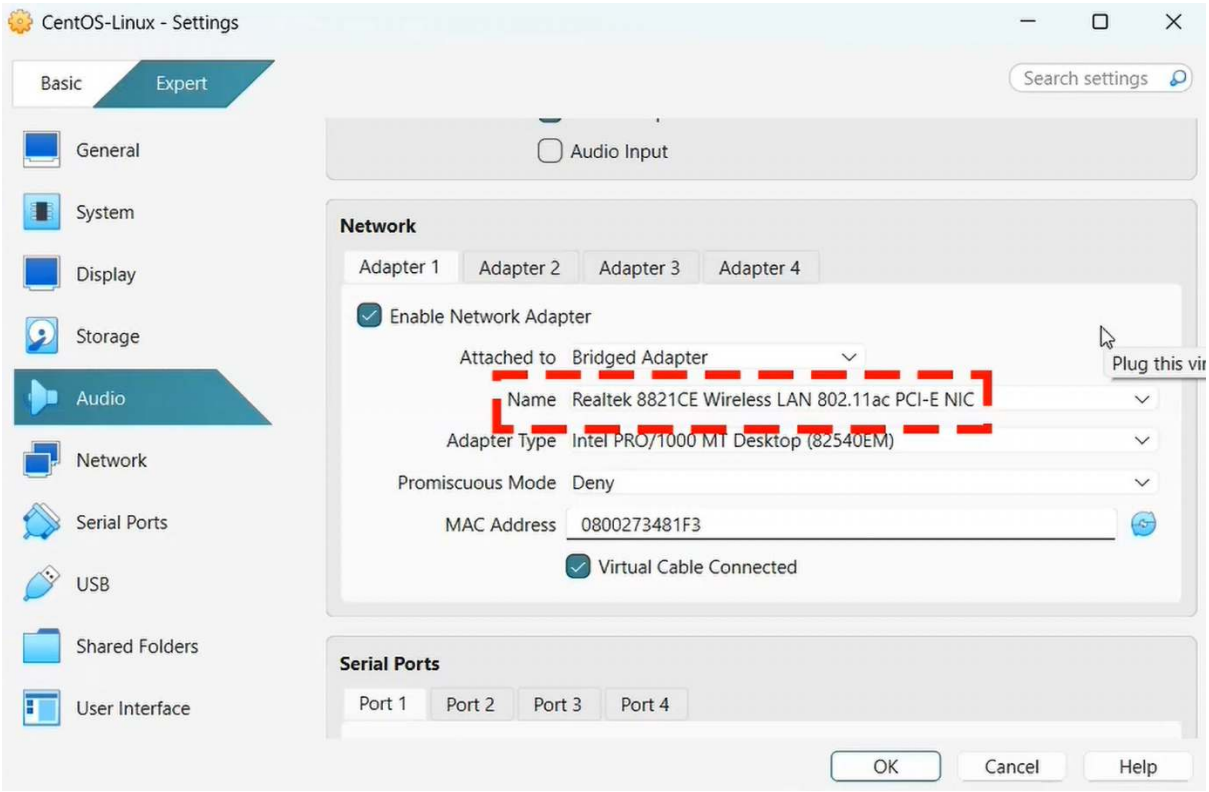


Figure 2.9: VirtualBox network settings showing Adapter 1 configured as a bridged adapter

4. Now open the **Storage** settings. The optical drive appears as **Empty**, indicating that no installation image is attached.
Select the optical drive icon and browse to the CentOS Stream x86_64 DVD ISO downloaded earlier. Choose the ISO so that VirtualBox

attaches it as a virtual DVD. The optical drive should now display the selected CentOS image.

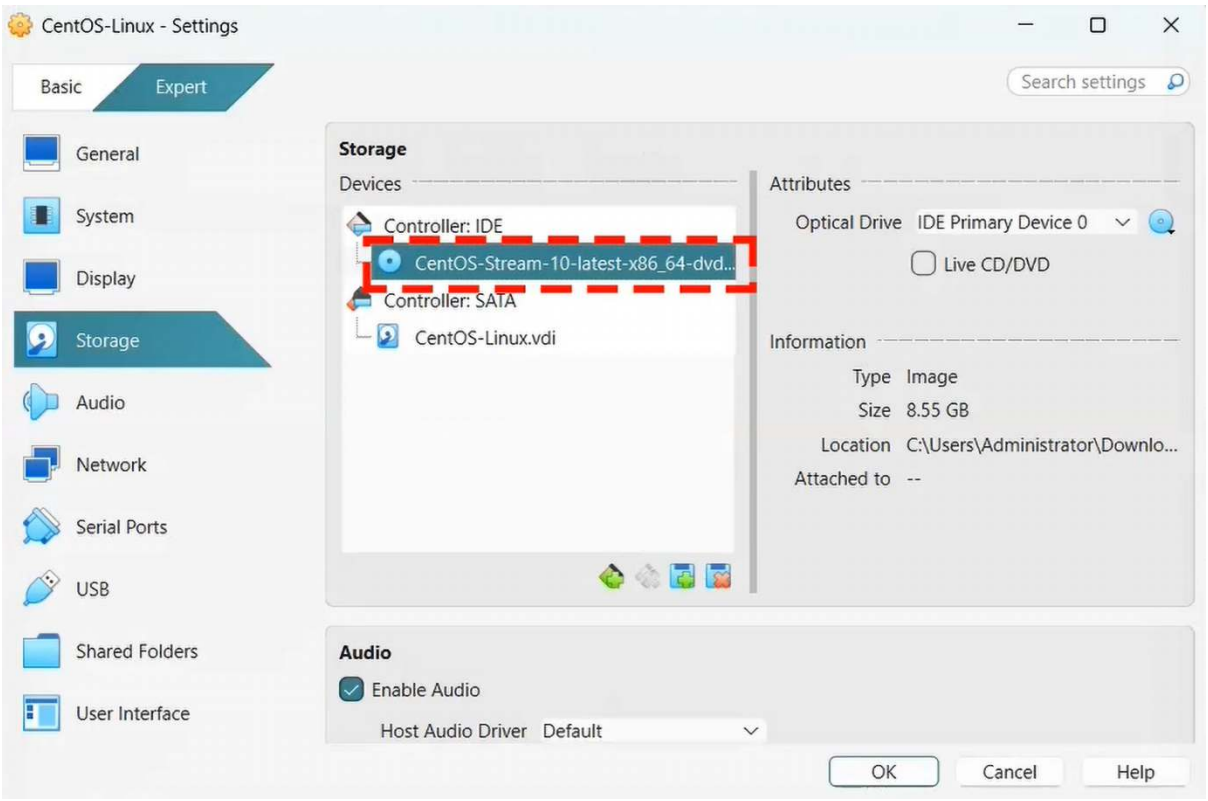


Figure 2.10: VirtualBox storage settings showing the CentOS Stream ISO attached to the virtual optical drive

This means the machine is now ready to boot from the installation file.

5. Select **OK** to save the settings.

Now that everything is ready, it's time to start the virtual machine.

Installing CentOS Stream

Here are the steps:

1. On the VirtualBox Manager screen, select the virtual machine and click **Start**. As soon as we click it, a new window opens, which is our virtual machine screen. Maximize the window if necessary so that the installation interface is easier to read.
2. On the boot screen, select the `Install CentOS Stream` option and press *Enter*.

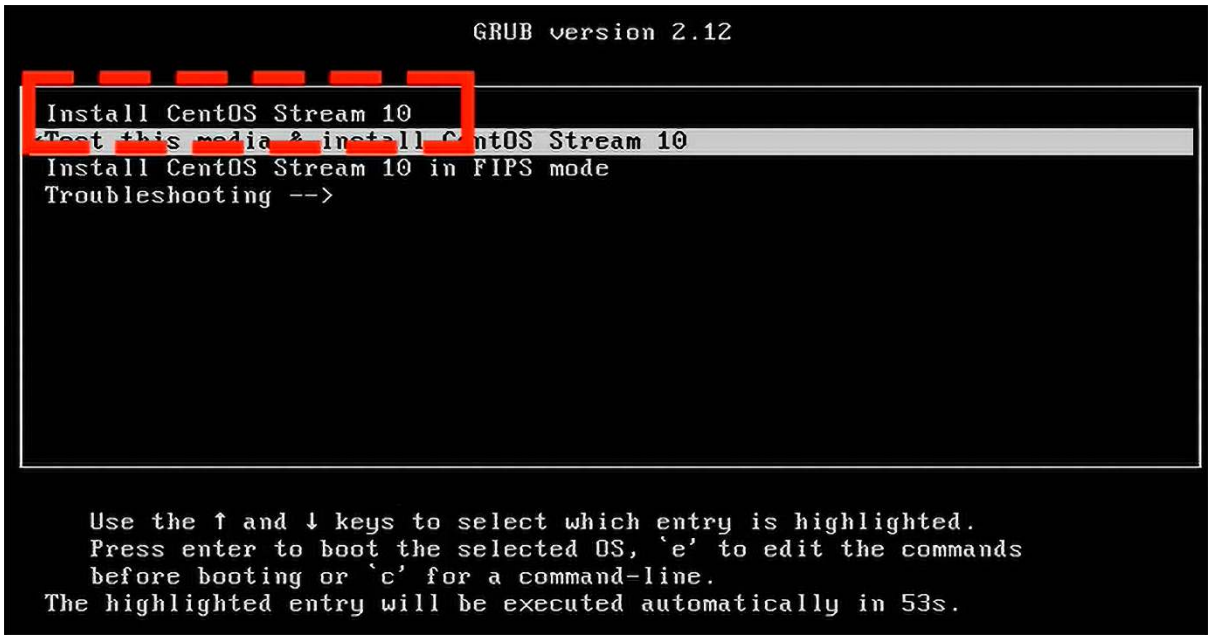


Figure 2.11: CentOS Stream boot screen with the installation option selected

3. Wait while the required files load. When the first installation screen appears, retain **English** as the installation language and select **Continue**.

   Now a new window appears, and this screen is called **INSTALLATION SUMMARY**. This page shows all the basic settings we must check before installing CentOS.

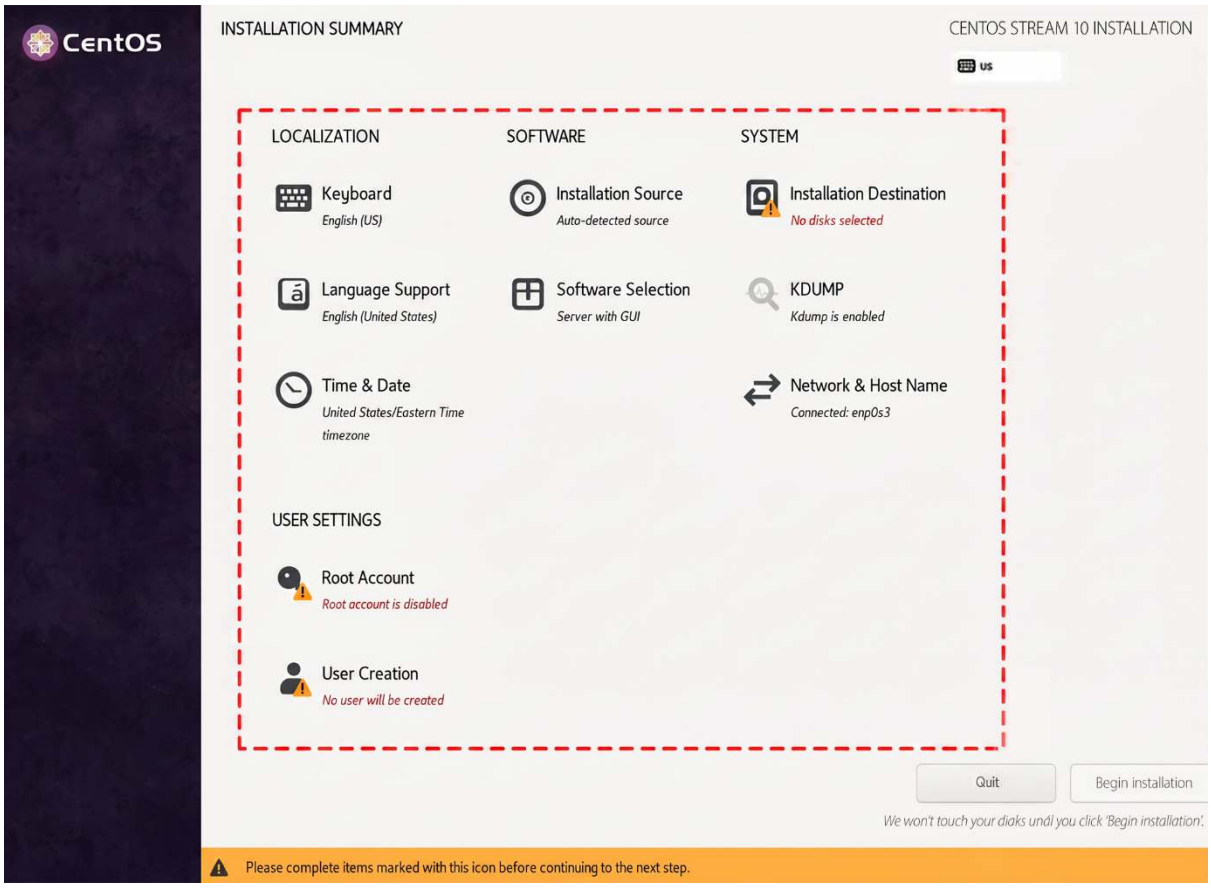


Figure 2.12: The installation summary screen on the VM

4. First, select **Installation Destination**. Choose the 30 GB virtual disk created earlier, retain the remaining default settings, and select **Done**.
5. Next, select **Network & Host Name**. Confirm that the network switch (**Ethernet**) is enabled.
   At the bottom of this window, there is a text box where we can type a hostname. Here we type, for example, `CentOS-Linux`, and then we select **Apply**, followed by **Done**.

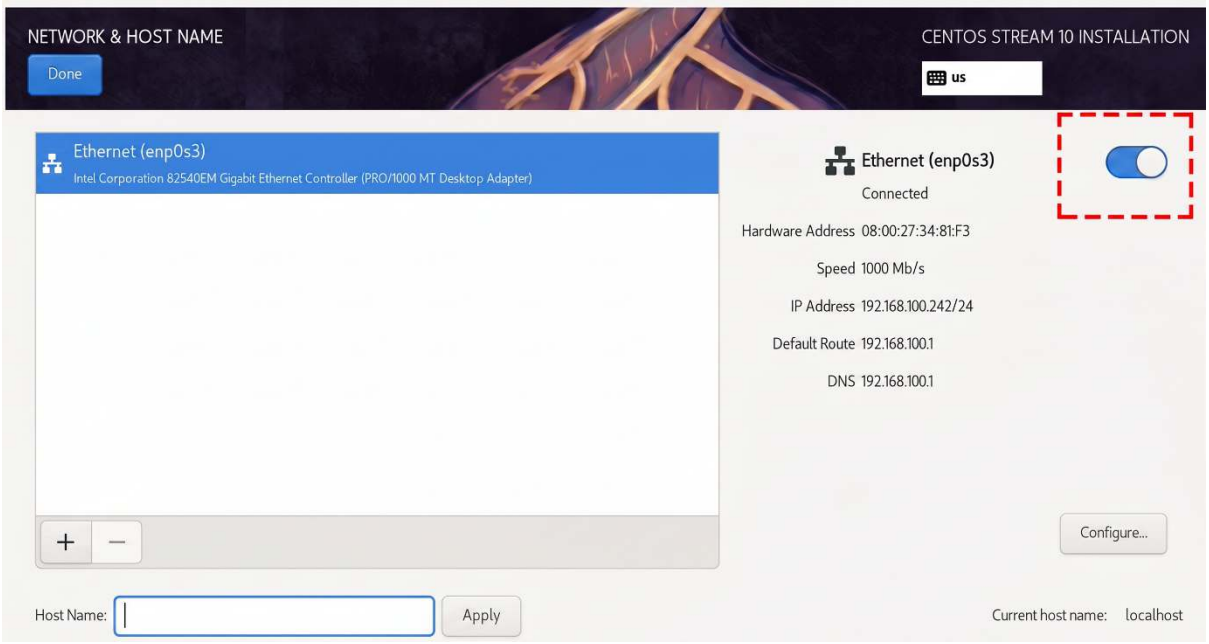


Figure 2.13: CentOS Network & Host Name screen showing Ethernet enabled

6. Leave the Root Account option disabled. We will connect through SSH using the regular administrative account and use sudo only when elevated privileges are required.
   Keeping direct root login disabled reduces risk and gives each administrative action a clearer audit trail.
7. Next, select **User Creation**. Create the regular user account by entering the full name and username, for example, `Imran Afzal`, with a generated username similar to `iafzal`.
8. Below that, you will notice two checkboxes already selected by default. This means this user will require a password to log in and will also have administrative privileges. Enter and confirm a suitable password.
9. Once the required settings are complete, select **Begin Installation**. Wait for the installation to finish, and then select **Reboot System**.
10. After the virtual machine restarts, select the account created during installation and enter its password. When the desktop loads, skip the optional system tour.

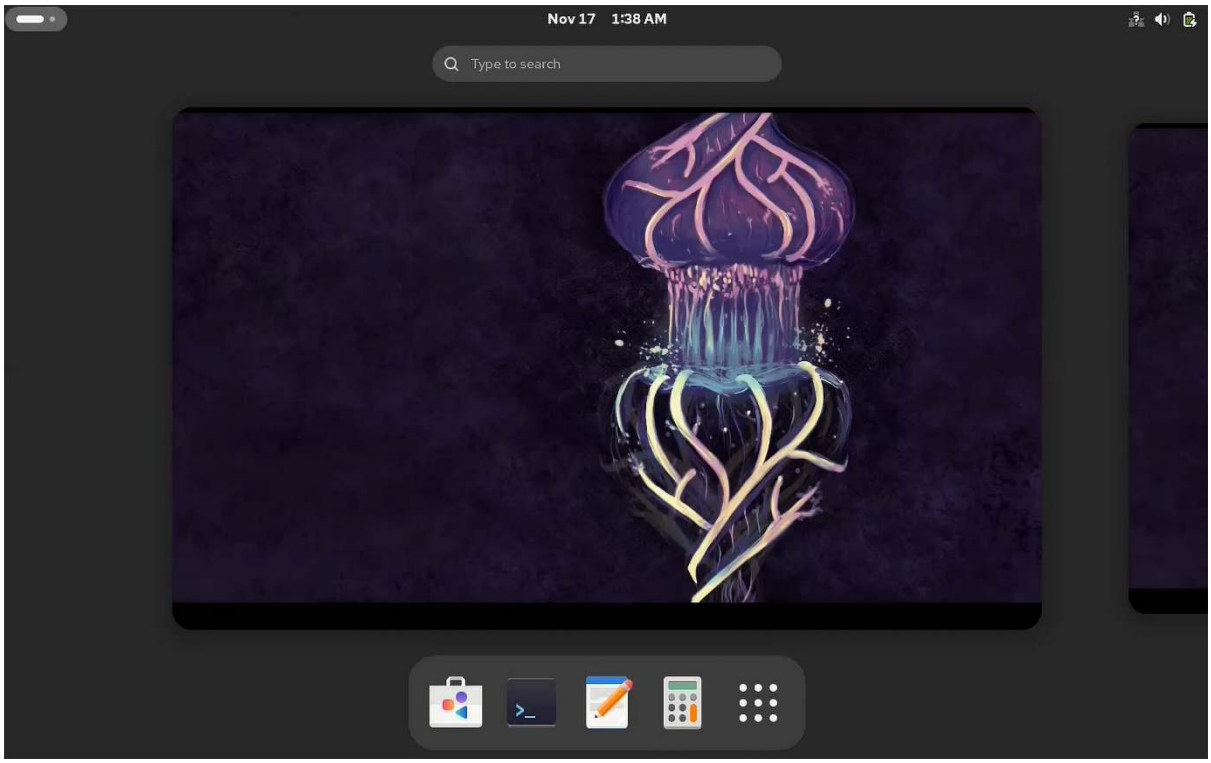


Figure 2.14: CentOS desktop displayed after the first successful login

Open the Terminal application from the desktop. The shell prompt should display the logged-in username and the machine hostname.

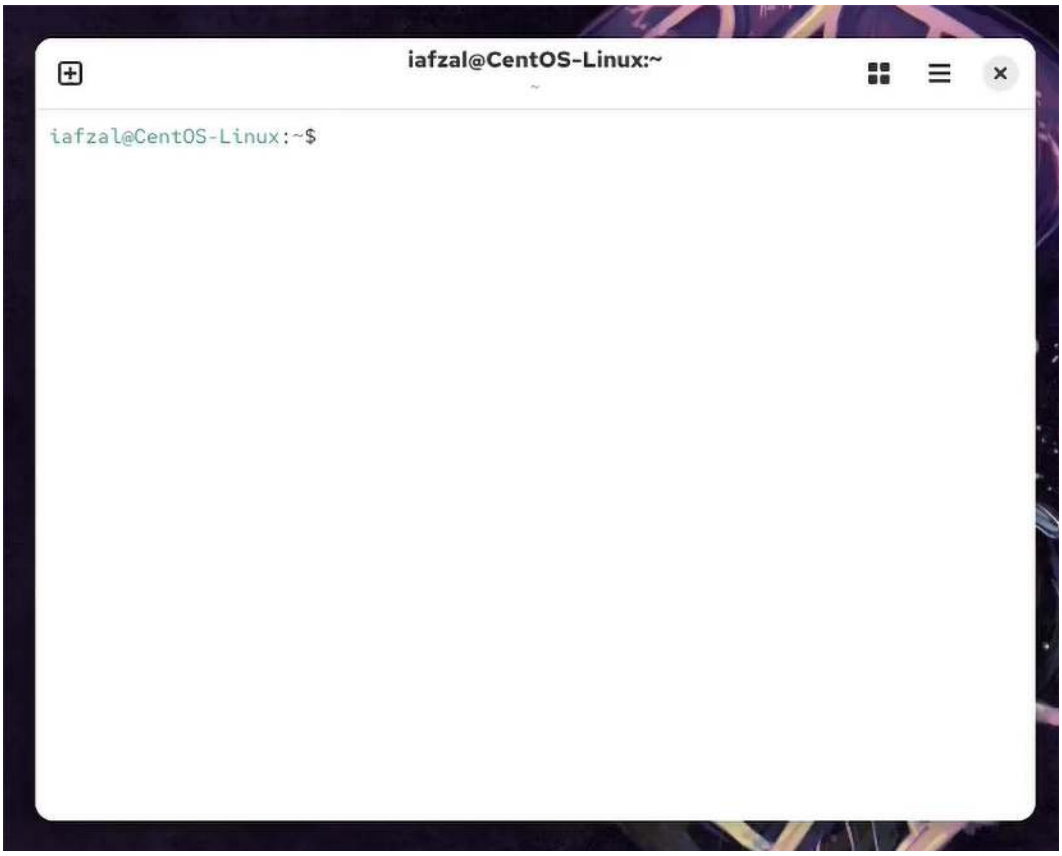


Figure 2.15: CentOS terminal showing the username and hostname in the shell prompt

This confirms that CentOS, the hostname, and the user account have been configured successfully.

Now that our system is ready, the next step is to connect this Linux machine remotely using PuTTY.

Connecting through PuTTY

PuTTY is an SSH client that allows a Windows computer to connect securely to the CentOS virtual machine. This makes it possible to manage Linux without working continuously through the VirtualBox console.

Before connecting, make sure the OpenSSH server is installed, running, and allowed through the CentOS firewall. In the CentOS terminal, run:

```
sudo dnf install -y openssh-server
sudo systemctl enable --now sshd
sudo firewall-cmd --permanent --add-service=ssh
sudo firewall-cmd --reload
```

1. Before opening PuTTY, identify the IP address assigned to the CentOS machine. In the CentOS terminal, run:

```
ip a
```

   Locate the IPv4 address assigned to the main network interface. In our setup, the interface is `enp0s3`, and its address is `192.168.100.242`.

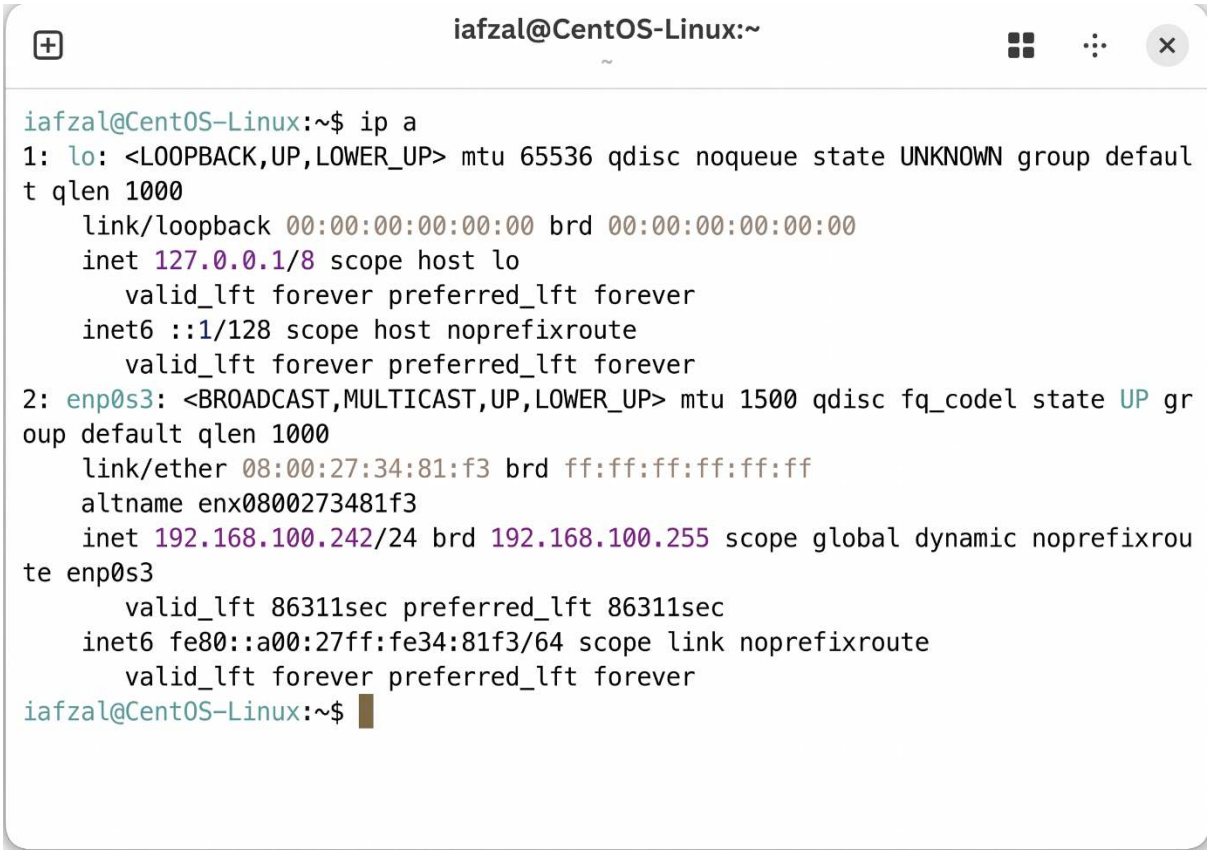


Figure 2.16: CentOS terminal showing the output of ip a, with the enp0s3 interface and its assigned IPv4 address visible

> **NOTE**
>
> The address shown in the book is only an example. Use the address assigned to the virtual machine in the current environment.

2. Open PuTTY on the Windows host. Enter the CentOS IP address in **the Host Name (or IP address)** field. Retain port 22 and confirm that **SSH** is selected.

> **NOTE**
>
> If PuTTY is not installed, download the current Windows installer from the official PuTTY website at `https://putty.org/index.html` and complete the installation before continuing.

3. Enter a name such as `CentOS-Linux` under **Saved Sessions**, and select **Save**. Saving the session is optional, but it prevents the IP address from having to be entered each time.

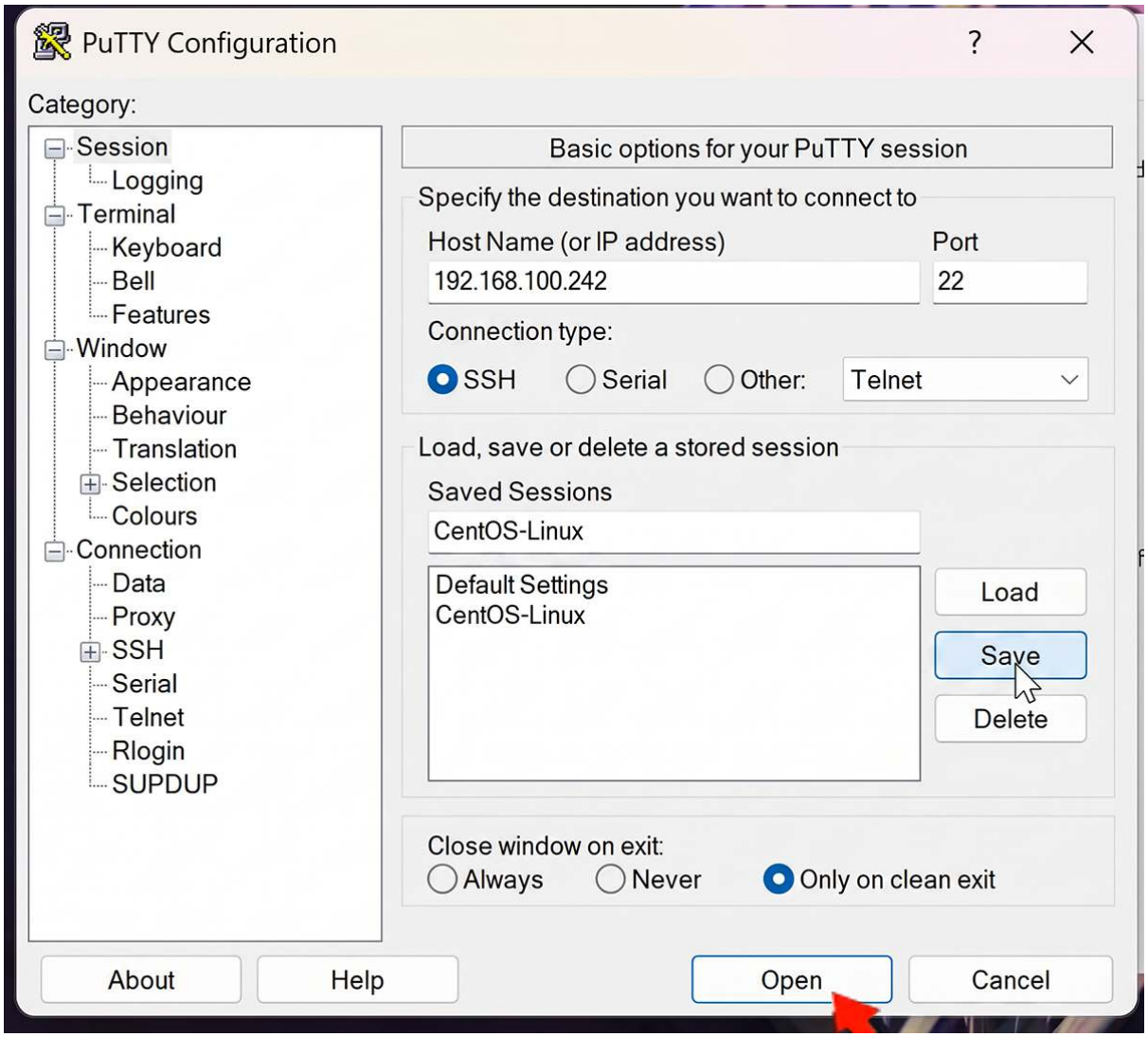


Figure 2.17: PuTTY configuration window

4. Select **Open** to initiate the SSH connection. On the first connection, PuTTY displays the server's host key. This message is normal because it is the first time we are connecting to this machine. We simply click **Accept** to save the SSH key and continue.
5. Enter the regular username created during installation, followed by its password. A successful login displays the CentOS shell prompt inside PuTTY.

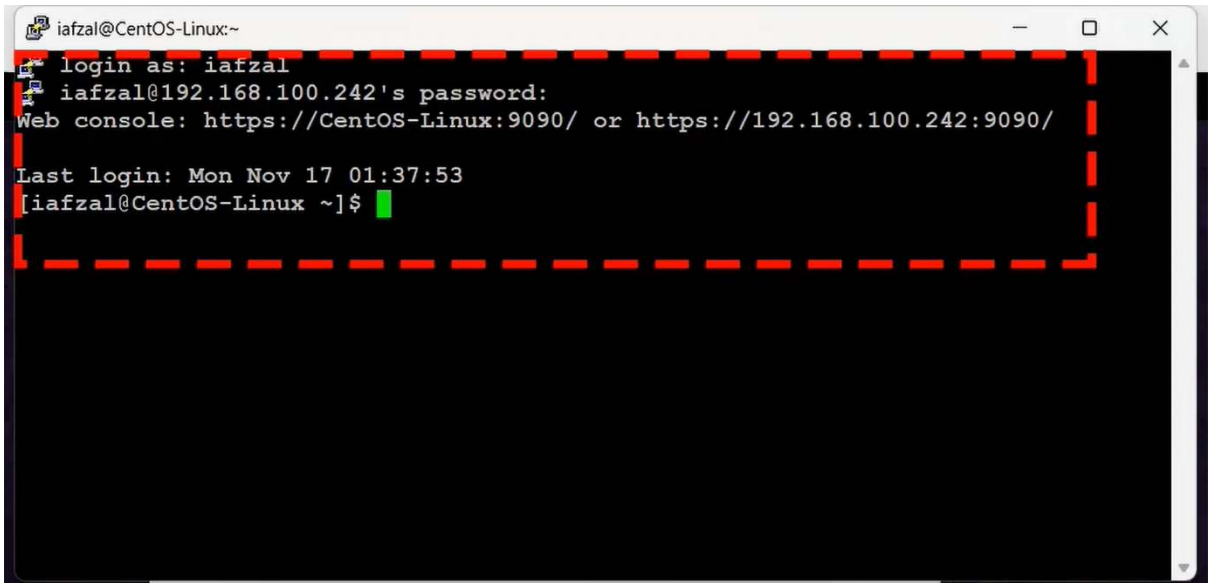


Figure 2.18: PuTTY terminal showing a successful SSH login to the CentOS virtual machine

This completes the setup of the CentOS virtual machine in VirtualBox and the remote SSH connection from Windows through PuTTY.

Exploring the Linux environment

Now that the CentOS virtual machine is running, we can explore the system and practise a few basic commands. These commands will help you identify the system, navigate the filesystem, create files, inspect logs, and understand permissions. This basic understanding will help a lot later in the book when we work on servers, automation, and daily Linux tasks.

Checking the user and system identity

Open the terminal locally or connect through the saved PuTTY session.

- Run the following command to identify the currently logged-in user:

```
whoami
```

  This command helps us confirm which user is currently logged in. In Linux, this is important because different users have different permissions, and some commands can only be run by certain users. The output should display the username created during installation.

- Next, we type the following command:

```
hostname
```

This command displays the name of the machine we are working on. This is useful when working with multiple servers or remote systems, because the hostname helps you quickly identify which machine you are connected to.

- To display more detailed system information, run:

```
uname -a
```

The `uname` command displays system information. The `-a` option requests all available information, including the kernel version and system architecture.

Figure 2.19: Terminal output from whoami, hostname, and uname -a

Now, let's move on and learn about the Linux filesystem.

Exploring the Linux filesystem

Linux uses an organized directory structure that begins at `/`, which is called the root directory.

Clear the terminal and list the directories under the root directory:

```
clear
ls /
```

The output includes directories such as `/home`, `/etc`, `/var`, `/bin`, `/usr`, `/tmp`, and `/root`.

Each directory has a particular purpose:

- `/home` contains the home directories of regular users.
- `/etc` contains system-wide configuration files.
- `/var` contains variable data such as logs, caches, and mail-related files.
- `/bin` provides access to common commands such as `ls`, `cp`, `mv`, `mkdir`, and `rm`.
- `/usr` contains programs, libraries, and shared files used by the operating system and installed applications.
- `/tmp` is used for temporary files.
- `/root` is the home directory of the root user.

To inspect some of these directories, run:

```
ls /home
ls /etc
ls /var
ls /bin
ls /usr
```

Let's move on and learn how to navigate inside the Linux filesystem.

Navigating between directories

Run `pwd` to display the current working directory:

```
pwd
```

For the demonstration account, the output should be similar to:

```
/home/iafzal
```

Move explicitly to the user's home directory:

```
cd /home/iafzal
```

Replace `iafzal` with the username created during installation.

Create a directory named test and confirm that it exists:

```
mkdir test
ls
```

Now intentionally run the following incorrect command:

```
cd /test
```

The leading `/` tells Linux to look for test directly under the root directory. Because the directory was created inside the user's home directory, Linux returns a "`No such file or directory`" error.

Correct the path by running:

```
cd test/
pwd
```

The working directory should now be similar to:

```
/home/iafzal/test
```

Return to the home directory with:

```
cd ~
pwd
```

The tilde character, ~, represents the current user's home directory.

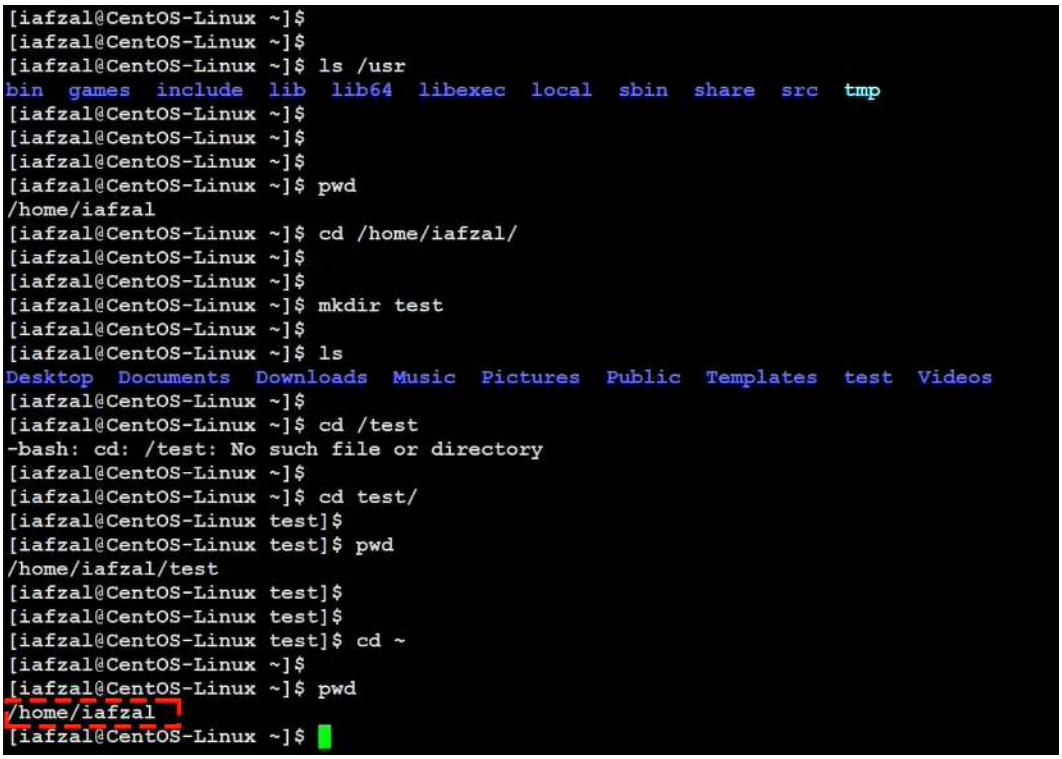


Figure 2.20: Terminal showing navigation between the home directory and the test directory

This small practice helps us to understand how to move inside directories, how a wrong path gives an error, and how Linux responds to commands.

Creating and reading a file

Let's move on and learn how to create a file in Linux. For this, we use a simple command called `touch`:

```
touch file.txt
```

The main purpose of the `touch` command is to create an empty file, meaning a file with no text inside.

Linux does not display a success message, so run `ls` to confirm that the file exists:

```
ls
```

Write text into the file:

```
echo "Hello Linux" > file.txt
```

The > operator redirects the output from `echo` into `file.txt`.

Display the file contents:

```
cat file.txt
```

The terminal should print:

```
Hello Linux
```

Figure 2.21: Terminal showing file.txt being created, populated with Hello Linux, and displayed with cat

Next, we check if this machine has internet connectivity.

Checking basic internet connectivity

Many Linux tasks depend on internet access, including installing software, downloading packages, and updating the system.

To confirm the network connection, we type the following command:

```
ping google.com
```

The `ping` command sends packets to the destination and waits for responses. Each response shows how long the packet took to return, measured in milliseconds.

Press *Ctrl+C* to stop the command. The resulting summary shows how many packets were transmitted and received and whether any packet loss occurred.

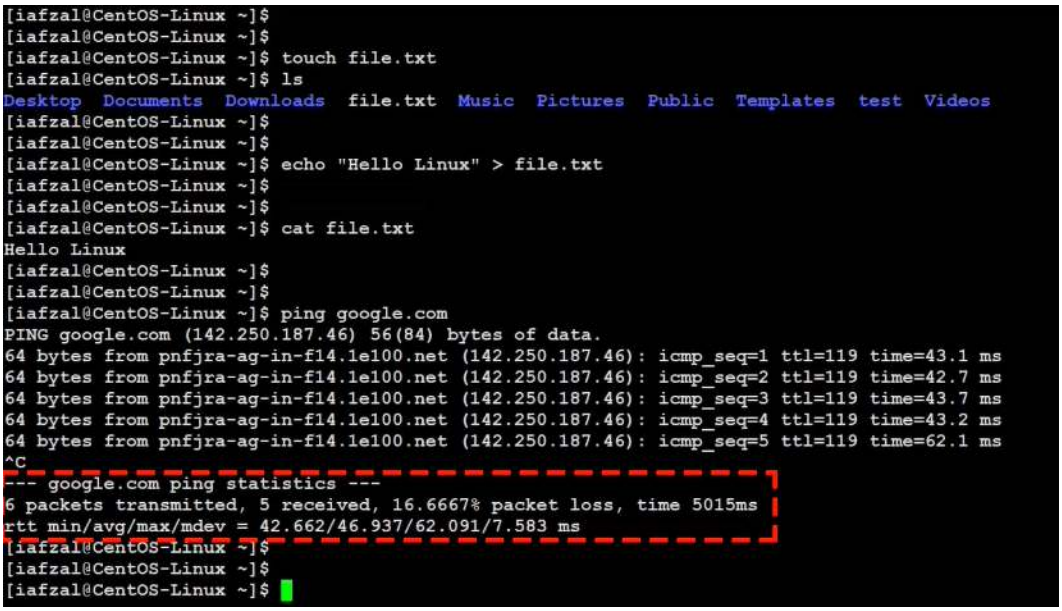


Figure 2.22: Terminal showing successful responses from ping google.com and the summary

Let's move on and look at system logs, which are useful when troubleshooting issues in Linux.

Reading system logs

System logs record errors, warnings, authentication attempts, network activity, and other system events. They are useful when something goes wrong because they help explain what happened and when it occurred.

To open the standard log directory and list its contents, run:

```
cd /var/log
ls
```

Depending on the logging services installed, this directory may contain files such as `boot.log`, `messages`, `secure`, and `cron`.

If the messages file is present, attempt to open it as the regular user:

```
cat messages
```

The command should return a permission error because ordinary users cannot read some sensitive log files.

Run the command again with administrative privileges:

```
sudo cat messages
```

Enter the regular user's password when prompted.

> **NOTE**
>
> If `/var/log/messages` is not present, the system journal can be inspected with `sudo journalctl -n 20` instead.

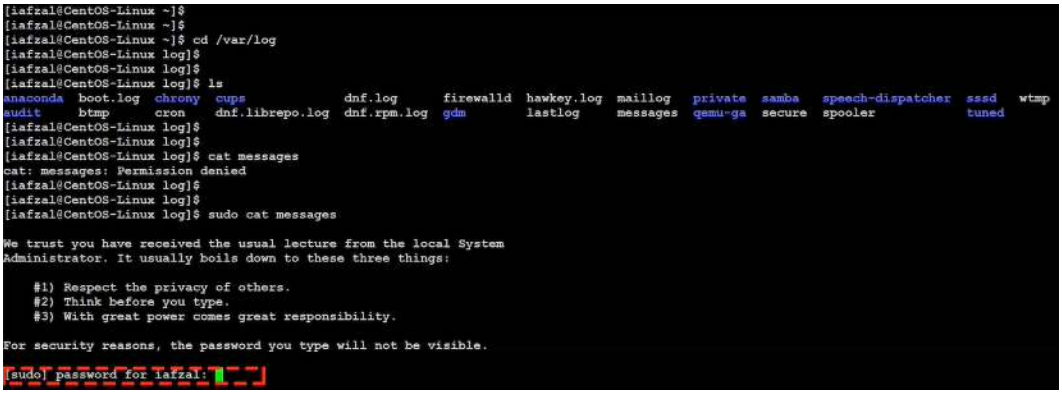


Figure 2.23: Terminal showing access to the messages log with sudo

Each log entry contains information such as a date and time, hostname, process name, and message.

Working with elevated privileges

The root user has unrestricted control over the Linux system. This makes root access powerful, but it also means that an incorrect command can damage the system.

In this example, we open an administrative login shell with:

```
sudo -i
```

Enter the regular user's password when prompted. The `sudo` command opens an administrative shell, and the prompt changes from `$`, which identifies a regular user, to `#`, which identifies the root shell.

After opening the administrative shell, running `ls` shows the contents of the root user's home directory rather than the regular user's home directory.

Move to `/etc` and display a detailed listing:

```
cd /etc
ls -ltr
```

The options used with `ls` provide a long listing and sort entries by modification time in reverse order, placing the most recently modified entries toward the bottom.

The listing shows permissions, ownership, group, size, modification time, and the name of each file or directory.

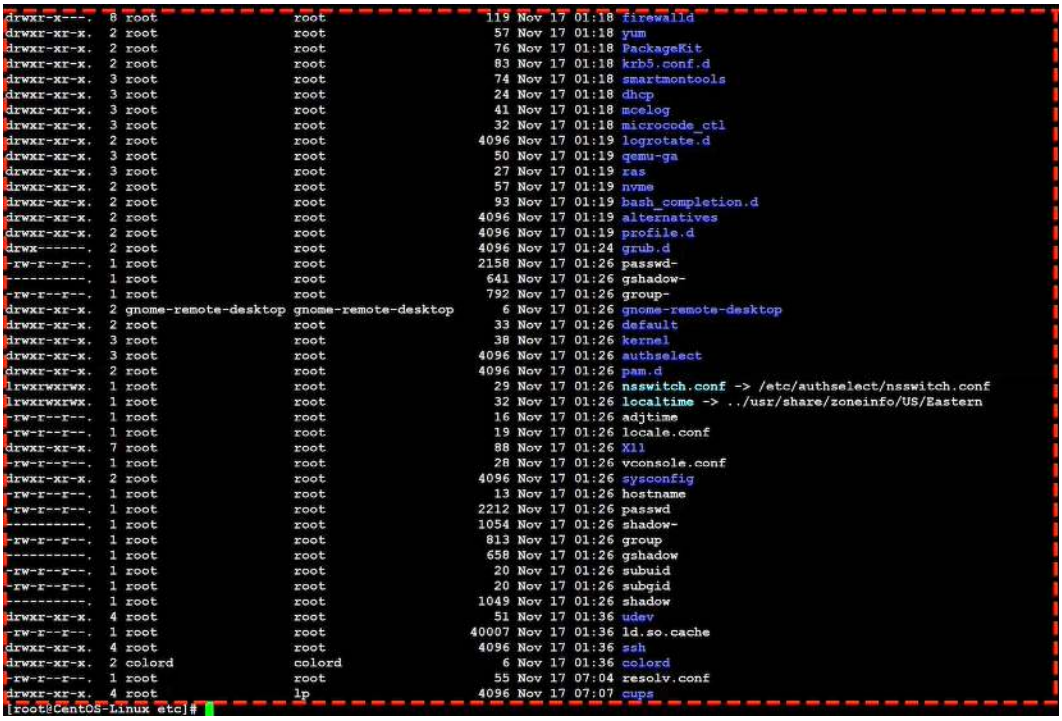


Figure 2.24: Terminal showing the listing from /etc

This way of viewing files is useful when we want to track recent changes or troubleshoot configuration files.

Let's move on and learn about permissions and how we can change them.

Inspecting and modifying permissions

Return to the administrative shell's home directory and create a test file:

```
cd ~
touch test.txt
ls -ltr
```

A newly created file may display permissions similar to:

```
-rw-r--r--
```

The first character identifies the item type. A hyphen indicates a regular file. The remaining characters are divided into three permission groups:

- The first group, `rw-`, applies to the owner.
- The second group, `r--`, applies to the file's group.
- The third group, `r--`, applies to everyone else.

Within each group, `r` means read, `w` means write, and `x` means execute.

You can add execute permission to the file using:

```
chmod +x test.txt
```

Run `ls -ltr` again and observe where the `x` permission appears:

```
ls -ltr
```

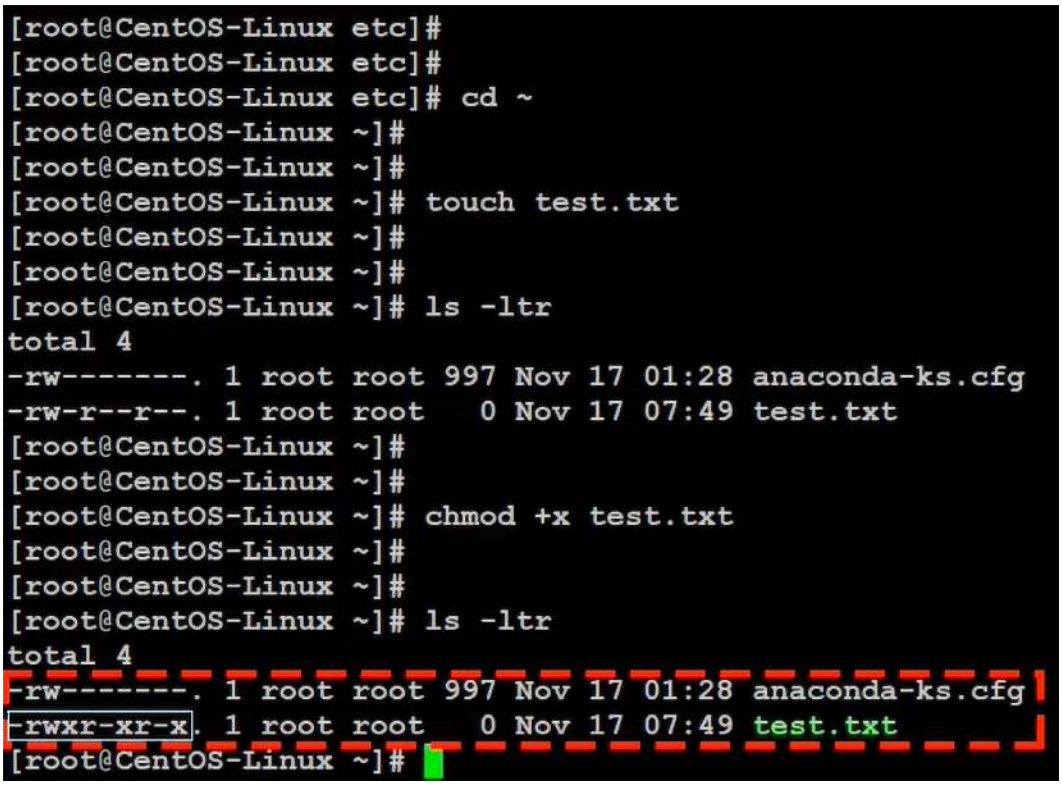


Figure 2.25: Terminal comparing the permissions of test.txt before and after chmod +x test.txt

These commands provide a foundation for navigating directories, creating files, checking connectivity, reading logs, working with elevated privileges, and managing permissions.

When the privileged work is complete, return to the regular user account:

```
exit
```

Verifying network connectivity

Before installing packages or AI tools, we need to confirm that the CentOS virtual machine has a working network connection.

The virtual machine was configured earlier to use a bridged adapter. Before starting it, open its VirtualBox network settings and confirm that **Bridged Adapter** remains selected.

> **NOTE**
>
> Bridged mode makes the virtual machine work like a separate device on the same network. It receives its own IP address from the network router, just like any other computer or laptop in the house or office.
>
> This makes remote access through SSH and external connections much easier. That's why we use bridged mode here.

Start the virtual machine, sign in, and open the terminal.

The first thing we do is check the network connection status. For this, we type the following command:

```
nmcli device status
```

`nmcli` stands for **NetworkManager command-line interface**. The command displays each network device, its type, and its current state.

The main Ethernet interface should appear as `enp0s3` or a similar name, with its state shown as `connected`. The `lo` device is the loopback interface.

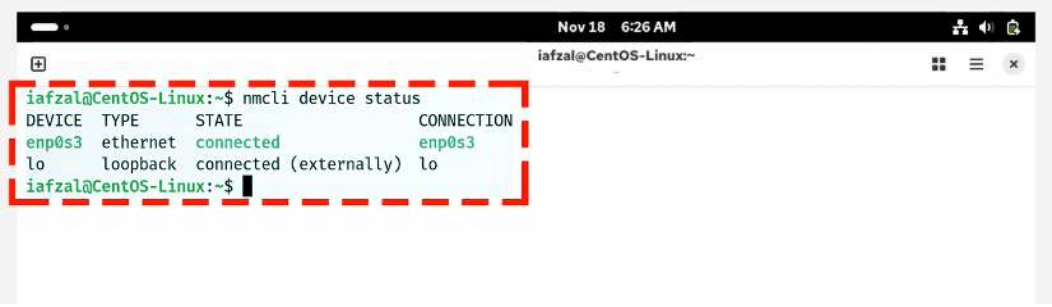


Figure 2.26: Terminal output from nmcli device status showing the Ethernet interface as connected and the loopback device

Next, display the available interfaces and IP addresses:

```
ip a
```

Locate the IPv4 address assigned to the main interface. Here we have `192.168.100.242`, but the address will vary between environments.

An address beginning with `192.168` belongs to one of the private IPv4 address ranges and indicates that the machine has received an address from the local network.

If the interface shows `DOWN` or `NO-CARRIER`, or no IPv4 address is assigned, recheck the VirtualBox network settings.

Next, we check routing because routing decides how this machine sends traffic outside the local network, especially when accessing the internet or external systems:

```
ip route
```

Look for an entry beginning with `default`. This entry identifies the gateway used when the machine sends traffic outside its local network.

In this case, the default gateway is `192.168.100.1`, and the local network is shown as `192.168.100.0/24`. These values are specific to that environment and will differ for other readers.

If no default route appears, the machine may have an IP address but still be unable to reach the internet.

Figure 2.27: Terminal output from ip route showing the default gateway and local network route

Further, we also need to check DNS.

Testing DNS resolution

DNS, or the **Domain Name System**, converts human-readable domain names into IP addresses. Without working DNS resolution, many internet services

will fail even when the network interface and route are configured correctly.

If `nslookup` is not installed, install the bind-utils package first:

```
sudo dnf install -y bind-utils
```

Then test DNS resolution:

```
nslookup google.com
```

A successful response identifies the DNS server that handled the query and returns one or more addresses for `google.com`.

The IP addresses returned for `google.com` can change. You should confirm that the command returns a valid response rather than expecting the exact address shown here.

Figure 2.28: Terminal showing a successful nslookup response for google.com

Once DNS is confirmed, the connection has passed the required checks. A service restart is not needed during normal verification.

Restarting and checking NetworkManager

Restart `NetworkManager` only as a troubleshooting step. Perform this action from the VirtualBox console rather than an active SSH session, because the restart can interrupt the remote connection.

From the VirtualBox console, run:

```
sudo systemctl restart NetworkManager
```

Enter the user password if prompted.

Check the service status:

```
sudo systemctl status NetworkManager
```

The service should report that it is active and running.

Figure 2.29: Terminal showing NetworkManager in an active and running state

You can also see the word `enabled`, meaning it will automatically start every time the machine boots.

The logs below also show messages like connected and activation successful, confirming that our network interface is active and everything is fine. This confirms that the network is fully working and ready to use.

These are the simplest ways to confirm that your machine has internet access, because without a working network connection, many commands, updates, and installations will not work properly.

Reviewing AI tools for system administrators

Linux administrators regularly work with terminal commands, logs, networking, security, scripting, and automation. AI-assisted tools can support many of these tasks by translating natural-language requests, explaining errors, analyzing information, and helping generate scripts.

The following overview introduces several categories of AI tools.

- **GitHub Copilot CLI** works inside the terminal and can help generate commands from natural-language requests. For example, an administrator could describe a task such as:

  ```
  Compress all log files in /var/log and move them to the backup directory.
  ```

  The tool can then suggest a corresponding Linux command. This can help beginners who are still learning command syntax and experienced administrators who want to work more quickly.

- **Warp Terminal** includes an AI assistant that can explain errors, suggest corrections for commands, and assist with shell scripts and Bash automation.
- **Elastic AI Assistant** works with data stored in the Elastic Stack. Administrators can use natural-language questions to investigate system, firewall, or application logs and receive more readable explanations.
- **Wazuh AI Analyst** can summarize security events, help identify suspicious activity, and support investigation of repeated or unauthorized SSH login attempts.
- **Visual Studio Code** can support Bash, Python, YAML, Terraform, and Ansible development through AI coding extensions such as GitHub Copilot and Amazon Q Developer.
- **Tabby** is an open source, self-hosted AI coding assistant. A local deployment can be useful when an organization wants code-completion services to run inside its own environment.

Cloud platforms also provide AI assistants, including **Amazon Q Developer**, **Azure Copilot**, and **Gemini Cloud Assist**. Their current capabilities include explaining resources, supporting operational investigation, and helping administrators work with cloud environments through natural-language requests.

The practical chapters in this book focus on three command-line tools: **ChatGPT CLI** for prompts, scripts, troubleshooting, and log analysis; **Shell Genie** for translating plain-English requests into Linux commands; and **Aider** for working with Bash files and script templates. The broader tools above

provide context, while these three form the hands-on path through the rest of the book.

AI tools can make system administration smoother and faster, but their commands, scripts, explanations, and recommendations still need to be reviewed before they are used on a Linux system.

Summary

In this chapter, we prepared the workspace required for the practical exercises in the rest of the book. We installed VirtualBox, created a CentOS Stream virtual machine, configured its hardware and networking, and installed the operating system.

We then explored the Linux environment through basic commands, filesystem navigation, file creation, logs, administrative privileges, and permissions. We also connected to the virtual machine through PuTTY and verified its network interface, IP address, routing, DNS resolution, and NetworkManager service.

Finally, we reviewed several ways in which AI-assisted tools can support command generation, scripting, troubleshooting, log analysis, security, and cloud-based system administration. The Linux workspace is now prepared for installing and configuring the AI tools used in the following chapters.

Knowledge check

Test your understanding of the concepts covered in this chapter.

1. What is the purpose of VirtualBox in this chapter's setup?
    A. To replace the Windows host operating system
    B. To create and run the CentOS virtual machine
    C. To analyze Linux log files
    D. To configure the network router
2. What is one effect of configuring the virtual machine to use a bridged adapter?
    A. The virtual machine receives an address on the same network as the host
    B. The virtual machine can no longer access the network

C. The virtual machine automatically becomes the network router
D. VirtualBox removes the virtual hard disk

3. Which command displays the currently logged-in user?
   A. hostname
   B. pwd
   C. whoami
   D. uname -a
4. What does the root directory / represent?
   A. The home directory of the root user only
   B. The starting point of the Linux filesystem hierarchy
   C. The directory used only for log files
   D. The directory in which VirtualBox is installed
5. Which command displays the current working directory?
   A. pwd
   B. cd
   C. touch
   D. cat
6. What information does nmcli device status provide?
   A. The contents of a text file
   B. The state of the available network devices
   C. The permissions assigned to a file
   D. The current kernel version
7. What does a default entry in the output of ip route identify?
   A. The current Linux user
   B. The directory opened by default
   C. The gateway used to reach networks outside the local subnet
   D. The default file permissions
8. What does a successful nslookup google.com result confirm?
   A. The root user is enabled
   B. DNS resolution is working
   C. The virtual disk has enough free space

D. PuTTY has saved the SSH session

Answers

1. **b. To create and run the CentOS virtual machine.** VirtualBox provides the virtualized hardware environment in which CentOS is installed.
2. **a. The virtual machine receives an address on the same network as the host.** Bridged mode makes the VM appear as another device on the local network.
3. **c. whoami.** This command displays the username associated with the current session.
4. **b. The starting point of the Linux filesystem hierarchy.** All Linux directories and files are organized beneath /.
5. **a. pwd.** The command prints the path of the current working directory.
6. **b. The state of the available network devices.** The output identifies devices and indicates whether they are connected, disconnected, or unavailable.
7. **c. The gateway used to reach networks outside the local subnet.** The default route is used when no more specific route matches the destination.
8. **b. DNS resolution is working.** A successful result shows that the system can translate the domain name into an IP address.

3

Using ChatGPT/Claude from the Linux Command Line

Linux administrators spend much of their working day in the terminal. When a command is forgotten or an error appears, the usual response is to open a browser, search for an answer, and then return to the terminal. Doing this again and again interrupts the work. A command-line AI workflow keeps the question, explanation, and next step in the place where the task is already happening.

In this chapter, we will first understand what the term ChatGPT CLI means in the context of this book and why it can be useful to Linux users. We will then verify Python, install pip and the OpenAI Python SDK, create an API key, and build a small Python program that sends prompts from the CentOS terminal. We will also set up Anthropic Claude as an alternative command-line workflow. Finally, we will make the program interactive so you can enter a different question each time it runs.

In this chapter, we will cover the following topics:

- Understanding ChatGPT CLI
- Installing Python, pip, and the OpenAI SDK
- Creating an OpenAI API key
- Sending the first request from a Python script
- Making the program interactive

Let's get started!

NOTE

Download the PDF version of this book

> Your purchase includes a DRM-free PDF copy of this book, the code bundle, and a range of exclusive benefits. To unlock everything, follow the *Free benefits with your book* section in the *Preface*.

Technical requirements

To follow the hands-on procedures in this chapter, make sure the following are available:

- The CentOS Stream virtual machine prepared in *Chapter 2*
- A working internet connection from the virtual machine
- A web browser for accessing the OpenAI Platform or the Anthropic Console
- An OpenAI Platform account with API access and an API billing method or credits, or an Anthropic Console account with an API key and billing enabled, depending on the workflow followed
- A text editor such as Vim

Understanding ChatGPT CLI

Before we talk about ChatGPT CLI, let's make the idea simple by starting with ChatGPT itself.

ChatGPT is an AI tool that can answer questions, explain commands, help diagnose errors, and create scripts from instructions written in normal English. Most people begin by using ChatGPT in a web browser. They enter a question on the website and read the answer there. A command-line workflow brings the same kind of interaction into the Linux terminal, where a system administrator is already working.

CLI stands for **command-line interface**. It means that we interact by typing commands in a terminal instead of clicking buttons in a graphical window. In this book, the term ChatGPT CLI refers to a small command-line workflow that uses the official OpenAI Python SDK and a Python script to send prompts to an OpenAI model. It does not refer to a separate application installed under the package name ChatGPT CLI.

Why a command-line AI workflow is useful

System administrators already spend much of their time in the terminal, so keeping AI assistance there can be very useful. If you forget how to create a new user, restart a service, change permissions, or fix an error, you do not need to leave the terminal, search online, and then come back. You can simply ask a question such as, 'How do I add a new user in Linux?' and receive the answer inside the terminal.

Another powerful use of ChatGPT CLI is creating Bash scripts. For example, you can ask for a script that checks disk usage, monitors logs, or schedules backups. This saves time when the script is long or when you are not completely sure about the syntax. Instead of starting with a blank file, you can describe the task in normal English and use the generated script as a starting point.

ChatGPT CLI is also helpful when Linux returns an error that looks confusing. Searching for the same message online can take time, but in the terminal you can provide the error and ask, 'What does this mean, and how can I fix it?' The response can explain the problem in simpler language and suggest the next step.

The same approach works with complicated commands. If a command contains many options and symbols, ask ChatGPT CLI to explain every part step by step. This does more than provide an answer: it helps you understand the command while you are working with it.

That is what makes the command-line workflow powerful for Linux users. It keeps the work, the explanation, and the learning process together in the terminal.

NOTE

The ChatGPT CLI helps you learn while working, but the generated answer is still a suggestion. Read every command and script before running it, check that paths and account names match the system, and never paste passwords, API keys, private configuration, or sensitive production logs into a prompt.

Installing Python, pip, and the OpenAI SDK

In this section, we will install Python, pip, and the package required for our ChatGPT command-line workflow. These tools are important because the

program is written in Python, and the OpenAI SDK is installed as a Python package. Before we build the program, we must make sure Python and pip are installed and working properly.

Log in to the CentOS virtual machine with the account created in *Chapter 2*. The examples use the `iafzal` account, but your username may be different. Administrative installation commands use `sudo` only where elevated privileges are required. The Python script itself does not need to run as root.

1. Check the installed Python 3 version:

```
python3 --version
```

   If the command prints a version number, Python is installed and ready to use. The exact version may differ from the version shown in the following figure.

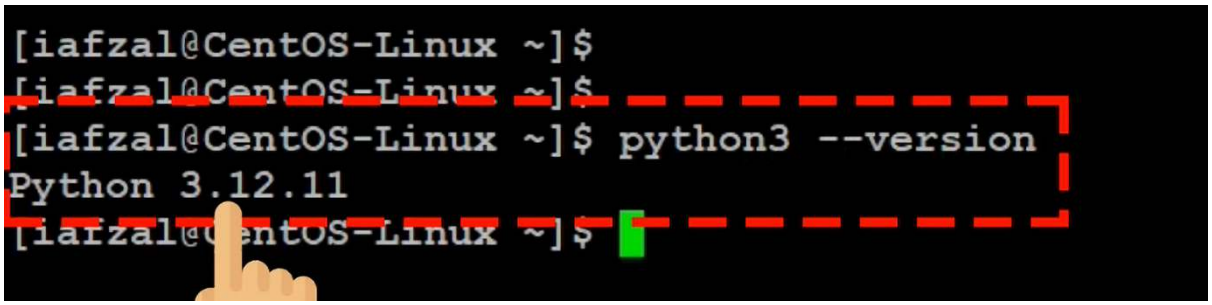


Figure 3.1: Checking the Python 3 version in the CentOS terminal

   The CentOS Stream installation used here includes Python 3 because many system tools and services depend on it, which is why the command may work even though we have not installed Python ourselves. We only need to install it manually if it is missing, or use a different installation method if a newer version or specific development features are required.

2. If Python is missing, install it with `dnf`:

```
sudo dnf install python3 -y
```

   The `-y` option answers yes to the package manager's confirmation prompt. If Python is already installed, `dnf` reports that the package is present and there is nothing to do. That message is also a useful confirmation that we can continue to the next requirement.

3. Install pip for Python 3:

```
sudo dnf install python3-pip -y
```

`pip` is the Python package manager. Python runs the program, while `pip` downloads and installs additional Python libraries from package indexes. Python alone does not install the OpenAI package used by this program, so we need `pip` before we can complete the command-line setup.

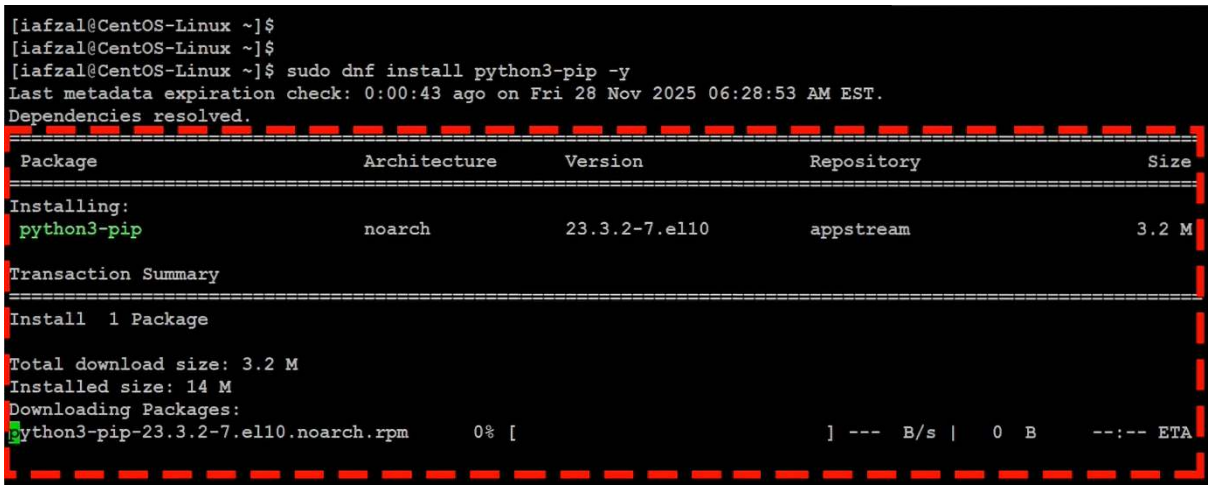


Figure 3.2: Installing the python3-pip package with dnf

4. Confirm that pip is available:

```
python3 -m pip --version
```

Using `python3 -m pip` makes it clear that pip belongs to the same Python 3 interpreter that will run the program.

5. Install or upgrade the OpenAI Python SDK:

```
python3 -m pip install --upgrade openai
```

The `--upgrade` option is important because it tells `pip` to install the newest available SDK release even if an older release is already present. Without it, `pip` may leave an existing version unchanged. An older SDK may not support the methods or features used in the chapter, which can lead to errors or missing functionality later.

Wait while `pip` downloads and installs the required packages. When the installation finishes without an error, Python, `pip`, and the OpenAI SDK are all

ready. We now have the three pieces needed to build the command-line program.

Creating an OpenAI API key

The Python SDK needs an API key before it can send a request to an OpenAI model. The OpenAI Platform is where we manage API keys, check API usage, explore developer documentation, and connect tools or applications to OpenAI models. The key identifies the API project and authorizes requests made from the Linux machine, so it must be treated as a secret.

Now we will create the key, make it available to the CentOS terminal, and verify that our ChatGPT CLI workflow is working properly. We will begin in the browser and then return to Linux to run the first request.

1. Open a web browser and go to `https://platform.openai.com/`. The platform requires an account before its API features can be used. Sign in with one of the options offered on the page, such as email, Google, Microsoft, Apple, or phone. If you do not yet have an account, select the **Sign up** option and create one first.

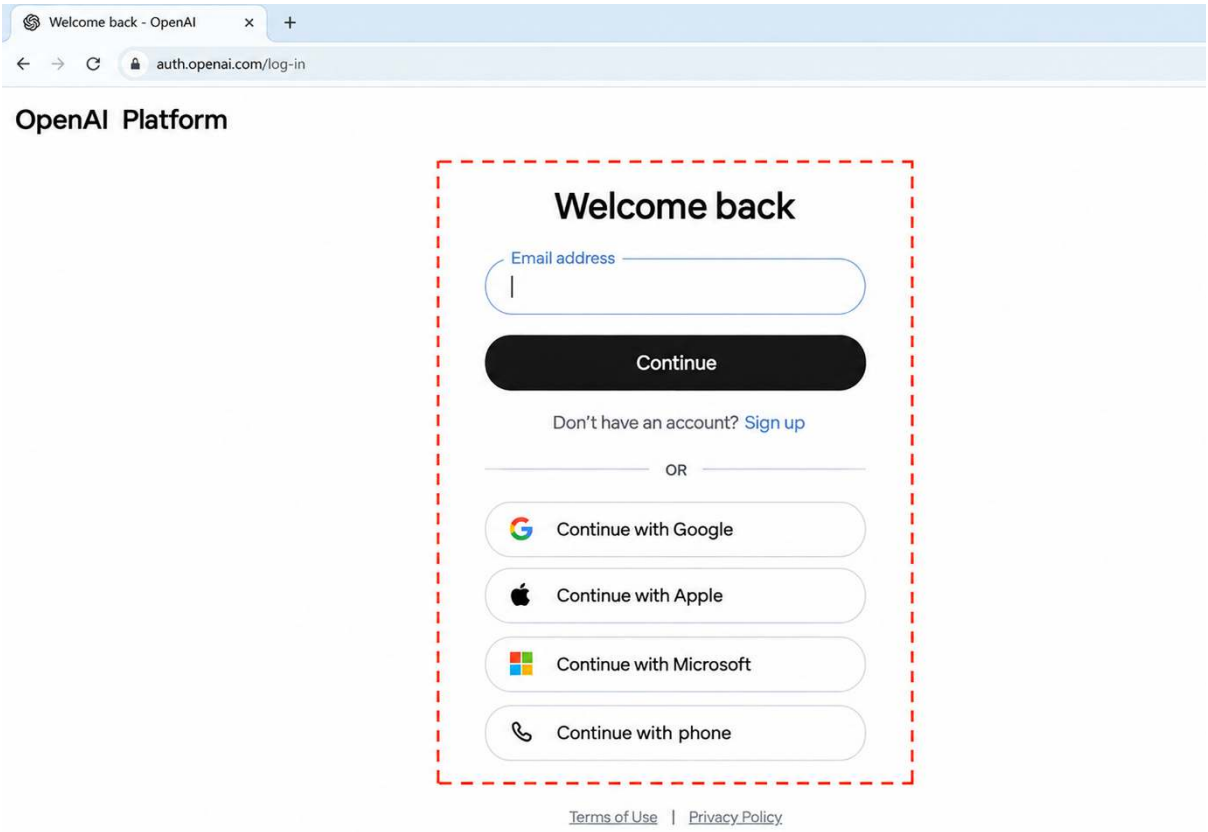


Figure 3.3: OpenAI login page

2. After signing in, return to the main platform dashboard. In the interface, the **Developer quickstart** block contains a **Get started** option. Clicking on it will navigate you to the following screen:

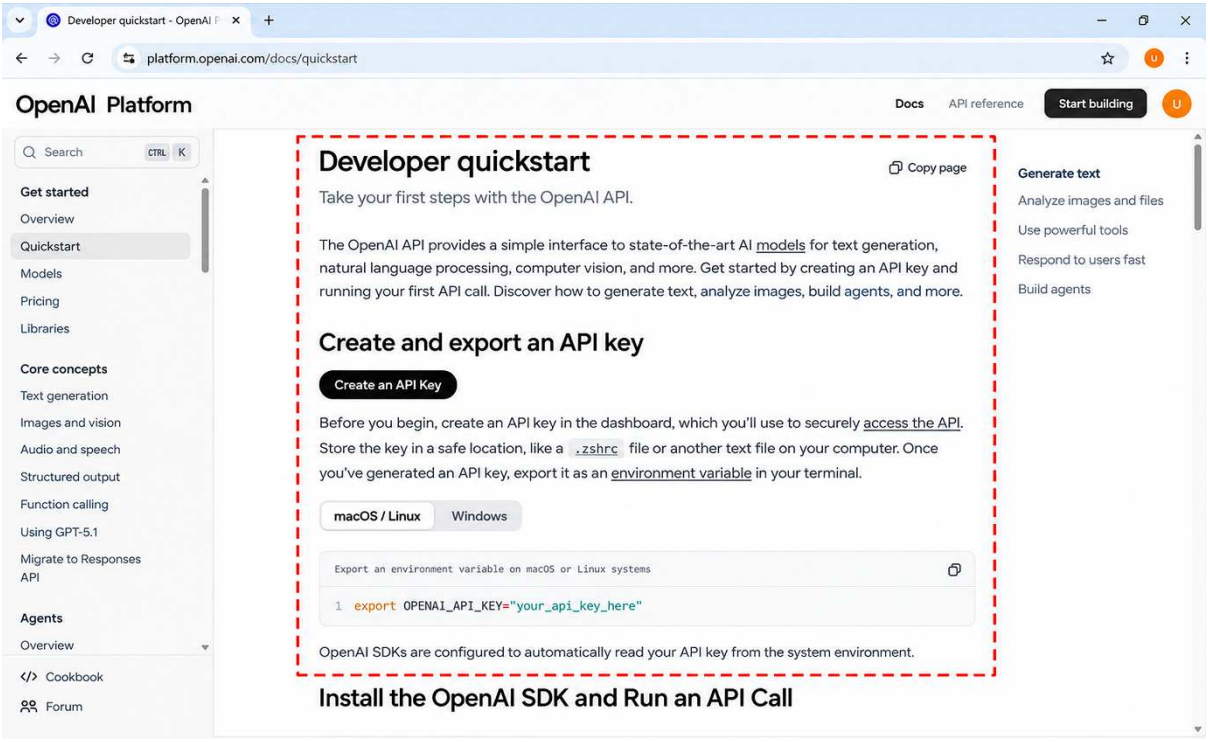


Figure 3.4: The Developer quickstart page

3. On this page, click on **Create an API key**. A pop-up opens asking for the name of the key. You can give any name, for example, `OPENAI_API_KEY`. Leave the rest of the settings as default and click **Create secret key**:

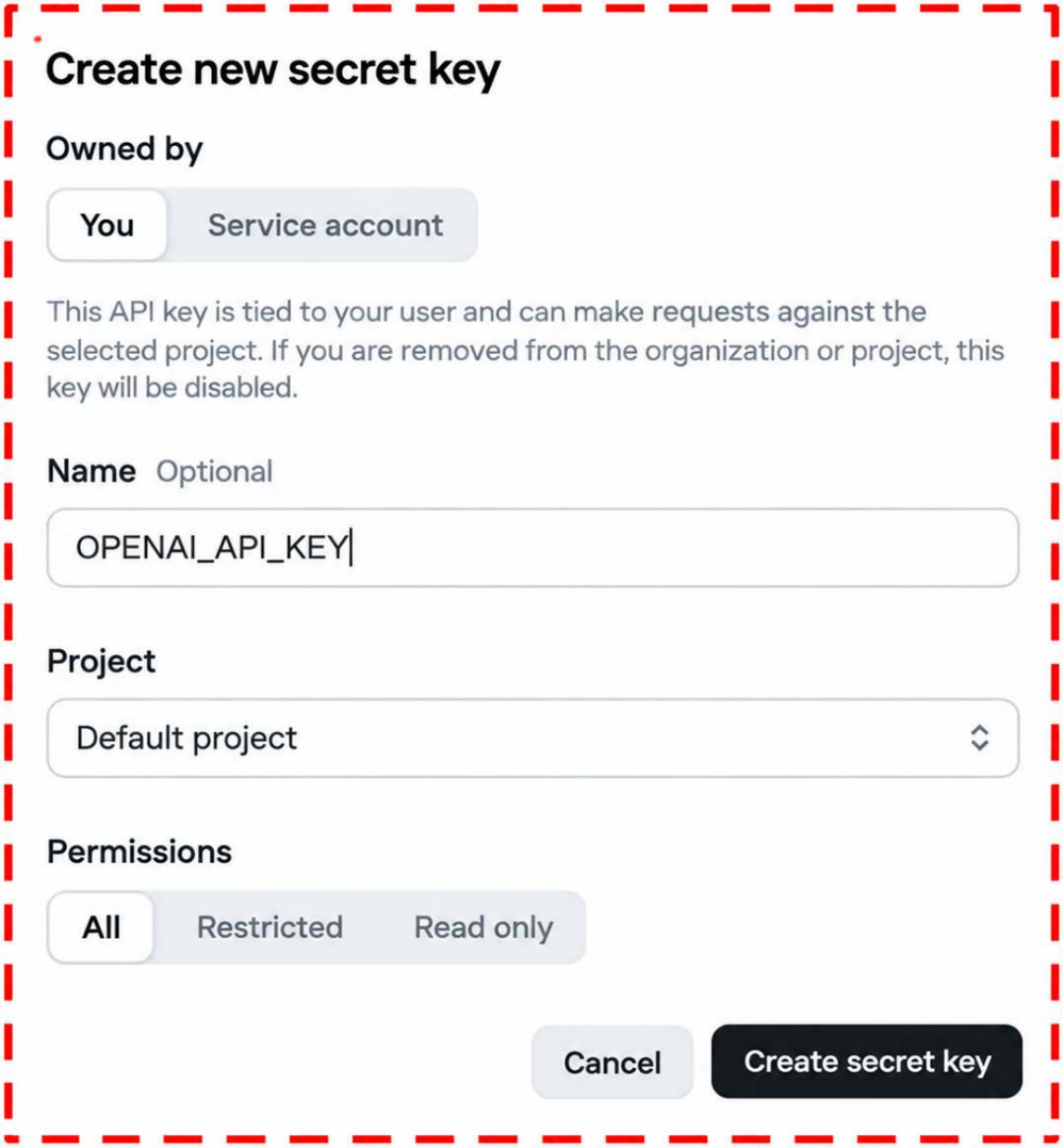


Figure 3.5: Creating an API key in the OpenAI Platform

After selecting **Create secret key**, the secret key appears in a box with a **Copy** button.

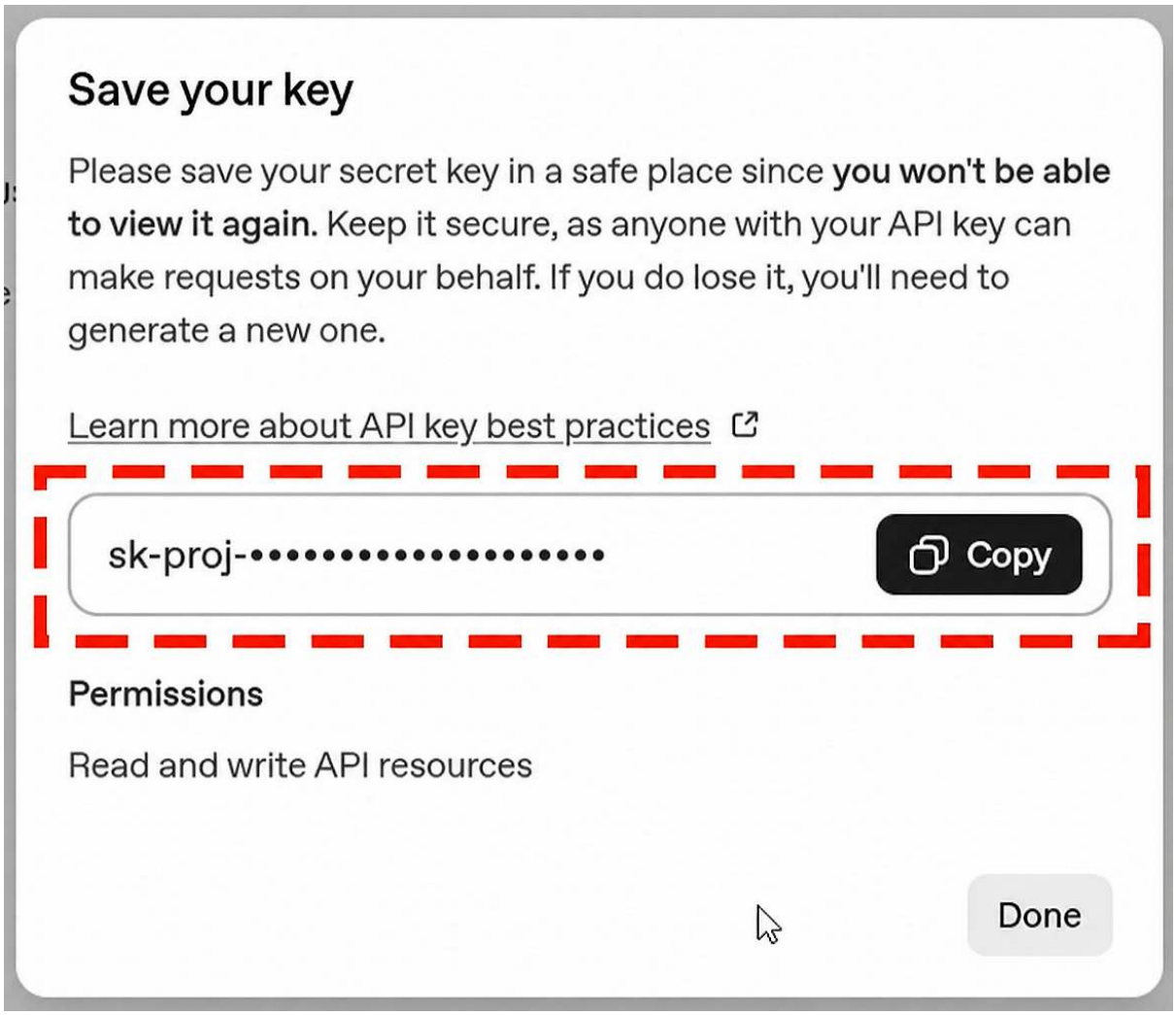


Figure 3.6: The generated secret key

This key is extremely important, and OpenAI displays it only once. Make sure to copy it and store it in a safe place, such as a password manager or another approved secret-management location. If you lose it, you will need to create a new one.

NOTE

Never paste the real key into the Python source file, a screenshot, a shared document, or a source-code repository. If a key is exposed, revoke or rotate it

> immediately and update the Linux environment with the replacement.

Exporting the API key in Linux

Return to the CentOS terminal and export the key as an environment variable. Replace `your_api_key_here` with the secret copied from the OpenAI Platform:

```
export OPENAI_API_KEY="your_api_key_here"
```

This command sets the variable for the current shell session. To verify the configuration without printing the secret, run the following command:

```
python3 -c "import os; print('OPENAI_API_KEY is set' if os.getenv('OPENAI_API_KEY') else
'OPENAI_API_KEY is not set')"
```

If the terminal prints `OPENAI_API_KEY is set`, the current shell can provide the key to the Python SDK. A new shell session will need the variable to be exported again unless it is configured through an approved persistent secret-management method.

For those using Claude instead of ChatGPT

If you prefer to use Anthropic Claude instead of ChatGPT, you can follow a very similar command-line workflow. The Linux administration tasks later in the book do not depend on one specific AI provider. The main difference is the Python SDK, API key environment variable, and the Python code used to send the request.

1. First, install or upgrade Anthropic's Python SDK:

   ```
   python3 -m pip install --upgrade anthropic
   ```

2. Create an API key in the Anthropic Console, then export it in the current Linux shell. Replace the example value with your own key:

   ```
   export ANTHROPIC_API_KEY="your_api_key_here"
   ```

3. Verify that the environment variable is set without displaying the secret:

```
python3 -c "import os; print('ANTHROPIC_API_KEY is set' if os.getenv('ANTHROPIC_API_KEY')
else 'ANTHROPIC_API_KEY is not set')"
```

4. Next, create a small Python file named claude-test.py:

```
vim claude-test.py
```

5. Add the following code:

```
from anthropic import Anthropic

client = Anthropic()
question = input("Enter your question: ")

message = client.messages.create(
    model="claude-sonnet-5",
    max_tokens=1024,
    messages=[
        {"role": "user", "content": question}
    ],
)

print("\n--- AI Response ---\n")
print(message.content[0].text)
```

6. Save the file and run it:

```
python3 claude-test.py
```

7. When prompted, enter a Linux question such as:

```
What is Linux?
```

Claude will return the response directly in the terminal. From this point forward, a reader using Claude can use `claude-test.py` anywhere the book asks the reader to run `chatgpt-test.py`. The Linux commands, Bash scripts, troubleshooting steps, and verification procedures remain the same.

> **NOTE**
>
> Claude model identifiers may change over time. If `claude-sonnet-5` is unavailable, select a currently supported Claude text model in your Anthropic API account and update the `model` value accordingly.

Reviewing the OpenAI Python example

The OpenAI Platform Developer quickstart provides example code in several programming languages. The **Overview** page may initially show JavaScript because it is a common choice for application development. Select **Python** from the language drop-down menu. The Python example imports the SDK, creates a client, selects a model, supplies an input prompt, and prints the returned text. This is the same basic pattern we will now place in a file on the Linux machine.

> **NOTE**
>
> Model names and availability change over time. The screenshot shows `gpt-5.1`, which was current when it was captured. In the scripts below, `gpt-x.x` is a placeholder. Replace it with the exact model ID of a current text model available to your API project.

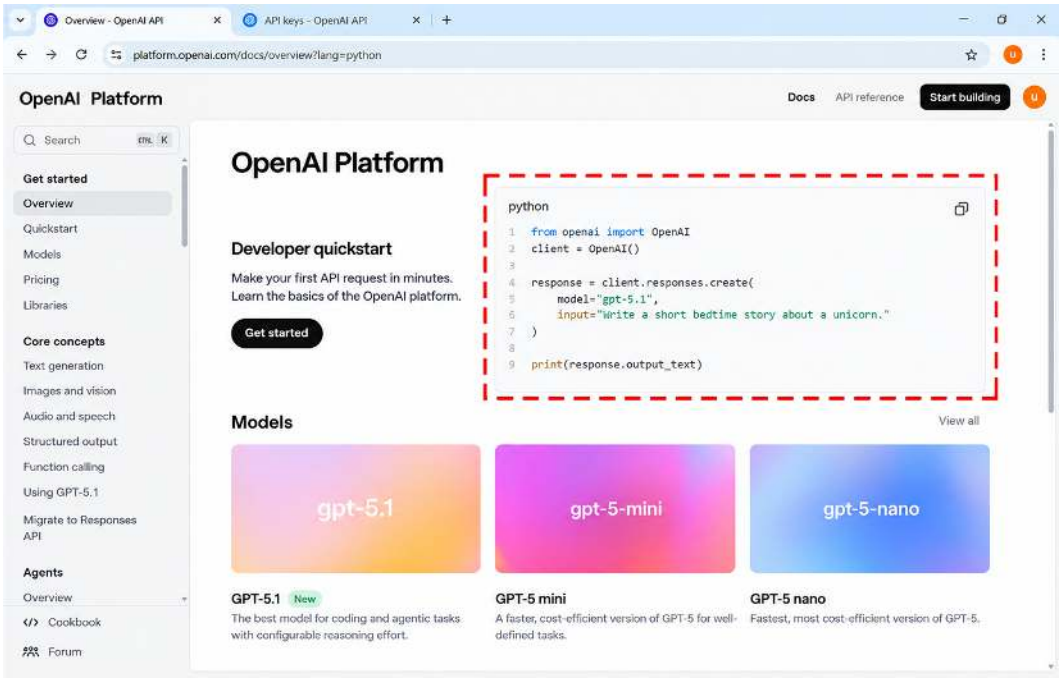


Figure 3.7: Python example and model choices in the OpenAI Platform

At the time of writing, the platform compared `gpt-5.1`, `gpt-5-mini`, and `gpt-5-nano`. The larger model was presented for more advanced work, the mini model as a lighter and more cost-effective option, and the nano model as the fastest option for simple tasks. Treat these descriptions as a snapshot and check the current model catalog before choosing a model for a production workflow.

If you want to explore other models, click on **View all** and open the current model catalog and compare the available options. The exact names shown in the platform can differ by date, API project, and access level, so use the model that fits the task and is available to the project.

Sending the first request from the terminal

Now it is time to return to the CentOS terminal, create a small Python file, and test the complete setup. The file will send one question to the model and print the answer in the terminal.

> **NOTE**
>
> The terminal screenshots in this section were captured from a root shell. The Python scripts do not require elevated privileges and can be run from the regular user account used throughout the procedure.

1. Create a file named chatgpt-test.py in Vim:

```
vim chatgpt-test.py
```

2. Press `i` to enter insert mode and add the following code:

```
from openai import OpenAI

client = OpenAI()

response = client.responses.create(
    model="gpt-x.x",
    input="What is the role of AI in Linux?",
)

print(response.output_text)
```

   The OpenAI import makes the SDK client available to the program. `OpenAI()` creates the client and automatically reads `OPENAI_API_KEY` from the environment. The `responses.create()` call selects the model and sends the question, 'What is the role of AI in Linux?' Finally, `response.output_text` retrieves the generated text so that `print()` can display it in the terminal.

3. Press *Esc*, type `:wq`, and press *Enter* to save the file and leave Vim.
4. Run the program:

```
python3 chatgpt-test.py
```

As soon as the command runs, Python loads the SDK, authenticates with the environment variable, and sends the question to the selected model. The terminal may appear to wait for a few seconds while the request is processed and the response is generated. When the answer is ready, the script prints it directly below the command.

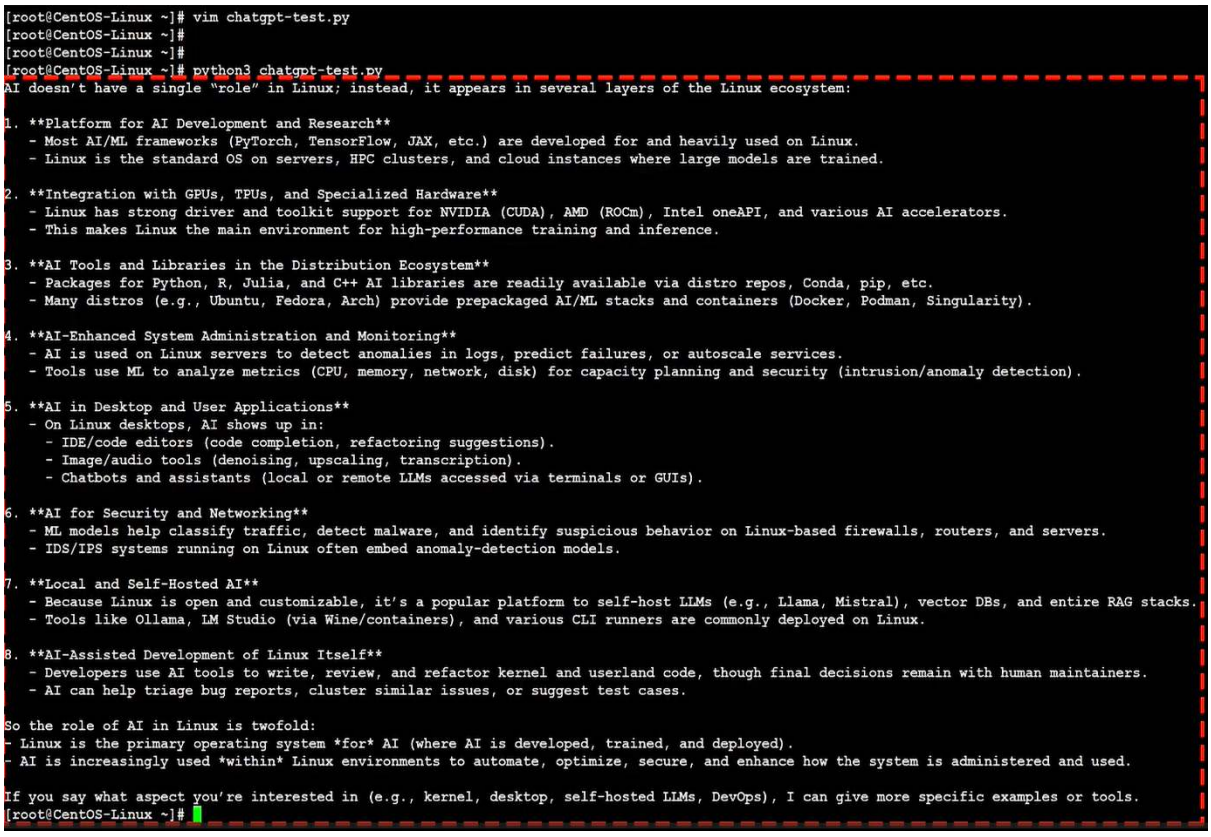


Figure 3.8: Verifying the first OpenAI API request from the CentOS terminal

A clear, detailed response confirms that the Python environment, pip package, API key, network connection, model access, and API request are all working together. At this point, the basic ChatGPT CLI setup is complete.

Making the program interactive

The first version works, but it uses a fixed question. If we want to ask something different, we have to open the file, replace the text, save it, and run it again. We can make the program smarter and more flexible by asking for the question when the script starts and passing that value to the API request.

1. Open the file again:

```
vim chatgpt-test.py
```

2. Replace the original contents with the following version:

```
from openai import OpenAI

client = OpenAI()
question = input("Enter your question: ")

response = client.responses.create(
    model="gpt-x.x",
    input=question,
)

print("\n--- AI Response ---\n")
print(response.output_text)
```

Here, the built-in `input()` function displays `Enter your question:` when the program starts. Whatever is entered is stored in the `question` variable. The API call then uses that variable instead of the fixed sentence from the first version, so the script sends a new prompt each time it runs. The extra `print` statement places a clear `AI response` heading before the returned text, making the terminal output easier to read.

3. Save the file, leave Vim, and run the updated program:

```
python3 chatgpt-test.py
```

4. When the program asks for a question, enter a prompt such as the following and press *Enter*:

```
What is Linux?
```

The script sends exactly that question to the model and waits for the response. After a moment, the answer appears below the `AI Response` heading. The question variable now removes the need to reopen and edit the file for every prompt.

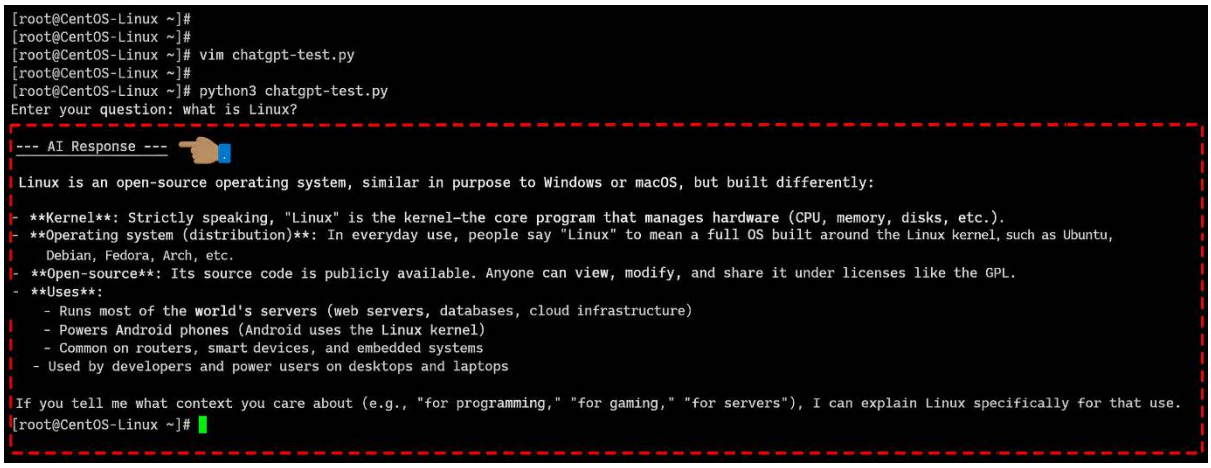


Figure 3.9: Running the interactive command-line assistant

From now on, run the same file, type any question you want to ask, and receive the response directly inside the CentOS terminal. We now have a simple, reusable command-line assistant: run it, type the question, and read the answer without leaving the terminal.

Summary

In this chapter, we introduced the command-line workflow referred to as ChatGPT CLI in this book. We saw how it can reduce context switching for Linux users by bringing explanations, troubleshooting help, command guidance, and script generation into the terminal. We also emphasized that AI-generated commands and scripts must be reviewed before they are run.

We then verified Python, installed pip and the OpenAI Python SDK, created an API key, and made the key available through the `OPENAI_API_KEY` environment variable. Finally, we built a Python program that sent a request through the Responses API and improved it so a new question can be entered each time the program runs. We also configured Claude as an alternative by installing the Anthropic Python SDK, exporting `ANTHROPIC_API_KEY`, and creating `claude-test.py`.

The Linux environment can now communicate with an OpenAI or Anthropic model from the command line. In the next chapter, we will use this foundation for practical system-administration tasks such as generating Bash scripts, explaining command output, analyzing logs, and building reusable automation.

Knowledge check

Test your understanding of the concepts covered in this chapter.

1. What does CLI stand for?
    A. Configured library input
    B. Central Linux installation
    C. Command-line interface
    D. Command link integration
2. Which command checks the installed Python 3 version?
    A. python3 --version
    B. python3 --install
    C. pip3 --python
    D. dnf --python3
3. What is the purpose of pip in this chapter?
    A. To configure the CentOS network
    B. To install Python packages such as the OpenAI SDK
    C. To create an OpenAI account
    D. To start the SSH service
4. Why is OPENAI_API_KEY used instead of placing the secret directly in the Python file?
    A. It removes the need for an API account
    B. It changes the Linux hostname
    C. It installs the selected model
    D. It keeps the secret out of source code
5. Which method sends the prompt in the Python example?
    A. client.responses.create()
    B. client.files.upload()
    C. python3.input()
    D. dnf.responses.create()
6. What makes the second version of `chatgpt-test.py` interactive?
    A. It uses `input()` to capture a question when the script runs

B. It stores the API key in the source file

C. It runs the script as root

D. It opens the OpenAI Platform in a browser

Answers

1. **c. Command-line interface**. A CLI accepts typed commands and text instead of relying on a graphical interface.
2. **a. python3 --version**. The command prints the installed Python 3 version.
3. **b. To install Python packages such as the OpenAI SDK**. `pip` is the Python package manager.
4. **d. It keeps the secret out of source code**. The SDK reads the key from the environment instead.
5. **a. client.responses.create()**. This call sends the input to the selected model through the Responses API.
6. **a. It uses input()** to capture a question when the script runs. The stored value is passed to the API request.

4

Using ChatGPT CLI for Linux Administration

In the previous chapter, we built a small Python program that lets us send prompts to an OpenAI model from the Linux terminal. Now we will put that command-line assistant to work. Instead of asking only general questions, we will use it to create scripts, investigate errors, schedule recurring tasks, interpret logs, review security recommendations, and automate user creation.

These are practical system-administration tasks, and several of them can change the machine. AI can provide a useful starting point, especially when you are still learning Bash syntax, but it does not know the full purpose, access policy, or current state of your system. Throughout the chapter, we will slow down long enough to inspect the generated response, test it in the lab, and verify the result before treating it as complete.

In this chapter, we will cover the following topics:

- Generating and reviewing a Bash system-health script
- Troubleshooting a file-permission error
- Creating and scheduling a disk-monitoring cron job
- Interpreting system and kernel logs with AI assistance
- Reviewing security-hardening recommendations safely
- Creating and verifying a user-account script

Let's get started!

Technical requirements

To follow the hands-on procedures in this chapter, make sure the following are available:

- The CentOS Stream virtual machine prepared in *Chapter 2*
- A working internet connection
- The command-line assistant configured in *Chapter 3*: `chatgpt-test.py` with `OPENAI_API_KEY`, or `claude-test.py` with `ANTHROPIC_API_KEY`
- A text editor such as Vim
- Enough API access or credits to run the example prompts

NOTE

Following a safe AI-assisted workflow

The fastest way to use AI is not to run the first answer immediately. A safer rhythm is to ask, inspect, test, execute, and verify. Begin with a precise prompt. Read the entire response and identify every command, path, username, package, and privilege it uses. Test the least destructive part first, preferably in the virtual machine. Use sudo only for the individual operations that need it, and then verify the outcome with an independent Linux command.

When a response proposes several fixes, treat them as alternatives rather than a checklist to run from top to bottom. The correct choice depends on what the system is meant to do. If a command modifies authentication, networking, the firewall, SELinux, user accounts, or system files, make sure you understand the effect and have a rollback path before continuing.

Generating a system-health script

Many Linux users, especially beginners or busy administrators, do not remember every command or Bash scripting rule. Writing a script manually can take time, require troubleshooting, and introduce mistakes. We will begin with a small script that collects four useful signals from the Linux machine: CPU load, memory usage, disk usage, and network connectivity. This is a good first example of how a plain-English request can reduce that effort and produce a more consistent starting point.

Instead of typing a long script from memory, run the Python assistant created in *Chapter 3* and describe the required checks in simple English. A user who is

still learning Bash can focus first on what the script must do, then study the generated commands and syntax before using them.

NOTE

If you created `claude-test.py` in *Chapter 3*, you can run `python3 claude-test.py` instead of `python3 chatgpt-test.py` throughout this chapter. Enter the same prompts shown in the examples. The generated Linux commands or scripts may vary slightly, so review them before running them and verify the result using the standard Linux commands shown in the chapter.

Run the command-line assistant:

```
python3 chatgpt-test.py
```

At the prompt, enter a request that names the checks and the target shell clearly:

```
Create a Bash script for Linux that displays CPU load, memory usage, disk usage, and network
connectivity. Add comments that explain each section.
```

After the prompt is submitted, the assistant sends the request to the API and returns a generated Bash script after a short wait.

```
#!/usr/bin/env bash

echo "=== System Resources ==="
# CPU load
uptime

echo
echo "=== Memory Usage ==="
# Memory usage
free -h

echo
echo "=== Disk Usage ==="
# Disk usage of all mounted filesystems
df -h

echo
echo "=== Network Connectivity ==="
# Change 8.8.8.8 to another IP address or host if needed
if ping -c 1 -W 2 8.8.8.8 >/dev/null 2>&1; then
    echo "Network: Online (able to reach 8.8.8.8)"
```

```
else
    echo "Network: Offline or cannot reach 8.8.8.8"
fi
```

The exact answer may vary, even when the prompt is the same. Do not copy it immediately. Read it from the beginning, confirm that it matches the request, and identify what each command is doing. In this example, the response uses `uptime`, `free -h`, `df -h`, and `ping`.

`uptime` reports how long the machine has been running and includes load averages. `free -h` summarizes memory in human-readable units, while `df -h` reports filesystem usage. The `ping` test sends one packet and waits up to two seconds. A failed ping can mean the host is unreachable or ICMP traffic is blocked, so it is only one connectivity signal rather than proof that the entire network is offline.

The comments beginning with `#` separate the checks and explain the purpose of each part. That is particularly useful for beginners because the output is not only automation; it is also a script that can be read and learned from. The generated response may also remind us to save the code in a `.sh` file, make it executable, and then run it.

Once we understand the generated script and confirm that it matches what we requested, save the code in a `.sh` file, make it executable, and then run it:

1. Create a file named `check_system.sh`:

   ```
   vim check_system.sh
   ```

2. Paste the reviewed script, then press *Esc*, type `:wq`, and press *Enter*.
3. Make the file executable:

   ```
   chmod +x check_system.sh
   ```

   Newly created files are not automatically executable. `chmod +x` adds execute permission, which tells Linux that this trusted file may be run as a program.

4. Finally, run the script:

   ```
   ./check_system.sh
   ```

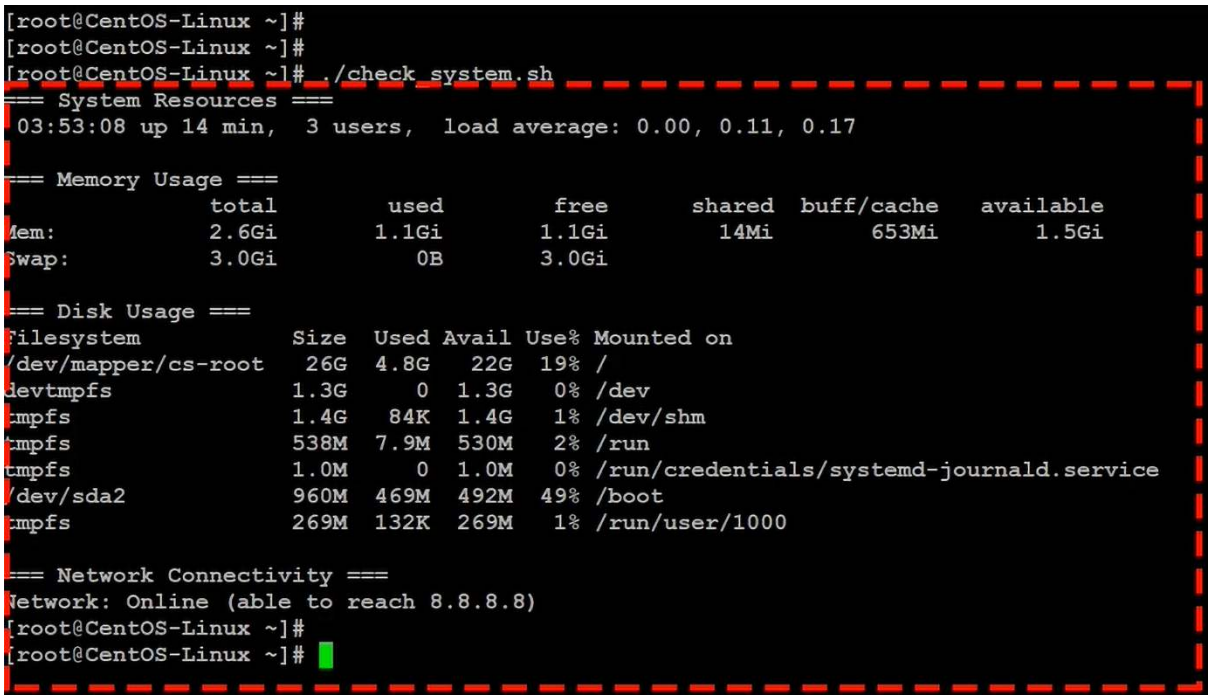


Figure 4.1: Running the AI-generated system-health script

When the script runs, the four sections should appear one after another without requiring us to enter each health-check command manually. Compare the result with what the script is supposed to collect. If a command is unavailable or a path does not match the system, return to the script and adjust it only after understanding the change. We have now turned one plain-English request into a repeatable Linux health check.

Troubleshooting a permission error

Linux commands and their output can feel confusing when we are still learning the system. An error message often tells us what failed but not why. In this example, we will reproduce a Permission denied error, ask the command-line assistant to explain it, and then confirm the diagnosis with Linux commands. Creating the test condition ourselves makes the procedure reproducible instead of assuming that a prepared file already exists.

NOTE

Run this example from a regular user account rather than a root shell. The root user can read the file even after the owner read bit is removed, so the intended `Permission denied` message would not appear.

1. Create a small test file:

```
printf 'Hello test\n' > testfile.txt
```

2. List the current directory to confirm that `testfile.txt` and the `chatgpt-test.py` assistant are both available:

```
ls
```

   The Python file connects the terminal to the OpenAI model, while `testfile.txt` gives us a simple file on which to reproduce the error.

3. Remove the file owner's read permission:

```
chmod u-r testfile.txt
```

4. Try to read the file:

```
cat testfile.txt
```

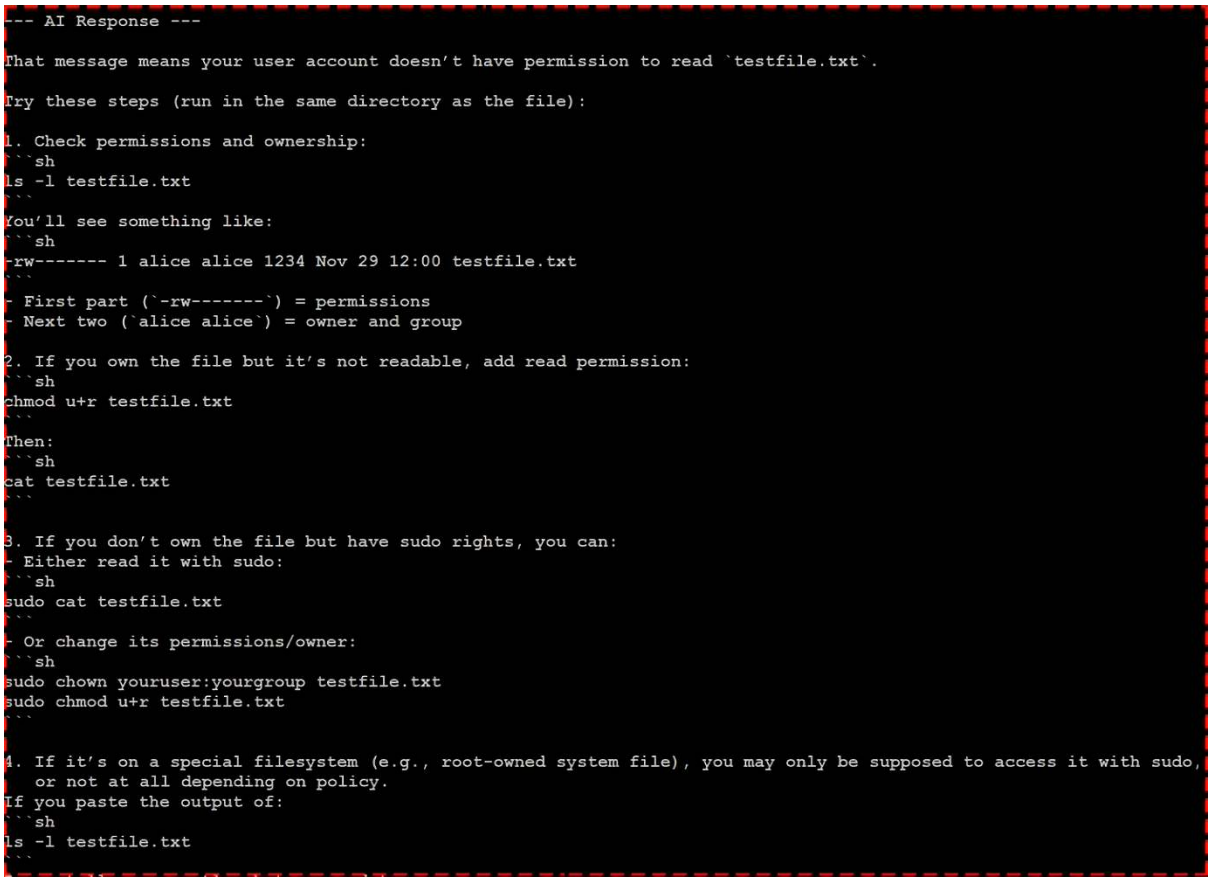
   `cat` is a common Linux command for displaying the contents of a file without opening it in a text editor. Here it returns an error instead of the expected text, which tells us that something is preventing the current user from reading the file.

   The terminal should report the following error:

```
cat: testfile.txt: Permission denied
```

5. Instead of guessing at the cause or searching through unrelated answers online, run the Python assistant. Paste the failing command together with the exact error message so that the model has both the action and its result, then ask it to begin with diagnosis rather than immediately changing the system:

```
python3 chatgpt-test.py
I ran `cat testfile.txt` and received `cat: testfile.txt: Permission denied`. Explain the
likely causes and give me safe commands to inspect the file before changing anything.
```



```
--- AI Response ---

That message means your user account doesn't have permission to read `testfile.txt`.

Try these steps (run in the same directory as the file):

1. Check permissions and ownership:
```sh
ls -l testfile.txt
```
You'll see something like:
```sh
-rw------- 1 alice alice 1234 Nov 29 12:00 testfile.txt
```
- First part (`-rw-------`) = permissions
- Next two (`alice alice`) = owner and group

2. If you own the file but it's not readable, add read permission:
```sh
chmod u+r testfile.txt
```
Then:
```sh
cat testfile.txt
```

3. If you don't own the file but have sudo rights, you can:
- Either read it with sudo:
```sh
sudo cat testfile.txt
```
- Or change its permissions/owner:
```sh
sudo chown youruser:yourgroup testfile.txt
sudo chmod u+r testfile.txt
```

4. If it's on a special filesystem (e.g., root-owned system file), you may only be supposed to access it with sudo,
   or not at all depending on policy.
If you paste the output of:
```sh
ls -l testfile.txt
```
```

Figure 4.2: ChatGPT CLI response to the permission error

The response may offer several possible solutions:

- First, it may suggest `ls -l testfile.txt` to check the file's owner and permission bits.
- Second, if the owner lacks read permission, it may recommend `chmod u+r testfile.txt`.
- Third, if the file belongs to another user or genuinely requires elevated access, it may mention `sudo` or a carefully reviewed ownership change with `chown`.
- Finally, if the file is inside a protected system directory, access may depend on that directory and the system policy. These are alternatives to investigate, not four commands to run automatically.

6. Begin with the least invasive diagnostic command:

```
ls -l testfile.txt
```

In this lab, the `r` permission is missing. Because we created the file as the current user and deliberately removed that user's read bit, the error has a clear cause.

7. The minimum correction is to restore that one permission:

```
chmod u+r testfile.txt
```

8. Now retry the original command:

```
cat testfile.txt
```

The content should now appear.

```
ls -l testfile.txt
```
I can tell you exactly what command to run.
[iafzal@CentOS-Linux ~]$
[iafzal@CentOS-Linux ~]$
[iafzal@CentOS-Linux ~]$ ls -l testfile.txt
--w-------. 1 iafzal iafzal 11 Nov 29 08:17 testfile.txt
[iafzal@CentOS-Linux ~]$
[iafzal@CentOS-Linux ~]$
[iafzal@CentOS-Linux ~]$ chmod u+r testfile.txt
[iafzal@CentOS-Linux ~]$
[iafzal@CentOS-Linux ~]$
[iafzal@CentOS-Linux ~]$ cat testfile.txt
Hello test
[iafzal@CentOS-Linux ~]$

Figure 4.3: Inspecting the file and restoring owner read permission

The same pattern can be used for other Linux problems. When a command fails, behaves unexpectedly, or produces output that is difficult to understand, provide the exact command and message, ask for an explanation, inspect the system state, and apply only the fix supported by the evidence.

NOTE

We did not use sudo or change ownership because the evidence showed that the current user already owned the file. This is an important troubleshooting habit: use AI to generate hypotheses, then choose the smallest change supported by the system's actual state.

Creating and scheduling a disk-monitoring job

A script becomes more useful when Linux can run it automatically. In this section, we will ask the assistant for a disk-usage script that adds a timestamp to every entry and for the exact cron line that runs it every two minutes. The

short interval lets us watch the first entry appear and then confirm that the schedule continues without manual input. In a real environment, the interval would be chosen according to the monitoring requirement.

1. CentOS uses the Cronie implementation of cron. Before scheduling the job, make sure the package is installed and the `crond` service is running:

```
sudo dnf install cronie -y
sudo systemctl enable --now crond
systemctl status crond --no-pager
```

2. Run the command-line assistant and enter the following prompt:

```
python3 chatgpt-test.py
Create a Bash script that checks disk usage and writes the result with a timestamp to a log file in my home directory. Then provide the exact cron entry to run the script every two minutes.
```

   The location named in this prompt is deliberate. If the log directory is not specified, a generated answer may place the file at `/var/log/diskmonitor.log`. This looks reasonable because `/var/log` is the standard system log location, but a normal user usually cannot create or write files there. If we kept that path and ran the job from the user's crontab, the script would fail with a permission error. This workflow therefore keeps the log at `$HOME/diskmonitor.log`, where it can be written without a root session.

   This is an important part of working with generated scripts: even when the logic is correct, every path and privilege assumption must match the account that will run the code.

   A concise version of the generated script is as follows:

```
#!/usr/bin/env bash

LOGFILE="$HOME/diskmonitor.log"

{
    echo "===== $(date '+%Y-%m-%d %H:%M:%S') ====="
    df -h
    echo
} >> "$LOGFILE"
```

The braces group the timestamp, disk report, and blank line so that one redirection appends the complete entry to the log. Because `$HOME` belongs to the user whose crontab runs the job, no root permission is required.

3. Create `diskmonitor.sh`, paste the reviewed script, and save it:

```
vim diskmonitor.sh
```

4. Make the script executable:

```
chmod u+x diskmonitor.sh
```

5. Verify the executable permission before continuing:

```
ls -l diskmonitor.sh
```

The execute bit should now be present. The `diskmonitor.log` file will not exist until the script runs for the first time, so its absence at this point is expected.

6. Run the script manually before scheduling it:

```
./diskmonitor.sh
cat "$HOME/diskmonitor.log"
```

7. The manual test separates script problems from scheduling problems. If the first log entry appears, find the absolute path of the script:

```
pwd
```

In this example, the path is `/home/iafzal/diskmonitor.sh`. If your username or location differs, use the path printed by `pwd` in the next step.

8. Open the current user's crontab:

```
crontab -e
```

9. Add the following entry, replacing the path if needed:

```
*/2 * * * * /home/iafzal/diskmonitor.sh
```

The five scheduling fields represent minute, hour, day of month, month, and day of week. `*/2` in the minute field means every two minutes.

10. Save the crontab and confirm the installed entry:

```
crontab -l
```

11. Wait for the first two-minute interval, then display the log:

```
cat "$HOME/diskmonitor.log"
```

The first scheduled run should add a timestamp followed by a fresh `df -h` report. This confirms that cron executed the script automatically. Wait for another four minutes and display the file again.

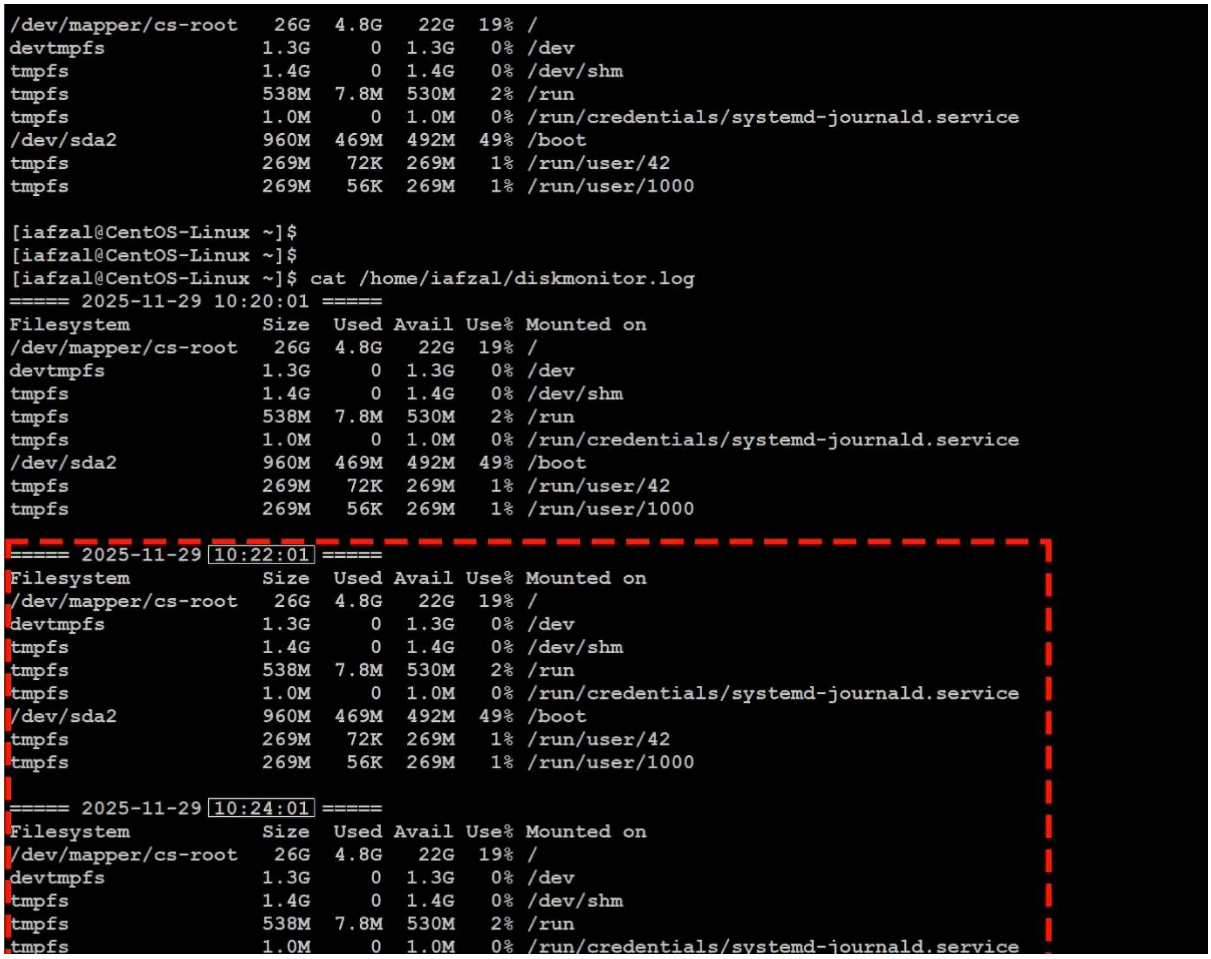


Figure 4.4: Confirming repeated disk-usage entries from the cron job

Two more entries, each separated by approximately two minutes, confirm that the job is continuing to run according to the schedule

rather than succeeding only once.

> **NOTE**
>
> The two-minute interval is intended only for this lab and will continue growing the log until it is removed. When the test is complete, run `crontab -e`, delete the `diskmonitor.sh` line, save the file, and use `crontab -l` to confirm that the entry is gone.

Interpreting system logs with AI

System and application logs preserve the clues that Linux services, the kernel, and background processes leave behind. They are especially useful when something is not working and we need to understand what happened behind the scenes. The difficulty is that log output can be dense, repetitive, and full of unfamiliar component names. AI can help translate a message into simpler language, but the explanation is only as useful as the context supplied with the log.

1. Start by viewing the most recent journal entries at error priority:

```
sudo journalctl -p err -n 50
```

`journalctl` reads the `systemd` journal. `-p err` limits the result to error priority and more severe messages, and `-n 50` returns the most recent 50 matching entries.

2. To look specifically for kernel messages containing the word error, run:

```
sudo dmesg -T | grep -i "error"
```

`dmesg` reports kernel-level and hardware-related messages, which a normal user may not be allowed to view completely. `sudo` supplies the required privilege for this read-only command, while the pipe sends the output to `grep` so that only lines containing `error` are shown. This produces a shorter set of messages to investigate.

> **NOTE**
>
> Before sending any log text to an external AI service, remove secrets and information that should not leave the system. This can include tokens, passwords, email addresses, usernames, hostnames, IP addresses, customer data, private paths, and configuration values. Share the smallest excerpt that contains the error and enough surrounding context to interpret it.

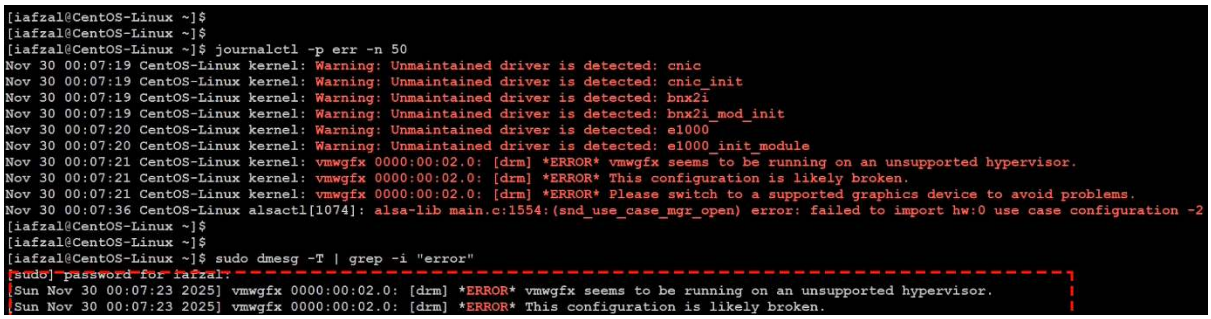


Figure 4.5: Filtering kernel messages for errors in the virtual machine

The lab output contains repeated `vmwgfx` messages stating that the driver appears to be running on an unsupported hypervisor.

3. Instead of pasting the line alone, include the operating system, hypervisor, symptoms, and a request for safe verification steps:

```
I am running CentOS Stream in VirtualBox. The desktop is working, but `dmesg` reports:
`vmwgfx seems to be running on an unsupported hypervisor`. Explain the likely cause in simple
terms. Before suggesting a fix, list safe commands or VM settings I can check. Do not assume
that I should disable a driver.
```

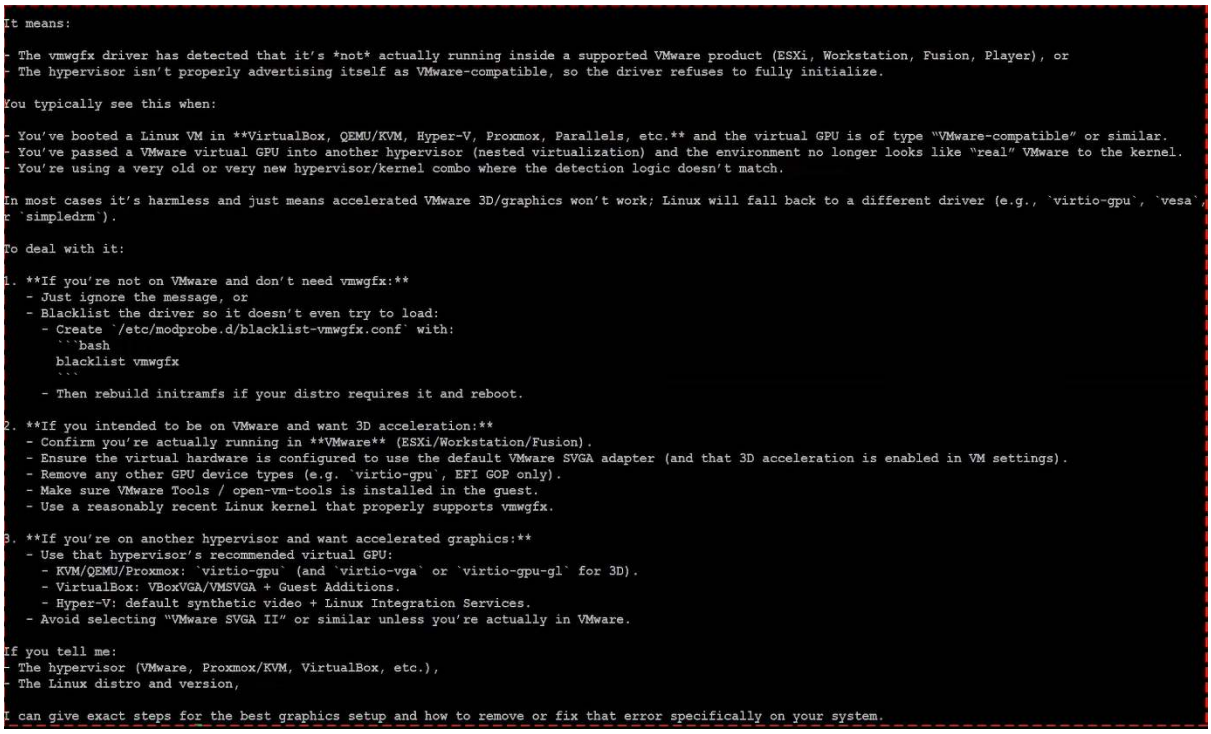


Figure 4.6: ChatGPT CLI interpretation of the vmwgfx message

The response explains that `vmwgfx` is a VMware-compatible graphics driver and that the message can appear when the guest loads it without detecting the VMware hardware environment it expects. In a virtual machine, that can affect VMware-specific graphics features or 3D acceleration. The desktop may continue working because the guest can fall back to another display path, but that does not make the message safe to ignore without checking the actual hypervisor and symptoms.

NOTE

The message is consistent with the guest loading a VMware-compatible graphics driver while running under a different virtualization environment. That explanation is a hypothesis, not permission to ignore the error automatically. Confirm the selected VirtualBox graphics controller, check whether the display has real symptoms, and compare any proposed change with the

> documentation for the installed CentOS and VirtualBox versions. If the desktop is stable, record the finding and avoid changing drivers merely to silence a message.

Repeat the process with the next distinct error line. In our lab, the second and third entries lead to similar explanations about `vmwgfx`, virtual graphics, and 3D acceleration. Seeing the same diagnosis more than once helps us recognize that the messages belong to one repeated issue rather than three unrelated failures.

The generated response may also include distribution-specific fixes, such as `apt` commands for Ubuntu or Debian, Fedora-specific commands, or instructions to disable or blacklist the driver. This machine runs `CentOS Stream`, so do not copy commands intended for another distribution, and do not disable a driver merely because it appears in a suggested fix.

The same approach applies to other logs. Copy the exact message, identify the distribution and version, name the hypervisor or application, explain what changed before the error, and say what has already been tested. Ask the model to separate likely causes from verification steps. This reduces manual searching and makes unfamiliar logs easier to understand, especially for beginners, but the system's observed behavior remains the final evidence and we still decide what to run.

Reviewing security-hardening recommendations

Security hardening is an excellent example of why AI output must be reviewed in context. A server that accepts remote SSH connections, a local workstation, and a disposable lab VM do not need identical settings. A generic hardening script can close the administrator's access, disable a required service, or replace a working firewall policy.

A broad request could ask the assistant for a checklist and a separate script for every recommendation. Before generating executable scripts, begin with a more reviewable prompt:

```
python3 chatgpt-test.py
Create a basic security-hardening checklist for CentOS Stream. Group the recommendations by risk
area. For each item, explain what it changes, what could break, how to verify it, and how to roll it
back. Do not assume that every recommendation should be applied automatically.
```

The generated checklist covers ten useful areas:

- Keep the operating system and packages updated, because outdated software can contain known vulnerabilities that attackers are able to exploit.
- Configure `firewalld` so that only the services or inbound ports the machine actually requires are exposed, while other inbound traffic remains blocked.
- Harden `SSH` by reviewing direct root access, preferring key-based authentication where appropriate, limiting authentication attempts, and evaluating any port change in the context of the environment.
- Review user accounts and password policy by disabling unused accounts and validating password strength, expiration, and history requirements.
- Disable services the system does not need, such as `NFS` or `FTP` when they have no defined purpose, to reduce the number of exposed components.
- Keep `SELinux` in enforcing mode unless a validated exception is required, rather than turning off an important security control to avoid investigating a denial.
- Collect and audit security-relevant logs so that authentication attempts, privileged actions, and other important events can be investigated later.
- Review filesystem permissions on sensitive and shared locations, including configuration under `/etc` and temporary locations such as `/tmp` and `/var/tmp`.
- Use `sudo` instead of routine direct root logins and record privileged activity so that administrative changes remain attributable.
- Use `AIDE` to create a file-integrity baseline and detect when monitored files or attributes change unexpectedly.

The generated answer may also provide a separate script for each recommendation, complete with filenames and execution instructions. That

makes the output look ready to use, but each script still needs an individual review because the controls affect different parts of the system.

> **NOTE**
>
> Do not copy and run the complete set of generated hardening scripts on a real system. Review one control at a time in a snapshot or disposable VM. Confirm the machine's purpose, preserve an active recovery path, record the current configuration, and test the rollback before applying authentication, firewall, SSH, SELinux, or service changes.

Trying one focused control with AIDE

To keep the hands-on example focused, we will try only the final checklist item, `AIDE`. In simple terms, `AIDE` works like a guard for selected system files: it records a known baseline and later reports what has changed. A reported difference does not prove that an attacker or malware changed the file, because legitimate updates also modify the system. It tells us where to investigate.

Create `10_install_aide.sh` with the following reviewed script:

```
#!/usr/bin/env bash
set -e

if [[ $EUID -ne 0 ]]; then
    echo "Run this script with sudo." >&2
    exit 1
fi

echo "[*] Installing AIDE..."
dnf install -y aide

echo "[*] Initializing the AIDE database..."
aide --init

echo "[*] Activating the new database..."
mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz

echo "[*] Running the first integrity check..."
aide --check
```

Make the file executable and run it with sudo:

```
chmod +x 10_install_aide.sh
sudo ./10_install_aide.sh
```

When the script starts, the terminal shows the `AIDE` package being downloaded and installed, followed by database initialization.

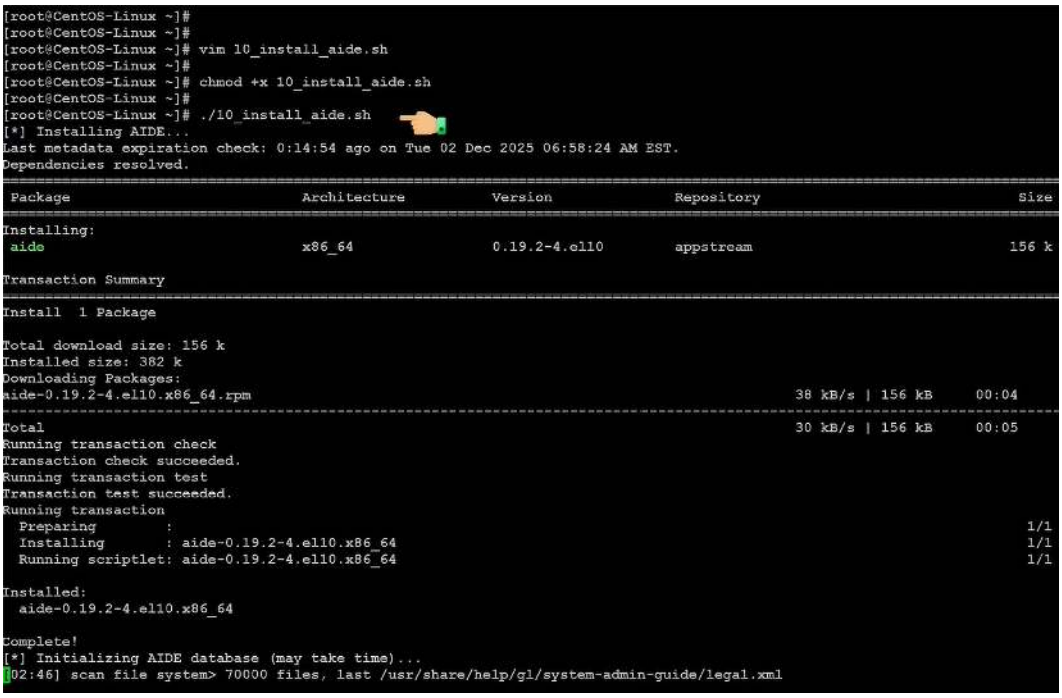


Figure 4.7: Installing AIDE and creating the initial integrity baseline

Initialization can take time because `AIDE` reads many files and records their attributes. The script then moves the newly created database into the active location and performs an initial comparison. These progress messages show which stage is running, but a completed command is not proof that monitoring is fully configured. Review the output and any reported differences.

Creating a user-account script safely

The final example asks the assistant for a user-account creation script that can be used directly in the Linux lab. It automates a common administrative sequence: accept a username, check whether the account exists, create it if needed, set a temporary password, require a password change at first login, and record the action. Because user management changes authentication and requires elevated privileges, every part of the generated script must be reviewed before it is run.

Ask the assistant for the workflow without putting a shared default password in the source code:

1. Run the command-line assistant:

```
python3 chatgpt-test.py
Create a Bash script for CentOS Stream that accepts a username, checks whether the account
already exists, creates it if needed, prompts the administrator to set a temporary password
interactively, forces a password change at first login, and logs the administrative actions
to `/var/log/usercreate.log`. Do not hardcode or log a password.
```

   The generated answer includes the requested username argument, existing-user check, account creation, temporary-password step, forced password change, and action log. It may also suggest a filename, the `chmod` command needed to make the file executable, and an example showing how to pass the username. After reviewing the output and removing the unsafe hardcoded-password approach, the complete script is as follows:

```
#!/usr/bin/env bash
set -u

LOGFILE="/var/log/usercreate.log"

if [[ $EUID -ne 0 ]]; then
    echo "Run this script with sudo." >&2
    exit 1
fi

if [[ $# -ne 1 ]]; then
    echo "Usage: $0 <username>" >&2
    exit 1
fi

USERNAME="$1"

log() {
    printf '%s : %s\n' "$(date '+%Y-%m-%d %H:%M:%S')" "$1" | tee -a "$LOGFILE"
}

if id "$USERNAME" &>/dev/null; then
    log "User '$USERNAME' already exists. No changes made."
    exit 0
fi

if useradd "$USERNAME"; then
    log "User '$USERNAME' created successfully."
```

```
else
    log "ERROR: Failed to create user '$USERNAME'."
    exit 1
fi

echo "Set a temporary password for '$USERNAME'."
if passwd "$USERNAME"; then
    log "A temporary password was set for '$USERNAME'."
else
    log "ERROR: Failed to set a password for '$USERNAME'."
    exit 1
fi

if chage -d 0 "$USERNAME"; then
    log "User '$USERNAME' must change the password at next login."
else
    log "ERROR: Failed to require a password change for '$USERNAME'."
    exit 1
fi

log "User creation process for '$USERNAME' completed."
```

The script still needs root privileges because it creates an account and writes to `/var/log`, but the administrator can grant those privileges to this one execution with sudo instead of remaining logged in as root.

This keeps the privileged boundary clear. The script file can be created and reviewed as the normal user; only the account-management execution receives `sudo` privileges.

2. Create `create_user.sh`, paste the reviewed script, and save it:

```
vim create_user.sh
```

3. Make the script executable:

```
chmod +x create_user.sh
```

4. Run it with the test username `anderson`:

```
sudo ./create_user.sh anderson
```

After the username is supplied, the script creates the account and `passwd` prompts for a temporary password. Enter a value that follows the lab system's password policy. The characters are not displayed, which prevents the password from appearing on screen, and the

command asks for the value again to confirm it. The script then expires that password so the new user must replace it at first login.

5. Verify the account identity and home directory:

```
id anderson
ls -ld /home/anderson
```

6. Verify the password state and expiry information:

```
sudo passwd -S anderson
sudo chage -l anderson
```

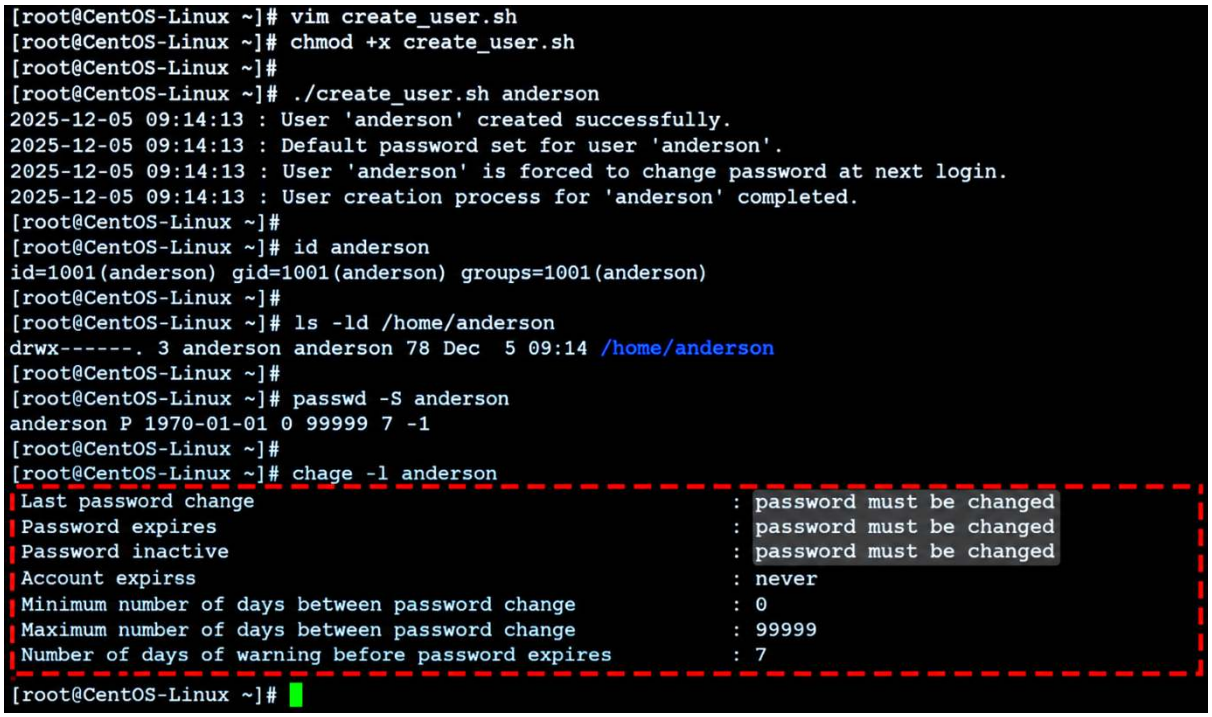


Figure 4.8: Verifying the newly created user and first-login password change

Begin with `id anderson`. A successful result displays the `UID`, `GID`, and group information, confirming that the account exists.

Next, `ls -ld /home/anderson` should show a home directory owned by `anderson` with the expected group and permissions.

Then inspect the password state with `passwd -S`. The letter `P` indicates that a usable password is set, while the remaining fields describe password-age information.

Finally, `chage -l` should report that the password must be changed. Together, these checks confirm that the account, home directory, temporary password, and first-login expiry policy were all created as intended.

This is the complete verification loop: the script reports success, and independent Linux commands confirm each result instead of relying on that message alone.

> **NOTE**
>
> If `anderson` is only a disposable lab account, remove it after verification with `sudo userdel -r anderson`. Check the username carefully before running this command because `-r` also removes the user's home directory and mail spool.

Summary

In this chapter, we moved from a working command-line assistant to practical Linux administration. We generated a Bash health-check script, reproduced and diagnosed a permission error, and created a disk-monitoring script that cron ran on a two-minute schedule. In every case, the AI response was a starting point, while Linux commands and observed output provided the verification.

We also used system and kernel logs to build a context-rich troubleshooting prompt, reviewed a broad security-hardening checklist without applying it blindly, and tried one focused control by installing and initializing AIDE. Finally, we revised a user-creation script so that it checks for an existing account, prompts for a temporary password without hardcoding it, forces a first-login password change, and records the administrative actions.

The key lesson is that AI can accelerate Linux work, but safe administration still depends on understanding scope, privilege, system context, verification, and rollback. In the next chapter, we will examine Shell Genie and use natural-language requests to generate Linux commands for common tasks.

Knowledge check

Test your understanding of the practical workflows covered in this chapter.

1. Why should an AI-generated Bash script be read before it is executed?
    A. To install Bash automatically
    B. To make the terminal prompt change color
    C. To confirm its commands, paths, privileges, and effects
    D. To prevent the file from being saved
2. Which command restores read permission for the owner of `testfile.txt`?
    A. chmod u+r testfile.txt
    B. chmod u-r testfile.txt
    C. chown root testfile.txt
    D. sudo rm testfile.txt
3. Why is the disk-monitoring script tested manually before it is added to cron?
    A. To separate script problems from scheduling problems
    B. To make cron run as root
    C. To delete the log file
    D. To disable the crond service
4. What does `*/2` mean in the minute field of the example cron entry?
    A. Wait two minutes after login
    B. Run at 2:00 only
    C. Run twice per month
    D. Run every two minutes
5. What should be done before sending production log text to an external AI service?
    A. Convert every line to uppercase
    B. Remove secrets and sensitive information and share only the needed excerpt
    C. Run the terminal as root
    D. Disable systemd-journald
6. Why should broad AI-generated hardening scripts not be run as a single batch?

A. CentOS cannot run Bash scripts
B. They can break required services, networking, or administrative access
C. AIDE replaces the Linux kernel
D. firewalld works only in a browser

7. What is the purpose of `aide --init` in the example?
   A. To edit the crontab
   B. To add a Linux user
   C. To create an initial file-integrity database
   D. To reset the firewall
8. Why does the revised user-creation script use `passwd` interactively?
   A. To avoid hardcoding or logging a shared plaintext password
   B. To remove the user's home directory
   C. To run the script without privileges
   D. To disable the account permanently

Answers

1. **c. To confirm its commands, paths, privileges, and effects**. Generated code must match the actual system before it is trusted.
2. **a. chmod u+r testfile.txt**. It adds read permission for the file owner.
3. **a. To separate script problems from scheduling problems**. A successful manual run confirms that the script itself can write the expected log entry.
4. **d. Run every two minutes**. The step value applies to the minute field.
5. **b. Remove secrets and sensitive information and share only the needed excerpt**. Logs can expose confidential system and user data.
6. **b. They can break required services, networking, or administrative access**. Each control needs system-specific review, verification, and rollback planning.
7. **c. To create an initial file-integrity database**. Later checks compare the system with that baseline.
8. **a. To avoid hardcoding or logging a shared plaintext password**. `passwd` collects the temporary password without placing it in the script.

5

Generating Linux Commands with Shell Genie

In the previous chapters, we prepared the CentOS environment and used a ChatGPT CLI workflow to ask questions, explain errors, and create scripts. Now we will add another assistant to our toolkit. Shell Genie accepts an instruction written in simple English, turns it into a Linux command, shows us that command, and asks whether we want it to run the command for us.

That small difference makes Shell Genie useful when we know what we want the system to do but cannot remember the exact syntax. A beginner can learn by reading the generated command, while an experienced administrator can save time on repetitive typing. In this chapter, we will install and configure Shell Genie, then use it for compression, file operations, password management, package installation, script creation, and basic system checks.

In this chapter, we will cover the following topics:

- Understanding Shell Genie
- Installing and configuring Shell Genie on CentOS
- Generating commands for compression, file operations, and password management
- Creating Bash scripts and interacting with the Linux system

Let's get started!

Technical requirements

To follow the demonstrations in this chapter, make sure the following are available:

- The CentOS Stream virtual machine prepared in *Chapter 2*
- The user account used in the earlier chapters, with sudo privileges
- Python 3.10 or later and a working internet connection
- The OpenAI API key created in *Chapter 3*, if the OpenAI backend is selected

Understanding Shell Genie

Shell Genie is a command-line tool that listens to simple English instructions and generates the corresponding Linux command. Instead of trying to remember a long or unfamiliar command, we can tell Shell Genie what we want to accomplish and let it translate that request into terminal syntax. It behaves like a friendly assistant standing beside us at the command line.

This is especially helpful for new users and for people who do not use advanced Linux commands every day but still need to manage a system smoothly. Shell Genie was created as an open-source project by Dylan Castillo. Because its source is available under an open-source license, learners and administrators can inspect the project, install it without a license fee, and adapt the code to their needs.

NOTE

A quick note before we continue: Shell Genie is retained here because its English-to-command workflow remains useful to study. However, the project repository was archived on March 4, 2024, and the maintainer states that it is no longer maintained. The installation, backends, and API connection shown in this chapter therefore reflect the recorded environment and may require adaptation on a newer system.

Think about a simple situation in the kitchen. If we need a cup of tea, we can tell someone, "Make me a cup of tea." We do not have to list the ingredients, explain every step, or specify how long the water should boil. The other person already knows how to turn that simple instruction into action. Shell Genie follows the same idea. We can say, "Compress this directory," and it works out the Linux command that performs the task.

The main purpose is to reduce complexity and make the terminal feel less confusing. It can save time, remove some of the fear around unfamiliar syntax,

and help beginners learn gradually because the final command appears on the screen before it is run. We can read the command, connect the English request to the Linux syntax, and remember more of it the next time. For an advanced user, the same process can speed up repetitive work and reduce tedious typing.

How Shell Genie complements ChatGPT CLI

Shell Genie and ChatGPT CLI overlap, but they are not doing exactly the same job in this workflow. ChatGPT CLI acts more like an intelligent mentor. We can use it to explain an error, create or improve a script, examine code, and work through a problem step by step. Shell Genie is more focused: it converts a plain-English request into a runnable Linux command as quickly as possible.

In this book, ChatGPT CLI is our shorthand for the terminal-based Python and OpenAI SDK workflow introduced in *Chapter 3*; it is not a separate OpenAI product.

Used together, the two tools form a practical combination. Shell Genie can generate a command immediately, while ChatGPT CLI can explain that command, troubleshoot it, customize it, or expand it into a complete script or scheduled task. One tool helps us move quickly from intention to command; the other helps us understand and develop the solution further.

Installing and configuring Shell Genie on CentOS

We will now install Shell Genie on the CentOS machine. The demonstrated environment uses Python 3.12.11, which is already present, so there is no need to install another Python version before continuing.

Checking Python and pip

First, confirm the installed Python version:

```
python3 --version
```

The output returns:

```
Python 3.12.11
```

Next, check whether pip is available:

```
pip3 --version
```

On our machine, this command is not found. CentOS offers to install the `python3-pip` package, but we enter `n` because this walkthrough uses Python's built-in `ensurepip` module instead.

```
Install package 'python3-pip' to provide command 'pip3'? [N/y] n
```

Run the following command:

```
python3 -m ensurepip --upgrade
```

The `-m` option tells Python to run a module. In this case, `ensurepip` creates or repairs the pip installation directly from the Python environment, and `--upgrade` asks it to install the bundled version if the existing pip is older. Because the system site-packages directory is not writable by this user, pip is installed in the user environment. When the command finishes, the message confirms that pip has been installed successfully.

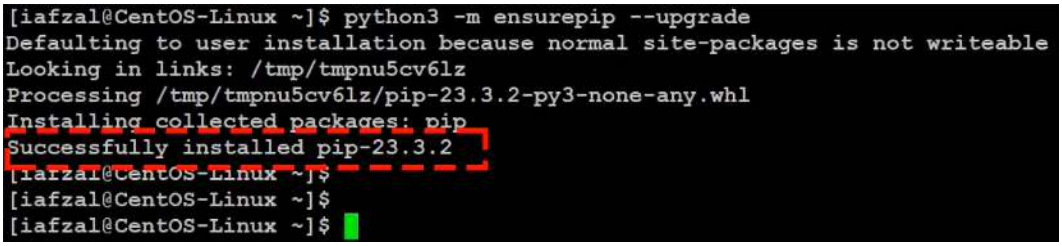


Figure 5.1: Checking Python and installing pip with ensurepip

Installing pipx and Shell Genie

Next, we install `pipx`. The reason is simple: `pipx` installs Python command-line applications in isolated virtual environments. Shell Genie can therefore use its own packages without mixing them into the system Python environment or another Python application.

```
python3 -m pip install --user pipx
```

The `--user` option places `pipx` in the current user's environment rather than writing to system directories. The installation provides `pipx` 1.8.0 along with its required packages.

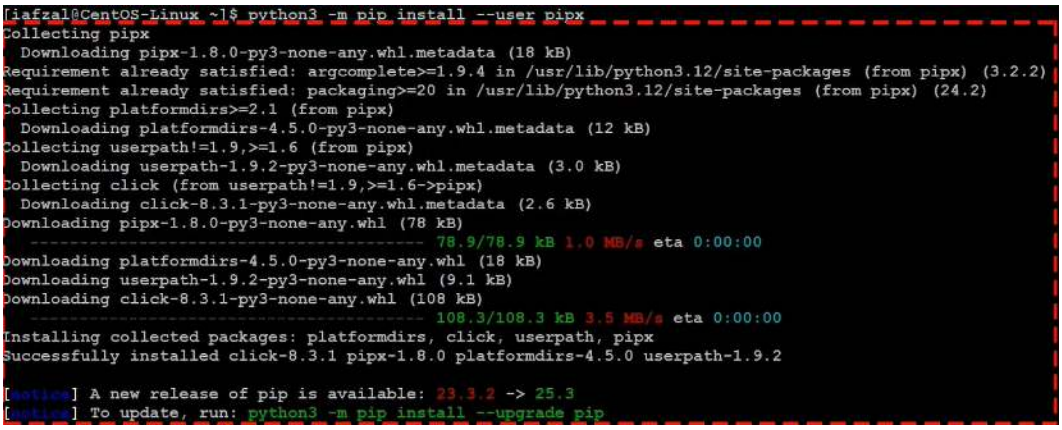


Figure 5.2: Installing pipx in the user environment

Make sure the `pipx` executable directory is on the user's PATH:

```
python3 -m pipx ensurepath
```

The output says that `/home/iafzal/.local/bin is already in PATH` and that pipx is ready to go.

Figure 5.3: Confirming that the pipx executable is available on PATH

PATH is the list of directories the shell searches when we enter a command, so this confirmation means we can run pipx without typing its full location.

Now install Shell Genie:

```
pipx install shell-genie
```

`pipx` downloads Shell Genie and prepares it inside an isolated environment. This may take a short while.

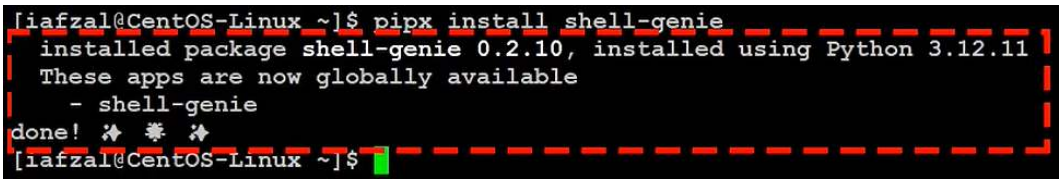


Figure 5.4: Installing Shell Genie with pipx

When the process finishes, the installation message confirms that the `shell-genie` application is available.

Resolving the Click compatibility error

Initialize the application:

```
shell-genie init
```

Oops, the first attempt produces a long traceback that ends with the following message:

```
TypeError: Secondary flag is not valid for non-boolean flag.
```

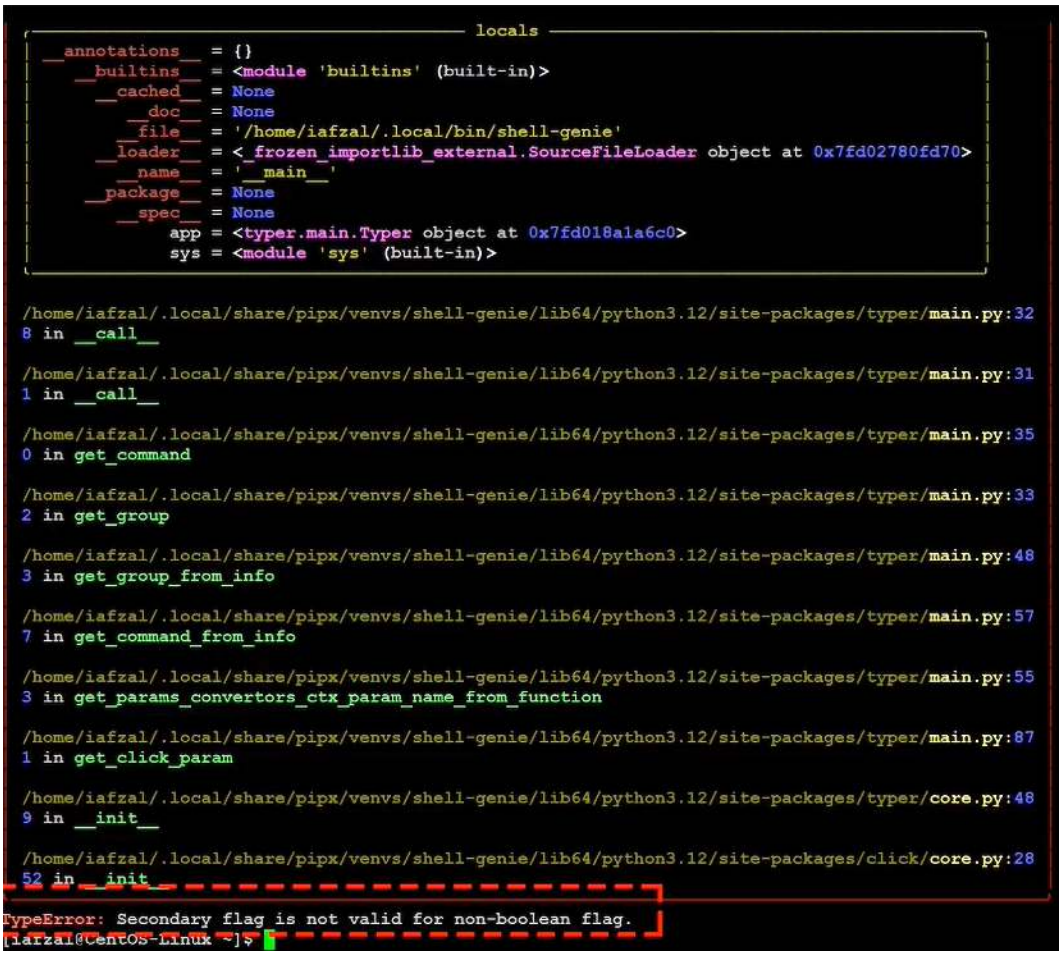


Figure 5.5: The Click compatibility error during Shell Genie initialization

Do not worry. In our environment, this happens because Shell Genie receives a newer version of Click than it can use. Click is the Python library that handles command-line options. Because pipx gives Shell Genie its own isolated

environment, we can target that environment and install an earlier Click release without changing Click for the entire system.

```
/home/iafzal/.local/share/pipx/venvs/shell-genie/bin/python3 -m pip install 'click<8.3.0'
```

The command removes Click 8.3.1 and installs Click 8.2.1 inside the Shell Genie virtual environment.

> **NOTE**
>
> The path contains the demonstration username. If you are using a different account, replace `/home/iafzal` with that user's home directory. The version numbers describe the recorded environment, so apply this downgrade only if `shell-genie init` ends with the same compatibility error.

Figure 5.6: Installing Click 8.2.1 in the Shell Genie environment

Run the initialization command again:

```
shell-genie init
```

This time, the backend selection menu appears, which confirms that the compatibility change worked.

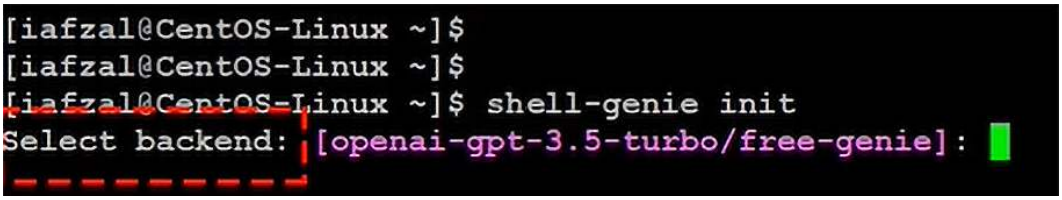


Figure 5.7: Resolving the Click compatibility error

Selecting the backend and saving the configuration

The menu shows two backend choices, `openai-gpt-3.5-turbo` and `free-genie`. The first option sends requests to an OpenAI model and requires an API key.

The second is the free backend offered by the project. Because we used OpenAI in the previous chapters, we select the same route here.

> **NOTE**
>
> The free-genie service was offered without an availability guarantee, and OpenAI has announced the retirement of the gpt-3.5-turbo model family on October 23, 2026. Because Shell Genie is no longer maintained, its OpenAI backend may stop working when that model is retired. The following steps reproduce the demonstration at the time of writing rather than presenting this backend as a current long-term integration.

```
openai-gpt-3.5-turbo
```

Shell Genie now asks for the OpenAI API key created earlier in *Chapter 3*. Paste your own key in the terminal. Never copy a key from a screenshot or include your key in a screenshot that will be shared:

```
<your-openai-api-key>
```

Next, Shell Genie asks whether the detected operating system is CentOS Stream 10 (Coughlan). Enter `y` to confirm:

```
Is your OS CentOS Stream 10 (Coughlan)? [y/n]: y
```

Shell Genie displays the selected backend, operating-system details, Bash shell, and API-key field, then saves the configuration in the following file:

```
/home/iafzal/.config/.shell_genie/config.json
```

The configuration file contains the API key in plain text. Restrict it so that only the current user can read or modify it:

```
chmod 600 /home/iafzal/.config/.shell_genie/config.json
```

> **NOTE**
>
> **Protect the key:** Do not commit this file to version control or share an unmasked terminal capture. If a key is exposed, revoke it and create a replacement. The initialization output below is recreated with the key redacted.

```
[iafzal@CentOS-Linux ~]$ shell-genie init
Select backend: [openai-gpt-3.5-turbo/free-genie]: openai-gpt-3.5-turbo
Enter an OpenAI API key: <redacted>
Is your OS CentOS Stream 10 (Coughlan)? [y/n]: y
The following configuration will be saved:
  'backend': 'openai-gpt-3.5-turbo'
  'os': 'linux'
  'os_fullname': 'CentOS Stream 10 (Coughlan)'
  'shell': 'bash'
  'openai_api_key': '<redacted>'
Config file saved at /home/iafzal/.config/.shell_genie/config.json
```

Figure 5.8: Initializing Shell Genie with the API key redacted

At this point, the setup is complete and Shell Genie is ready to receive an English request.

Testing Shell Genie by installing Apache

Let's test the installation with a simple task. Ask Shell Genie to install the Apache server:

```
shell-genie ask "install apache server"
```

Shell Genie converts the request into the following command:

```
sudo dnf install httpd
```

This is the CentOS command for installing the httpd package. Shell Genie shows the command and asks whether we want to run it. The demonstration enters `y` so that Shell Genie executes the command automatically:

```
Do you want to run the command? [y/n]: y
```

NOTE

Read before you run: Shell Genie can generate an incorrect or damaging command. Before entering `y`, make sure you understand the command, confirm every path and package name, and check that it does only what you requested. In this example, `httpd` is the expected Apache package on CentOS.

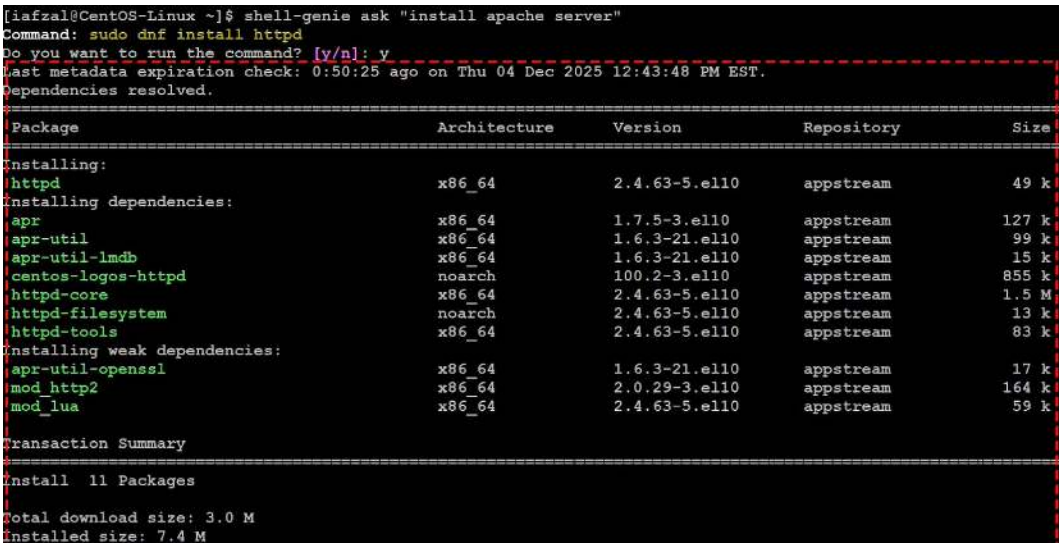


Figure 5.9: Reviewing the Apache packages and dependencies before installation

Because the generated command uses `sudo`, the system may request the user account password. DNF then lists httpd and its dependencies and asks for another confirmation. Enter `y` again at the transaction prompt:

```
Is this ok [y/N]: y
```

DNF downloads and installs httpd together with the required packages. When `Complete!` appears, Apache has been installed.

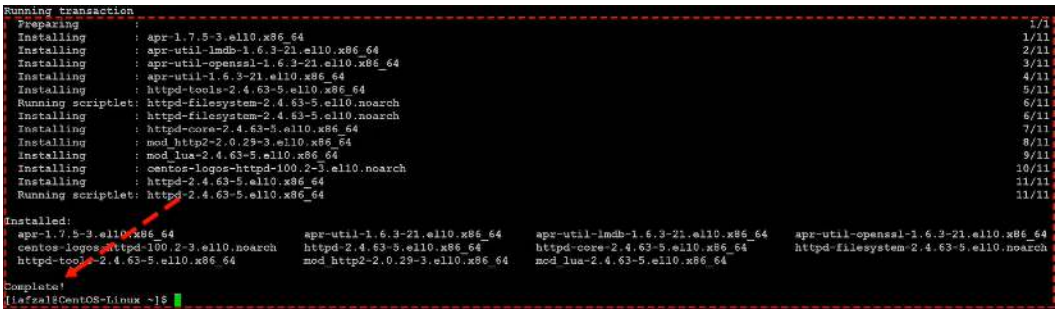


Figure 5.10: Installing Apache from a Shell Genie request

All of that started with one short English sentence. This is a good example of how Shell Genie connects an everyday request to the command CentOS expects.

Generating commands from English

Now that Shell Genie is installed and working, we can use it for several everyday Linux tasks. We will start with compression, move an archive before

deleting the original, and then change the current user's password. Each example follows the same rhythm: describe the result in plain English, read the generated command, allow Shell Genie to run it, and verify what changed.

Compressing the Documents directory

First, list the sample files in the `Documents` directory.

If these files are not already present from your lab setup, create the same ten empty sample files before continuing:

```
mkdir -p /home/iafzal/Documents && touch /home/iafzal/Documents/file{1,3,5,7,9}.sh
/home/iafzal/Documents/file{2,4,6,8,10}.txt
```

The brace expressions expand to the five odd-numbered `.sh` filenames and the five even-numbered `.txt` filenames shown in the demonstration.

```
ls -ltr /home/iafzal/Documents/
```

The `Documents` directory contains ten empty sample files: `file1.sh`, `file2.txt`, `file3.sh`, `file4.txt`, `file5.sh`, `file6.txt`, `file7.sh`, `file8.txt`, `file9.sh`, and `file10.txt`. The mixture of text files and Bash script files gives us a small directory to archive.

Ask Shell Genie to compress the directory:

```
shell-genie ask "Compress the Documents directory in /home/iafzal/"
```

Shell Genie generates:

```
tar -czvf /home/iafzal/Documents.tar.gz /home/iafzal/Documents
```

Here, `tar` creates the archive, `-c` starts a new archive, `-z` applies gzip compression, `-v` lists each file as it is added, and `-f` identifies the archive filename. Enter `y` when Shell Genie asks whether to run the command.

```
Do you want to run the command? [y/n]: y
```

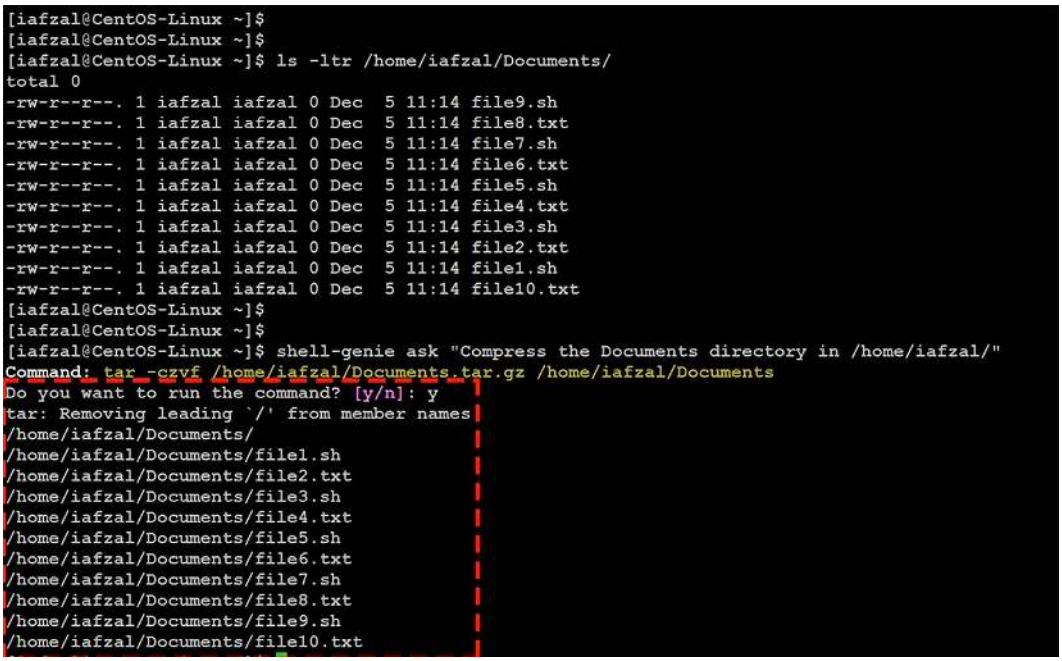


Figure 5.11: Shell Genie generating and running the tar command

As `tar` works, it lists the ten files. It also reports that it is removing the leading `/` from member names. This is normal: storing relative paths in the archive avoids recreating files from an absolute root path during extraction.

Verify that the archive exists in the home directory:

```
ls -ltr /home/iafzal/
```

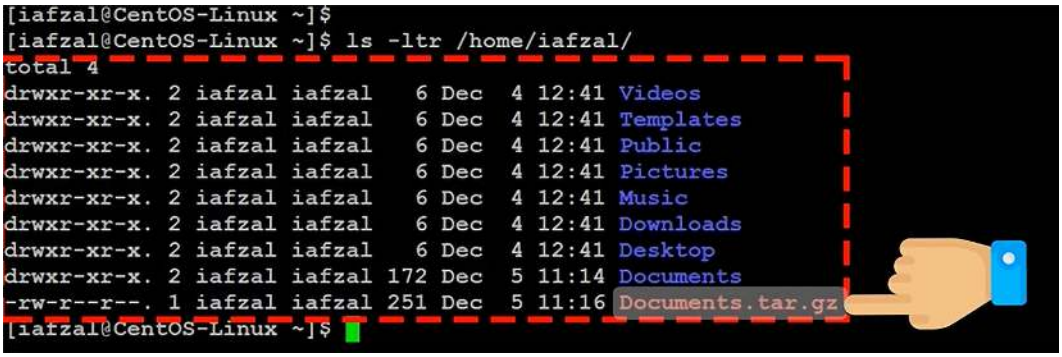


Figure 5.12: Verifying the Documents.tar.gz archive in the home directory

A new file named `Documents.tar.gz` appears beside the `Documents` directory, confirming that the directory was compressed and the archive was saved where we requested.

Copying an archive before deleting the original

Let's change the scenario. First, confirm that **Downloads** is empty:

```
ls -ltr /home/iafzal/Downloads/
```

Now give Shell Genie a slightly larger instruction. We want it to copy the archive into `Downloads` and only then remove the original from the home directory:

```
shell-genie ask "Delete the Documents.tar.gz file in /home/iafzal/, but before you do that, first
copy the file into /home/iafzal/Downloads/"
```

Shell Genie translates the request into two commands joined with `&&`:

```
cp /home/iafzal/Documents.tar.gz /home/iafzal/Downloads/ && rm /home/iafzal/Documents.tar.gz
```

The `cp` command copies the archive. The `&&` operator tells Bash to run rm only if `cp` succeeds. That ordering matters because the original should not be removed when the copy fails. The `rm` command permanently removes only the original `Documents.tar.gz` archive from `/home/iafzal`; it does not delete the `Documents` directory or the copy in `Downloads`. Confirm both paths before entering `y` to execute the generated command.

```
Do you want to run the command? [y/n]: y
```

Check the home directory first:

```
ls -ltr /home/iafzal/
```

`Documents.tar.gz` is no longer there, so the remove part is completed. Next, inspect `Downloads`:

```
ls -ltr /home/iafzal/Downloads/
```

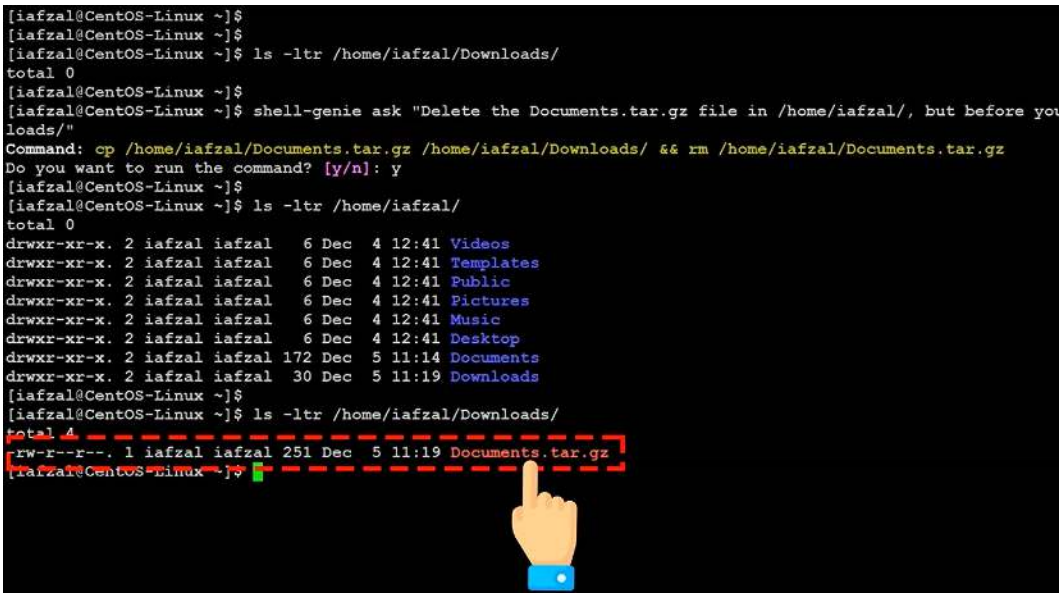


Figure 5.13: Copying the archive before deleting the original

`Documents.tar.gz` appears in `Downloads`. This verifies both parts of the instruction: Shell Genie copied the archive to the new location and then deleted the original.

Changing the user password

For the next example, we will perform a user-account task. Ask Shell Genie to change the password of the `iafzal` user:

```
shell-genie ask "Change the iafzal user password."
```

The generated command is:

```
passwd iafzal
```

Enter `y` to allow Shell Genie to run it. The `passwd` command first asks for the current password, then asks for the new password, and finally asks us to retype the new password. Nothing appears on the screen while a password is being entered; this is normal terminal behavior.

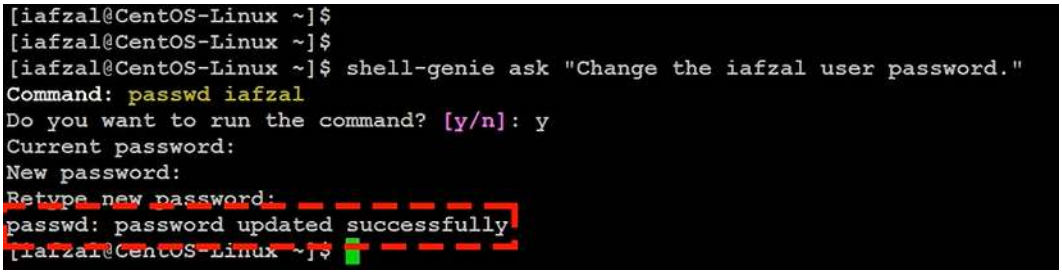


Figure 5.14: Changing the current user's password

The final success message confirms that the password was changed. In just these three examples, the same plain-English interface has handled an archive, a conditional file operation, and an account-management command.

Creating scripts and interacting with the system

Shell Genie can also combine several shell operations into one generated command. In the next two examples, we will create a Bash file, write commands into it, make it executable, and run it. A task that normally takes several separate commands begins with one English instruction.

Creating and running an EPEL installation script

In the first request, we ask Shell Genie to create a script in `Downloads` that installs the `epel-release package`, give the script execute permission, and run it:

```
shell-genie ask "Create a Bash file inside /home/iafzal/Downloads that contains code to install the
epel-release package, give the file executable permission and run it"
```

The generated command performs all three parts:

```
echo -e "#!/bin/bash\nsudo dnf install epel-release -y" > /home/iafzal/Downloads/install_epel.sh &&
chmod +x /home/iafzal/Downloads/install_epel.sh && /home/iafzal/Downloads/install_epel.sh
```

The `echo` command writes the `shebang` and `dnf` command into `install_epel.sh`. The `>` operator creates the file or replaces its existing contents. `chmod +x` makes the script executable, and the final path runs it. The `&&` operators keep the sequence in order so each stage begins only after the preceding command succeeds.

Enter `y` when Shell Genie asks whether to execute the command:

```
Do you want to run the command? [y/n]: y
```

The script uses `sudo`, so enter the `iafzal` account password if prompted.

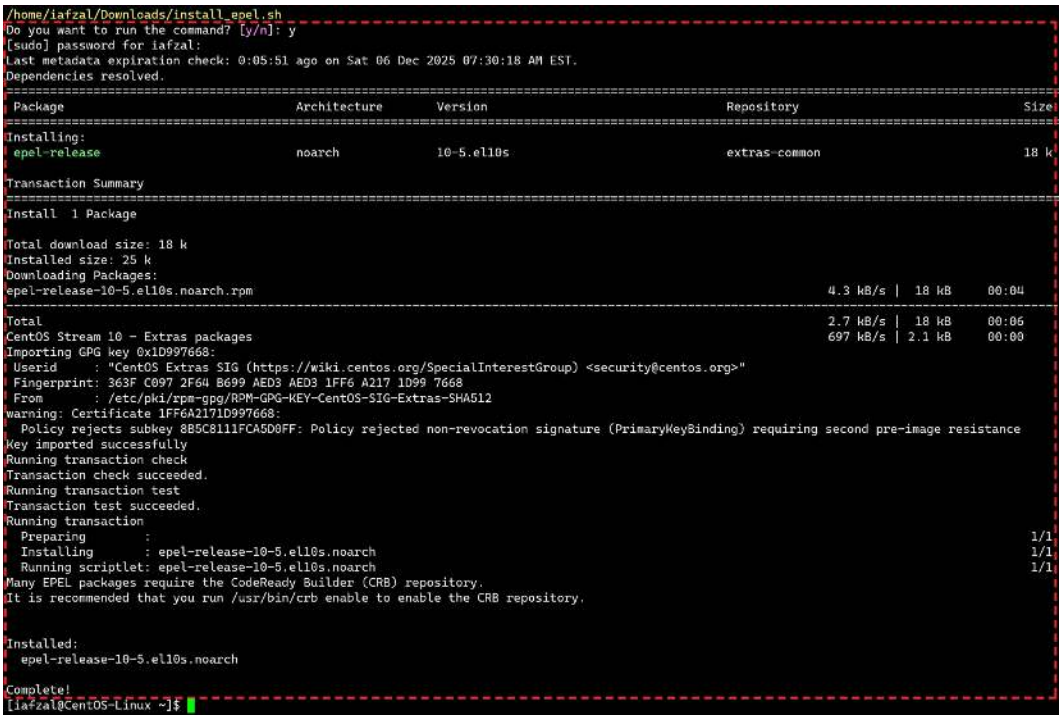


Figure 5.15: Installing the epel-release package

DNF downloads `epel-release` and imports the repository signing key. In the demonstrated CentOS Stream 10 environment, the output also includes an EPEL policy or revocation-signature warning and suggests enabling CRB with `/usr/bin/crb enable`. After the transaction finishes, DNF prints `Complete!`

The CRB line is an advisory message rather than a failure of this transaction. CRB supplies dependencies required by some EPEL packages. If a later EPEL package reports a missing dependency, enable CRB as suggested and retry that package installation; no additional CRB action is needed for the completed epel-release installation shown here.

Verify that the Bash file was created:

```
ls -ltr /home/iafzal/Downloads/
```

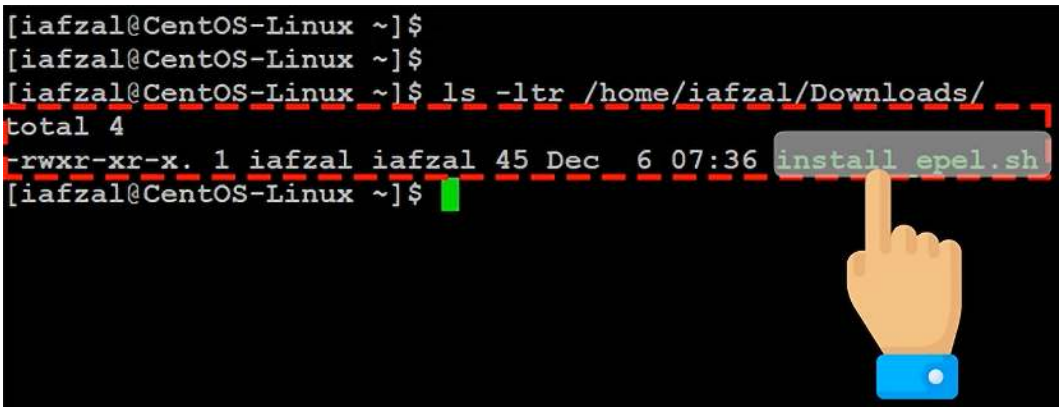


Figure 5.16: Creating and running the EPEL installation script

The output shows `install_epel.sh` with execute permission. The demonstrated file is 45 bytes, which confirms that Shell Genie created it in the requested directory and populated it with the package-installation command.

Creating and running a system-check script

For the final example, ask Shell Genie to create a script in `Documents` that reports memory, disk usage, and network status, then make the script executable and run it:

```
shell-genie ask "Create a Bash file in /home/iafzal/Documents that checks memory, checks disk usage,
and shows network status, then make it executable and run it."
```

Shell Genie generates the following combined command:

```
echo -e '#!/bin/bash\nfree -m\ndf -h\nip a' > /home/iafzal/Documents/system_check.sh && chmod +x
/home/iafzal/Documents/system_check.sh && /home/iafzal/Documents/system_check.sh
```

The generated script contains three familiar system commands. `free -m` displays memory figures in mebibytes, `df -h` displays filesystem usage in human-readable units, and `ip a` displays the network interfaces and their addresses. As before, `chmod +x` makes the new file executable and the final path runs it.

Enter y to run the complete sequence:

```
Do you want to run the command? [y/n]: y
```

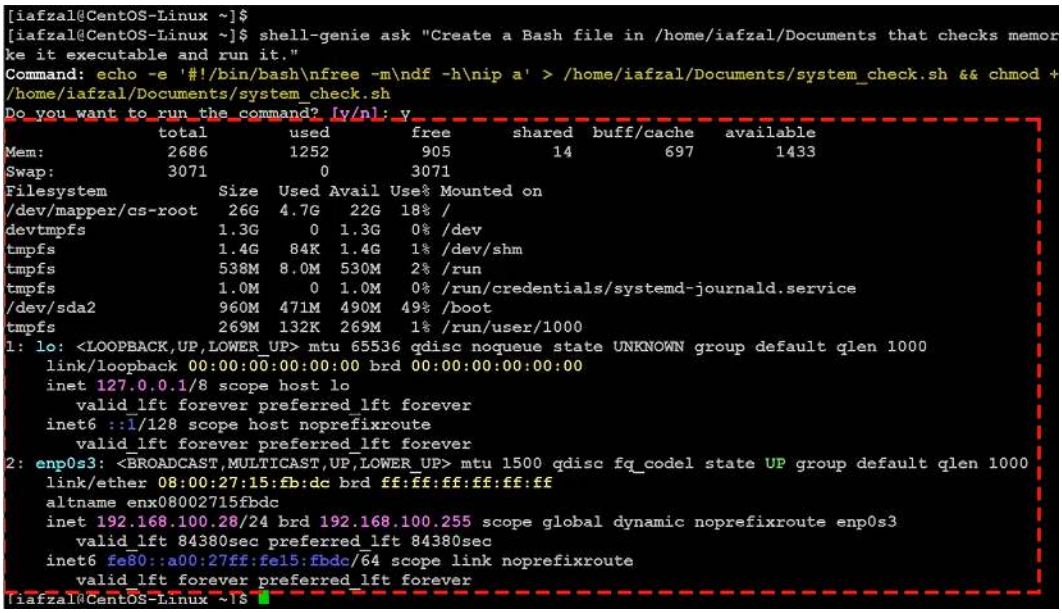


Figure 5.17: Creating and running the system-check script

The output appears in the same order as the script. First we see the memory totals, used memory, and available memory. Next we see the mounted filesystems and their disk usage. Finally, `ip a` displays the loopback interface and the main `enp0s3` network interface. In our demonstration, `enp0s3` has the IPv4 address `192.168.100.28/24`; the address on another machine will probably be different.

This example brings the chapter's main idea together. We did not write each Bash line manually. We described the intended result in ordinary English, Shell Genie assembled the file-creation and execution command, and the resulting script interacted with the system to collect useful information.

Summary

In this chapter, we introduced Shell Genie as a command-line assistant that converts plain-English instructions into Linux commands. We used the kitchen tea example to see why this approach feels natural: we describe the outcome, and the assistant translates that intention into the steps the system requires. We also compared Shell Genie's command-focused role with the broader explanatory and scripting role of ChatGPT CLI.

We then installed `pip` with `ensurepip`, installed `pipx` and Shell Genie in the user environment, corrected the demonstrated Click compatibility error, selected

the OpenAI backend, and saved the Shell Genie configuration. A simple request generated the `dnf` command that installed Apache.

Finally, we used English requests to compress a directory, copy an archive before deleting the original, change a user password, create and run an EPEL installation script, and create a script that reports memory, disk usage, and network status. Each task showed the generated Linux syntax and then verified the result on the CentOS machine.

Knowledge check

Test your understanding of the concepts and commands covered in this chapter.

1. What is Shell Genie primarily designed to do?
   A. Replace the CentOS operating system
   B. Convert plain-English instructions into Linux commands
   C. Monitor hardware without running commands
   D. Store API keys in the Linux kernel
2. Why is pipx used to install Shell Genie?
   A. It installs command-line applications in isolated environments
   B. It changes the Linux hostname
   C. It replaces the Bash shell
   D. It creates a virtual machine
3. What did the Click compatibility command change?
   A. It removed Python from CentOS
   B. It installed an earlier Click release only in Shell Genie's pipx environment
   C. It changed the user password
   D. It enabled Apache at boot
4. What does && do in the generated copy-and-delete command?
   A. It runs rm only if cp succeeds
   B. It runs both commands at exactly the same time
   C. It compresses both commands
   D. It sends both commands to a remote machine

5. Which command was generated to change the iafzal user password?
   A. usermod iafzal
   B. passwd iafzal
   C. password --reset iafzal
   D. sudo useradd iafzal
6. Which three commands were placed in system_check.sh?
   A. ls, pwd, and whoami
   B. tar, cp, and rm
   C. free -m, df -h, and ip a
   D. dnf, rpm, and systemctl

Answers

1. **b. Convert plain-English instructions into Linux commands**. Shell Genie translates the requested outcome into terminal syntax and displays the generated command.
2. **a. It installs command-line applications in isolated environments**. This keeps Shell Genie's Python dependencies separate from system packages and other applications.
3. **b. It installed an earlier Click release only in Shell Genie's pipx environment**. The demonstrated command replaced Click 8.3.1 with Click 8.2.1 for this application.
4. **a. It runs rm only if cp succeeds**. This preserves the requested order and prevents the original archive from being removed when the copy command fails.
5. **b. passwd iafzal**. The command prompts for the current password and then requests the new password twice.
6. **c. free -m, df -h, and ip a**. These commands report memory, filesystem usage, and network-interface information, respectively.

6

Using Aider for Bash Scripting and Linux Automation

In the previous chapter, we used Shell Genie to turn simple English instructions into Linux commands. Now we will take that idea a step further with Aider. Aider is an AI assistant that works inside the terminal and helps us create, read, and improve code. We can describe what we need in ordinary English, and Aider can turn that request into a Bash script or Linux command.

That is especially useful when we are still learning. We may understand the task perfectly but not remember every bracket, option, loop, or systemctl command. Instead of getting stuck on syntax, we can explain the outcome we want, study what Aider produces, and decide whether to apply or run it. Experienced administrators can use the same workflow to save time when creating repetitive snippets and script templates.

In this chapter, we will first understand where Aider fits into a Linux workflow. We will then install it with pipx, connect it to an OpenAI model, and test the connection by asking for the firewall status. After that, we will create and run an Apache installation script, followed by a second script that loops through log files and adds a cron job.

In this chapter, we will cover the following topics:

- Understanding Aider and when to use it
- Installing and configuring Aider on CentOS
- Using Aider to create and run a Bash snippet
- Generating a loop, a cron job, and a reusable script template

Let's get started!

Technical requirements

To follow the demonstrations in this chapter, make sure the following are available:

- The CentOS Stream virtual machine prepared in *Chapter 2*
- The user account used in the earlier chapters, with sudo privileges
- Python 3.12 and pipx, as configured in the preceding chapters
- A working internet connection
- An active OpenAI API key and API billing or credits

Understanding Aider and when to use it

Let's begin with a simple question: what exactly is Aider? Think of Aider as a helpful AI partner that sits with us inside the terminal. It is designed to help us write code, edit existing files, understand what a program is doing, and fix problems. We do not need to start by giving it perfect code. We can start with a sentence in simple English.

For example, suppose we enter the following request at the Aider prompt:

```
create a bash file that prints hello
```

Aider can turn that request into a small Bash script such as:

```
#!/bin/bash
echo 'Hello, World!'
```

There are two useful ideas in this tiny example. First, Aider understands the intention even though the request does not contain Bash syntax. Second, it gives us working code that we can read. The first line tells Linux which interpreter should run the file, and the second line prints the message. This is why the tool can help beginners learn instead of simply hiding the command line from them.

Aider is not limited to complete programs. We can also ask it for a Linux command. Suppose we want to create a directory inside `Documents` and then list the files. We could describe the task like this:

```
make a directory inside Documents and list all files
```

Aider can turn that request into a command such as:

```
mkdir -p Documents/new_folder ; ls -l
```

Here, `mkdir` creates the `new_folder` directory. The `-p` option also creates any missing parent directories and avoids an error if the directory already exists. After the semicolon, `ls -l` displays a detailed listing of the current working directory, where the `Documents` directory appears. To inspect the new directory itself, use `ls -ld Documents/new_folder`. One sentence has therefore become two connected shell operations.

This is the basic pattern we will use throughout the chapter: tell Aider what we want, read the proposed code or command, allow the requested file change, and then choose whether to run the shell commands. Aider is helping, but we remain in control of the decisions shown at the terminal.

Using Aider with Git

One of Aider's strongest features is its integration with Git. When we work inside a Git repository, Aider can keep track of the files it changes and commit those changes with descriptive messages. We can review what was modified, move through the history, and undo an AI change if it does not produce the result we expected.

This is useful when a project contains several files or when we are improving existing code. We can ask Aider to improve a function, fix an error, or update part of a script, and Git gives us a record of the work. In other words, Aider can help with the code while Git helps us keep that work organized and recoverable.

Working without Git

Aider can also work when Git is not connected. It can still read files, understand instructions, generate code, and propose shell commands. What we lose is the automatic Git history and the easy ability to review or undo changes through commits.

That is the mode used in the practical demonstrations in this chapter. We will start Aider with `--no-git` so that the focus remains on the Bash requests and system interaction. The startup screen will therefore report `Git repo: none`

and `Repo-map: disabled`. This does not mean Aider has failed. It simply confirms that Git integration has been intentionally disabled for this session.

> **NOTE**
>
> When `--no-git` is used, Aider does not create Git commits for its edits. Keep a separate copy of any important file before experimenting.

Installing and configuring Aider

We are now ready to install Aider. Log in to the CentOS machine and open the terminal. We will use `pipx` because it installs a Python command-line application in its own isolated environment. This keeps Aider and its dependencies separate from the system Python packages.

Installing Aider with pipx

Run the following command:

```
pipx install aider-chat
```

The download and installation can take a little time. In our environment, `pipx` installs the `aider-chat 0.86.1` package using Python 3.12.11 and reports that the aider application is now globally available. The word globally here means that the aider command is available from the user's shell; the Python packages themselves still remain inside pipx's isolated environment.

Verify the installed version:

```
aider --version
```

The demonstrated system returns:

```
aider 0.86.1
```

Figure 6.1: Installing Aider and checking its version

If a later version appears on your machine, that simply means `pipx` downloaded a newer release. The screens or default model may be different, but the installation has still succeeded if the version command returns normally.

Providing the OpenAI API key

Aider needs access to a language model before it can answer our requests. We use an OpenAI model, so we provide the same OpenAI API key used in the earlier chapters. Export it as an environment variable in the current shell:

```
export OPENAI_API_KEY='<your-openai-api-key>'
```

Replace `<your-openai-api-key>` with the real key. The `export` command does not print a success message. It makes the value available to programs launched from this terminal session, including Aider. If the terminal is closed, the variable will normally need to be exported again in a new session.

Starting Aider without Git

Start Aider with Git integration disabled:

```
aider --no-git
```

The startup information confirms the connection and the selected mode. In our environment, Aider displays the following details:

Figure 6.2: Starting Aider without Git

The main model handles the coding request. The weak model can be used for smaller supporting tasks. `Git repo: none` and `Repo-map: disabled` are

expected because we started the program with `--no-git`. When the green > prompt appears, Aider is ready.

Testing the connection

Before asking Aider to create a complete script, let's test it with a straightforward system-administration question. At the Aider prompt, enter:

```
check firewall status
```

Aider proposes the following command:

```
sudo systemctl status firewalld
```

It also explains that `firewalld` is commonly used to manage firewall rules on Linux and that a different command might be needed if the system uses another firewall tool. Notice that Aider has not run anything yet. It shows the shell command and asks for confirmation:

```
Run shell command? (Y)es/(N)o/(D)on't ask again [Yes]:
```

Enter `y`. Because the command uses `sudo`, enter the user account password when prompted.

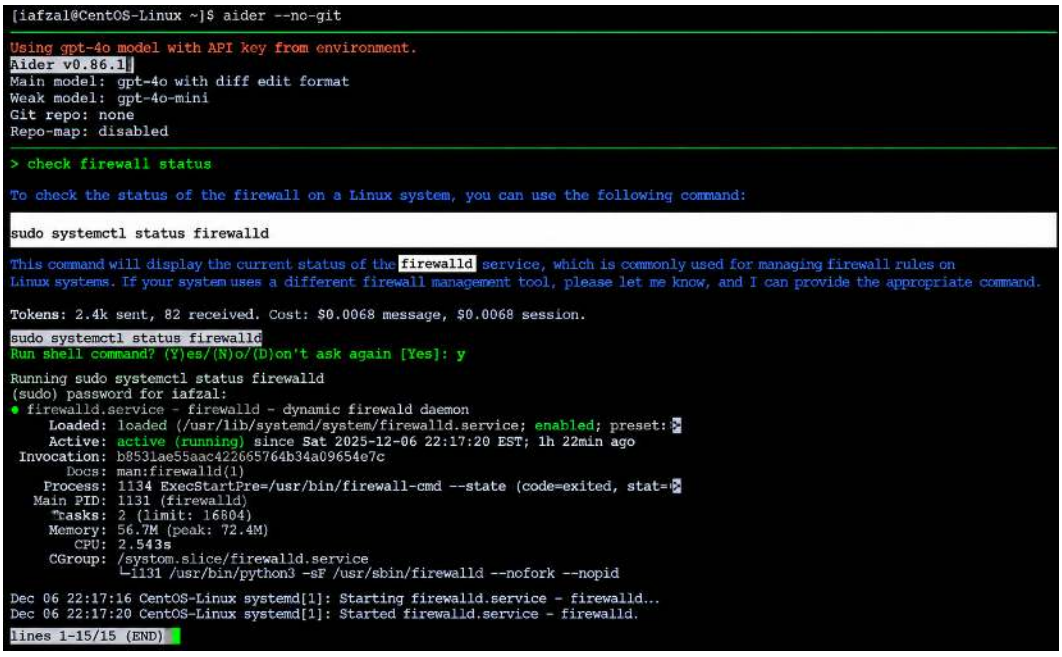


Figure 6.3: Checking the firewall status through Aider

The `systemctl` output identifies `firewalld.service` as the dynamic firewall daemon. In the demonstrated environment, `Loaded` shows `enabled` and `Active` shows `active (running)`. This confirms both that the service is currently running and that it is configured to start automatically.

The status display opens in a pager and ends with `(END)`. Press `q` to return to the Aider prompt.

Our first test is complete: Aider understood a short English request, selected a CentOS command, asked before executing it, and showed the live service status.

Creating and running a Bash snippet

Now let's move from a single command to a complete Bash file. We want a script that installs the Apache HTTP Server, enables it at boot, starts it now, and displays its status. Normally, that requires several commands and a little care with their order. We will ask Aider to assemble the snippet and save it for us.

Start Aider again if it is not already running:

```
aider --no-git
```

At the Aider prompt, enter the full request:

```
Write a Bash snippet to install httpd, enable it, start it, and check its status, then save the file
in /home/iafzal/Documents
```

Aider sends the request to the configured model and replies that it will create `/home/iafzal/Documents/install_httpd.sh`. It then displays the proposed file contents.

The generated script is:

```
#!/bin/bash

# Install httpd
sudo yum install -y httpd

# Enable httpd to start on boot
sudo systemctl enable httpd

# Start httpd service
```

```
sudo systemctl start httpd

# Check the status of httpd service
sudo systemctl status httpd
```

Let's pause and read what Aider produced. The shebang selects Bash. `yum install -y` installs the httpd package and answers yes to the package transaction. `systemctl` enable creates the boot-time link, `systemctl start` launches the service immediately, and `systemctl status` shows whether Apache is running. Aider has kept the four tasks in the same order in which we requested them.

The white `SEARCH/REPLACE` block shown by Aider is its proposed edit format. Because this is a new file, there is no previous content to search for. The replacement side contains the complete script. Aider also proposes two shell commands for the next stage:

```
chmod +x /home/iafzal/Documents/install_httpd.sh
/home/iafzal/Documents/install_httpd.sh
```

Approving the file and shell commands

Aider first asks whether it should create the new file:

```
Create new file? (Y)es/(N)o [Yes]:
```

Enter `y`. Aider confirms that it applied the edit to `/home/iafzal/Documents/install_httpd.sh`. It then asks whether it should run the `chmod` command and the script:

```
Run shell commands? (Y)es/(N)o/(D)on't ask again [Yes]:
```

Enter `y` again. Aider makes the file executable and runs it. When `sudo` asks for the user password, enter it.

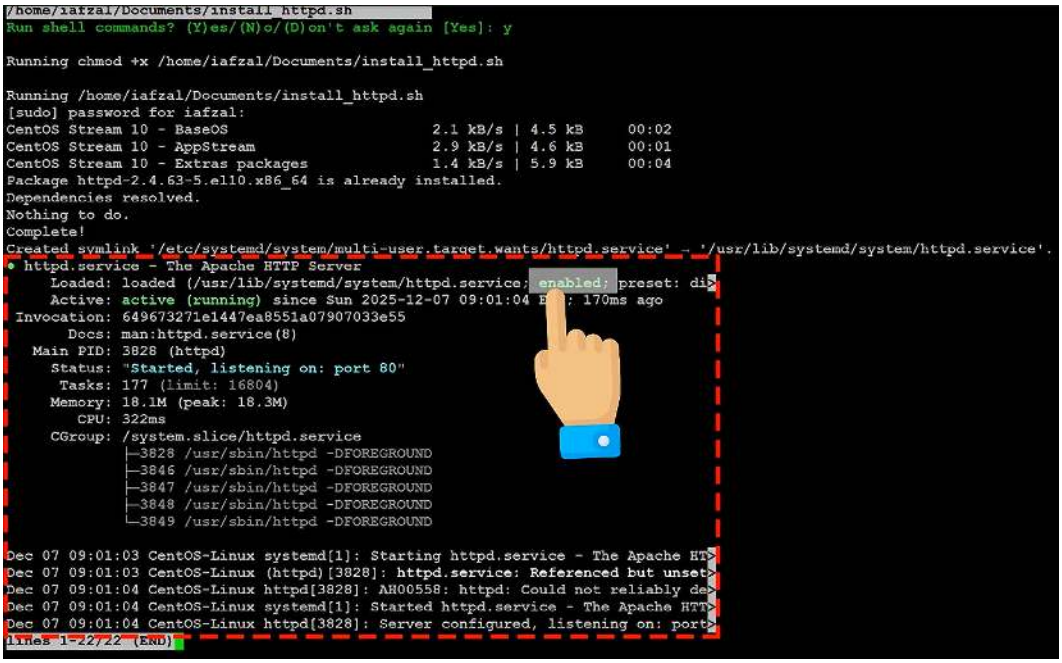


Figure 6.4: Creating and running the Apache installation script

On our machine, `httpd` is already installed, so `yum` reports that there is nothing to do. The `enable` command creates a symbolic link for `httpd.service`, and the status output shows both `enabled` and `active (running)`.

The httpd status also includes Started, listening on: port 80. This confirms that Apache has started and is listening for web traffic on its normal HTTP port. The `AH00558` message visible in the output is a warning that Apache could not determine a fully qualified server name; it does not mean the service failed to start. Press `q` if the output stops at `(END)` so that you can return to Aider.

Generating loops, cron jobs, and script templates

Our final example brings several Bash ideas together. Imagine that a CentOS administrator wants to go through the log files one by one and also wants a system-check command to run every ten minutes. Instead of creating the loop, cron syntax, and executable file separately, we will describe the complete requirement to Aider.

Start Aider in the same no-Git mode:

```
aider --no-git
```

At the prompt, enter:

```
Create Bash snippets for a CentOS admin who needs a loop to process log files, a cron job to run a system check every 10 minutes, and save the file in /home/iafzal/Documents/. Then give it executable permission before running.
```

Aider breaks the task into five parts: create the Bash file, add a loop, add the cron job, make the script executable, and run it. It chooses the filename `/home/iafzal/Documents/system_check.sh` and displays the following template:

```
#!/bin/bash

# Loop to process log files
for log_file in /var/log/*.log; do
    echo "Processing $log_file"
    # Add your log processing commands here
done

# Add a cron job to run a system check every 10 minutes
(crontab -l 2>/dev/null; echo "*/10 * * * * /usr/sbin/system_check") | crontab -
```

The `for` loop takes every filename matching `/var/log/*.log` and stores one name at a time in the `log_file` variable. The `echo` command prints `Processing` followed by that filename. The comment inside the loop marks the place where the administrator can add the real processing command later. As generated, this is a template: it identifies the log files but does not read, transform, archive, or delete them.

The final line reads the current crontab, hides the error that appears when no crontab exists, appends a new entry, and writes the combined result back with `crontab -`. The five asterisks represent the cron schedule fields. `*/10` in the minute field means every ten minutes. In the generated draft, the scheduled command is `/usr/sbin/system_check`.

NOTE

Before running this draft, notice two details. Aider creates `/home/iafzal/Documents/system_check.sh`, but the cron line points to `/usr/sbin/system_check`, which does not exist in this workflow. If the path were corrected but the `cron-update` command remained inside the script, each scheduled run could add the entry again. Keep the generated output above as a record of what Aider proposed, but correct the template before approving it.

Replace the generated file content with the following version, which keeps the demonstrated log-file loop and leaves cron registration as a separate, one-time step:

```
#!/bin/bash
# Loop to process log files
for log_file in /var/log/*.log; do
    echo "Processing $log_file"
    # Add your log processing commands here
done
```

After correcting the file, use the two shell commands proposed by Aider:

```
chmod +x /home/iafzal/Documents/system_check.sh
/home/iafzal/Documents/system_check.sh
```

Approve the new file when Aider asks:

```
Create new file? (Y)es/(N)o [Yes]: y
```

Then approve the chmod and execution commands:

```
Run shell commands? (Y)es/(N)o/(D)on't ask again [Yes]: y
```

Aider creates `system_check.sh`, applies execute permission, and runs it. The output lists the log files one by one, including `boot.log`, `dnf.librepo.log`, `dnf.log`, `dnf.rpm.log`, `hawkey.log`, and `usercreate.log`. That output comes from the echo command inside the loop.

Aider next asks whether the command output should be added to the chat. Enter `y`. It reports that seven lines of output were added and returns to the prompt. Adding output can be useful when we want Aider to examine the result, explain an error, or help with the next change.

```
Add command output to the chat? (Y)es/(N)o/(D)on't ask again [Yes]: y
```

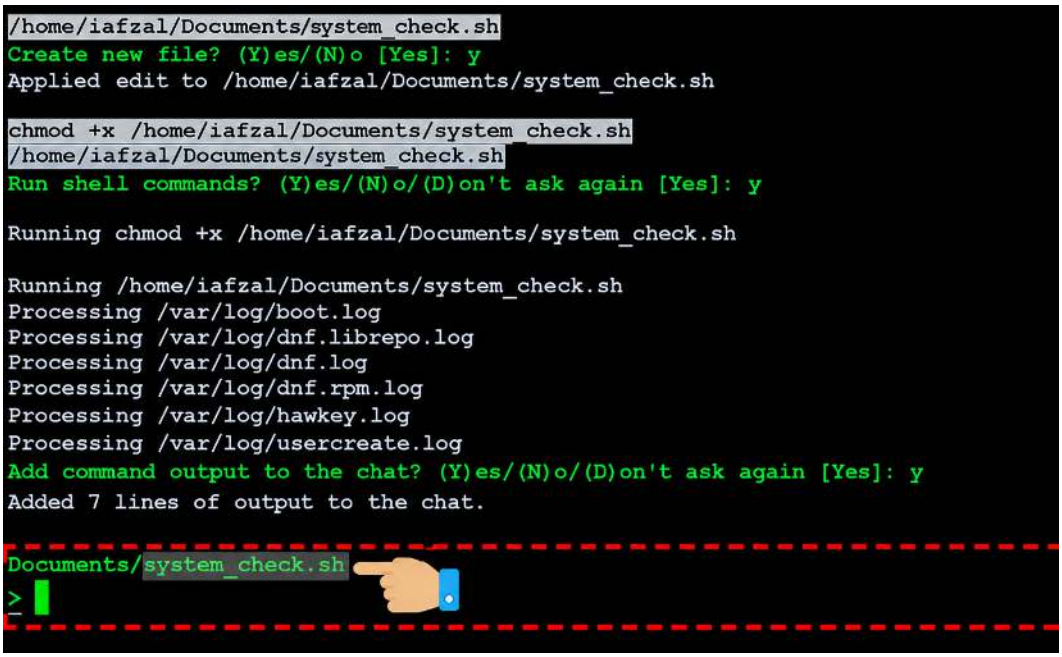


Figure 6.5: Creating and running the log-processing script

Exit Aider when the demonstration is complete:

```
/exit
```

Now add the cron entry once, using the path of the file Aider actually created:

```
(crontab -l 2>/dev/null | grep -vF "/home/iafzal/Documents/system_check.sh"; echo "*/10 * * * *
/home/iafzal/Documents/system_check.sh") | crontab -
```

The `grep` command removes any existing entry for this script before the corrected schedule is added. This prevents the command from creating duplicate entries when it is run again.

Verifying the cron entry and script

Back at the normal shell prompt, list the current user's crontab:

```
crontab -l
```

With the correction applied, the output should be:

```
*/10 * * * * /home/iafzal/Documents/system_check.sh
```

This confirms that the cron entry was added for the `iafzal` user and is scheduled every ten minutes. *Figure 6.6* retains the original

`/usr/sbin/system_check` output; the corrected result on your system should use the `Documents` path shown above.

Next, inspect the Documents directory:

```
ls -ltr /home/iafzal/Documents/
```

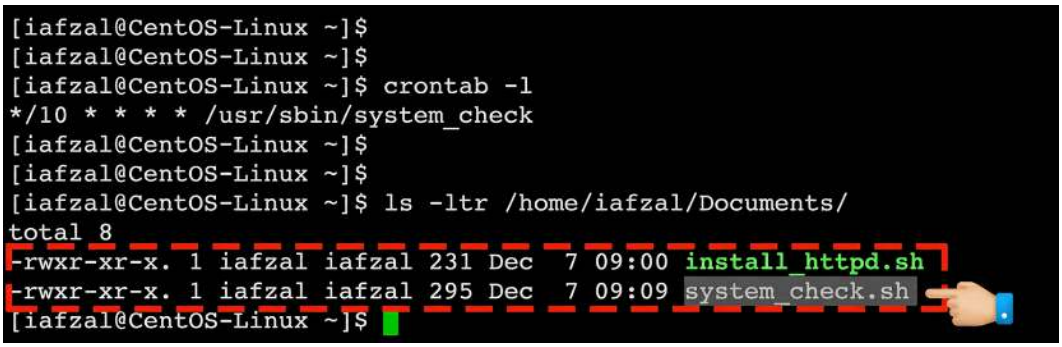


Figure 6.6: Verifying the cron schedule and generated scripts

The listing contains install_httpd.sh and system_check.sh. Both permission strings begin with -rwxr-xr-x, so the owner can read, write, and execute the files, while the group and other users can read and execute them. The exact size of `system_check.sh` will differ from the screenshot because we corrected the generated cron logic before saving the file.

We have now verified both results that Aider created: the script exists with execute permission, and the cron entry exists in the user's crontab. This is a good habit whenever an AI tool changes the system. Do not stop at the generated answer; use ordinary Linux commands to confirm what was actually written or scheduled.

Summary

In this chapter, we introduced Aider as a terminal-based AI assistant that translates plain-English instructions into Bash code, Linux commands, and file edits. We installed Aider with pipx, configured the OpenAI API key, and used --no-git to work without Git integration. We then used Aider to check the firewall status, create and run an Apache installation script, and generate a log-file loop with a corrected cron entry scheduled every ten minutes.

These examples showed how Aider can simplify scripting and system administration while still asking for confirmation before creating files or running commands. However, because `--no-git` does not provide Git commits

or an easy way to undo changes, all generated scripts, commands, and schedules should be reviewed and verified using standard Linux commands.

Knowledge check

Test your understanding of the concepts and commands covered in this chapter.

1. What is Aider primarily used for in this chapter?
   A. Replacing the CentOS kernel
   B. Translating plain-English requests into code, files, and shell commands
   C. Hosting a website without a web server
   D. Managing VirtualBox snapshots
2. Why is pipx used to install aider-chat?
   A. It installs the application in an isolated Python environment
   B. It disables the network connection
   C. It creates the Documents directory
   D. It replaces Git
3. What does the `--no-git` option do?
   A. It uninstalls Git from CentOS
   B. It prevents Aider from using Git for the current workflow
   C. It deletes all Git repositories
   D. It prevents Aider from reading files
4. Which command did Aider propose to check the firewall service?
   A. sudo systemctl status firewalld
   B. sudo firewall --list
   C. nmcli firewall status
   D. ip route firewalld
5. What is the purpose of chmod +x in the two script examples?
   A. To compress the files
   B. To make the files executable
   C. To copy the files into /usr/sbin

D. To add them to Git

6. What does */10 in the minute field of a cron entry mean?
   A. Run at 10:00 only
   B. Run every ten hours
   C. Run every ten minutes
   D. Run on the tenth day of each month
7. What does the generated log loop do as written?
   A. It deletes every log file
   B. It prints each matching log filename and leaves a placeholder for processing
   C. It compresses the logs
   D. It sends the logs to Aider automatically
8. Which command verifies the current user's cron entries?
   A. cron --status
   B. systemctl list-cron
   C. crontab -l
   D. ls /etc/cron

Answers

1. **b. Translating plain-English requests into code, files, and shell commands**. Aider acts as an AI assistant inside the terminal.
2. **a. It installs the application in an isolated Python environment**. This keeps Aider's dependencies separate from the system Python packages.
3. **b. It prevents Aider from using Git for the current workflow**. The startup screen therefore reports no Git repository and a disabled repository map.
4. **a. sudo systemctl status firewalld**. The resulting output shows whether firewalld is loaded, enabled, and active.
5. **b. To make the files executable**. The scripts can then be run by entering their paths.
6. **c. Run every ten minutes**. The remaining asterisks match every hour, day of the month, month, and day of the week.

7. **b. It prints each matching log filename and leaves a placeholder for processing**. No real log-processing command has been added yet.
8. **c. crontab -l**. This lists the cron entries belonging to the current user.

7

Bringing It All Together

We have reached the final chapter, so this is a good time to step back and look at what we have built. We began with a Linux machine and gradually introduced three AI-assisted tools: ChatGPT CLI, Shell Genie, and Aider. Each tool helped us in a different way, but the purpose was always the same: to make Linux administration easier to understand, faster to perform, and less intimidating when we do not remember every command.

In this chapter, we will first recap those tools, with special attention to ChatGPT CLI as the central assistant used throughout the book. Then we will bring everything together in one practical mini project.

The project is intentionally more detailed than a single command. It collects the current users, recent login history, and local account names; writes each result to a log file; displays the logs; and runs again automatically. This gives us a final opportunity to see how an English request becomes a working Linux workflow while we continue to read and verify every part of the result.

In this chapter, we will cover the following topics:

- Reviewing ChatGPT CLI, Shell Genie, and Aider
- Understanding how the three tools fit into one Linux workflow
- Using ChatGPT CLI to generate a user-audit Bash script
- Creating the script and making it executable as root
- Scheduling the script in the root user's crontab
- Verifying the repeated audit output and generated log files

Let's bring everything together!

Technical requirements

To follow the demonstrations in this chapter, make sure the following are available:

- The CentOS Stream virtual machine prepared in *Chapter 2*
- The custom `chatgpt-test.py` terminal program created in *Chapter 3*
- A working OpenAI API key and API billing or credits configured for that program

Throughout the book, we have referred to the custom Python terminal program as ChatGPT CLI. It is the project-specific `chatgpt-test.py` program built with the OpenAI Python SDK, not a separately installed official ChatGPT CLI product. In this final chapter, we continue to use the book's established name while running the file from the user's home directory.

Recapping the AI-assisted Linux toolkit

Before we begin the project, let's quickly recap the three tools. We did not introduce them so that one tool could replace the other two. Each one has a particular strength. Once we understand those strengths, we can choose the tool that fits the task instead of forcing every job through the same interface.

ChatGPT CLI as the central assistant

ChatGPT CLI gave us the greatest flexibility. We used it from the terminal, but we could still explain a problem in natural language. It could create Bash scripts, explain an error, analyze log information, and guide us through a task step by step. We did not have to leave the Linux environment and search for every command separately. We could stay at the prompt, explain what we wanted, and continue the conversation.

That flexibility is useful for both beginners and experienced administrators. A beginner may know the goal but not the exact syntax. An experienced administrator may know the commands but want to save time on a repetitive script or a long troubleshooting sequence. In both cases, ChatGPT CLI is the central interface in this book's workflow: we give it a question, a problem, or a task, and it helps us work toward the result.

It is still important to remember what we have practiced throughout the book. A generated command is a proposal, not proof that the task is complete. We

read the command, understand the paths and privileges it uses, run it when appropriate, and verify the real system state afterward.

Shell Genie for quick commands

Shell Genie is helpful when the requirement is smaller and more direct. We can enter an everyday instruction such as create a new folder, and Shell Genie turns it into a shell command. This makes it a convenient companion when we need a quick command and do not need a longer conversation or a complete script.

In the demonstrations, Shell Genie used the same OpenAI API key that we configured for the other OpenAI-backed tools. This means that the tools can draw on the same model service, although each program is configured and run independently. Shell Genie does not replace ChatGPT CLI. It supports it by handling short command-generation tasks quickly.

Aider for working inside scripts

Aider becomes especially useful when we are already working inside code. It can help us finish a script, correct syntax, create a file, and update code step by step. In *Chapter 6*, we used plain-English requests to create Bash files and then decided whether Aider should apply the file changes and run the proposed shell commands.

Aider also used the configured OpenAI API key in our demonstrations. Its strength is not that it is another name for ChatGPT CLI. Its strength is the editing workflow around files and code, including its ability to work with Git when that integration is enabled.

The simplest way to remember the relationship is this: use Shell Genie for a quick command, use Aider when working inside a script or code project, and return to ChatGPT CLI when we need the widest control over a question, problem, or detailed task. The three tools complement one another. Together, they give us several ways to bring AI assistance into the Linux terminal without taking the final decision away from the administrator.

Mini project: Creating and scheduling a user-audit script

Now let's use ChatGPT CLI for a complete system-administration project. The goal is to create a script that records who is logged in, shows recent login history, lists the user accounts known to the system, and saves those results in separate log files. We will then schedule the script to run every minute and confirm that the scheduled execution is producing output.

This is a useful final example because the request contains several connected tasks. Instead of asking eight separate questions and joining the answers ourselves, we will describe the complete requirement in one prompt. ChatGPT CLI will organize the request into a Bash script and propose a cron entry. We will then create and schedule the script exactly as demonstrated.

Starting ChatGPT CLI as root

Open the saved SSH session for the CentOS machine and log in as root. We use the root account because the script is created and scheduled as an administrative task.

After logging in, list the files in the root user's home directory:

```
ls
```

The directory returns:

```
anaconda-ks.cfg  chatgpt-test.py  create_user.sh
```

> **NOTE**
>
> If you followed the Claude alternative in *Chapter 3*, start `claude-test.py` instead of `chatgpt-test.py` and enter the same project prompt shown below. Claude can generate the same type of Bash user-audit script and cron suggestion. As with any AI-generated administration command, review the output and verify the actual files, permissions, cron schedule, and logs before considering the task complete.

The `chatgpt-test.py` file is the Python-based ChatGPT CLI program created earlier in the book. Start it with:

```
python3 chatgpt-test.py
```

When the program displays `Enter your question:`, it is ready for the project request.

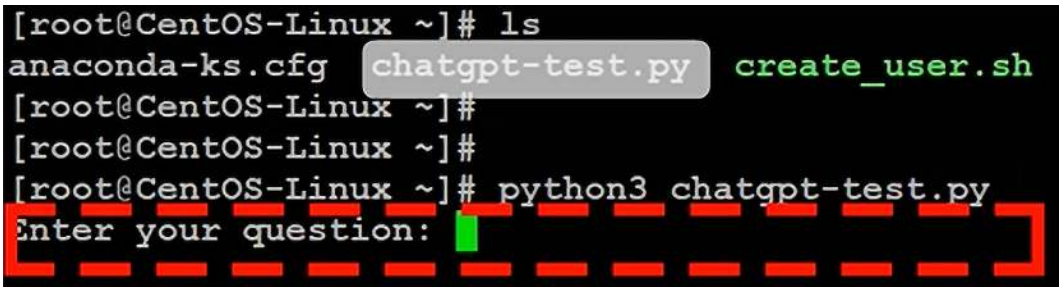


Figure 7.1: Starting the ChatGPT CLI program from the root account

The prompt is long, and that is intentional. We want to see whether the tool can understand a complete scenario rather than a short command. Enter the following request as one prompt:

```
Create a simple Bash script for a Linux system administrator that (1) gets the current date and time,
(2) lists all currently logged-in users using the who command and saves the output with date and time
into /tmp/logged_in_users.log, (3) shows recent login history using the last command and saves it
with date and time into /tmp/login_history.log, (4) displays all system user names by reading the
/etc/passwd file and saves them with date and time into /tmp/system_users.log, (5) automatically
creates all log files if they do not exist, then (6) displays the contents of all three log files on
the screen, (7) verifies the log files by listing them using ls, and finally (8) provides the cron
job line to run this script every 1 minute for a normal user without requiring root access, using
only simple Linux commands and keeping the script easy to understand and suitable for CentOS.
```

Notice how specific the request is. It names the Linux commands, the three output paths, the information that each file should contain, the schedule, and the intended CentOS environment. Those details help the model produce an answer that is closer to the workflow we want.

Reviewing the generated script

ChatGPT CLI responds with a simple, commented Bash script named `user_audit.sh`. The generated script is:

```
#!/bin/bash
# Simple user / login audit script for CentOS

# 1) Get current date and time
NOW="$(date '+%Y-%m-%d %H:%M:%S')"

# Log file paths
LOGGED_IN_LOG="/tmp/logged_in_users.log"
LOGIN_HISTORY_LOG="/tmp/login_history.log"
SYSTEM_USERS_LOG="/tmp/system_users.log"

# 2) Log currently logged-in users (who)
```

```
{
    echo "===== Logged-in users at: $NOW ====="
    who
    echo
} >> "$LOGGED_IN_LOG"

# 3) Log recent login history (last)
{
    echo "===== Login history at: $NOW ====="
    last
    echo
} >> "$LOGIN_HISTORY_LOG"

# 4) Log all system user names from /etc/passwd
{
    echo "===== System users at: $NOW ====="
    # Cut the first field (username) from /etc/passwd
    cut -d: -f1 /etc/passwd
    echo
} >> "$SYSTEM_USERS_LOG"

# 6) Display contents of all three log files
echo "=== Contents of $LOGGED_IN_LOG ==="
cat "$LOGGED_IN_LOG"
echo

echo "=== Contents of $LOGIN_HISTORY_LOG ==="
cat "$LOGIN_HISTORY_LOG"
echo

echo "=== Contents of $SYSTEM_USERS_LOG ==="
cat "$SYSTEM_USERS_LOG"
echo

# 7) Verify log files by listing them
echo "=== Verifying log files in /tmp ==="
ls -l "$LOGGED_IN_LOG" "$LOGIN_HISTORY_LOG" "$SYSTEM_USERS_LOG"
```

Let's walk through the script before creating it. The shebang selects Bash. The `NOW` variable runs date once and stores a timestamp in the format year-month-day `hour:minute:second`. The next three variables give readable names to the paths in `/tmp`. Using variables makes the later commands easier to read and reduces the need to repeat each complete filename.

Each pair of braces groups several commands so that all their output can be redirected together. The >> operator appends that output to the relevant log. If the file does not exist, the shell creates it automatically. That is how

requirement 5 is satisfied even though the generated comments move directly from step 4 to step 6. If the file already exists, the new timestamped section is added instead of replacing the earlier results.

The `who` command records the users who are currently logged in. The last command reads login history from `/var/log/wtmp` and can show user sessions, system boots, and reboots. Its output will vary according to the records retained on the machine and any log rotation. The `cut` command uses a colon as the field separator and prints the first field from every line of `/etc/passwd`. That first field is the username, so the resulting file includes regular users as well as system and service accounts.

After collecting the information, the script uses cat to display all three cumulative log files. Finally, ls -l verifies that the files exist and shows their permissions, owner, group, size, and modification time. This final check is important because it confirms the files on the Linux system rather than relying only on the generated explanation.

After the script, ChatGPT CLI proposes making it executable:

```
chmod +x /path/to/user_audit.sh
```

It then suggests editing the current user's crontab and adding this generic entry:

```
crontab -e
* * * * * /bin/bash /path/to/user_audit.sh >/dev/null 2>&1
```

The five schedule fields all contain an asterisk, so the command is eligible to run every minute. `/bin/bash` runs the script, and the final redirections discard both normal output and error output. The response explains that a normal user can install this entry in that user's own crontab.

The live demonstration now adapts the generic answer to the root-based workflow. The script is saved in `/root`, the root user installs the cron entry, and the output is preserved in a file instead of being discarded.

Creating user_audit.sh

Return to the shell and list the current directory:

```
ls
```

The listing does not yet contain `user_audit.sh` because ChatGPT CLI generated text; it did not create a file on the system.

Create the file with Vim:

```
vim user_audit.sh
```

Enter insert mode, paste the generated script from the previous section, and then save and close the editor. Back at the root prompt, give the script execute permission:

```
chmod +x user_audit.sh
```

Because the current directory is `/root`, the full path to the executable file is `/root/user_audit.sh`. The execute permission is required by the cron line used in the demonstration because that line calls the script directly.

Scheduling the script

Open the root user's crontab:

```
crontab -e
```

Add the exact line used in the demonstration. It intentionally differs from the generic non-root cron line generated earlier:

```
*/1 * * * * /root/user_audit.sh > /root/user_audit_cron.out 2>&1
```

The `*/1` expression in the minute field means every one minute. The remaining fields allow every hour, day of the month, month, and day of the week. The command runs `/root/user_audit.sh` directly. A single `>` sends its normal output to `/root/user_audit_cron.out`, and `2>&1` sends errors to the same file.

Save and exit the crontab editor. The system reports crontab: installing new crontab. In our environment, it also reports that a backup was saved to `/root/.cache/crontab/crontab.bak`. The job is now installed for the root user.

NOTE

The every-minute schedule is used here so that the result can be observed quickly. The job remains installed and continues to run after the demonstration unless the root crontab is edited again.

Verifying the cron output

Immediately after installing the crontab, list the root home directory:

```
ls
```

At first, `user_audit.sh` is present but `user_audit_cron.out` may not be. This is expected if the next minute boundary has not arrived. Wait for one minute and run ls again. The new output file should now appear:

```
anaconda-ks.cfg  chatgpt-test.py  create_user.sh  user_audit_cron.out  user_audit.sh
```

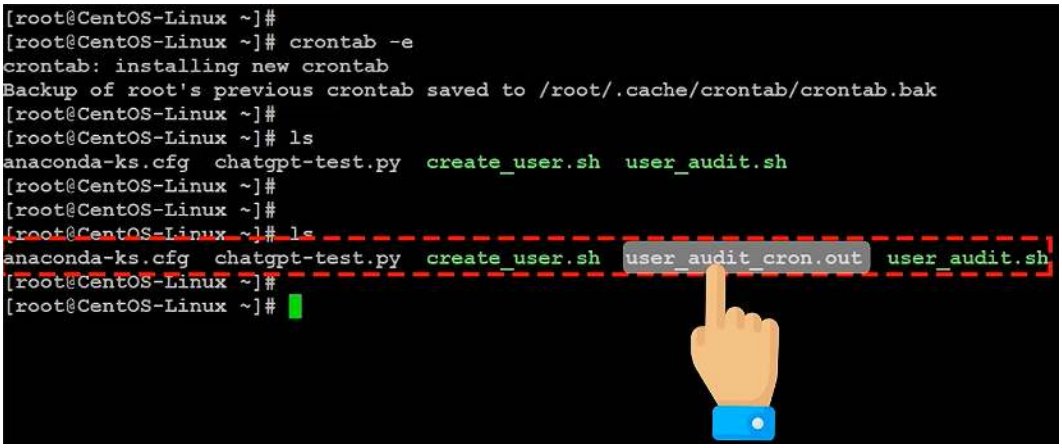


Figure 7.2: Verifying that cron created the audit output file

This before-and-after check proves that cron started the script. Now display the captured output:

```
cat user_audit_cron.out
```

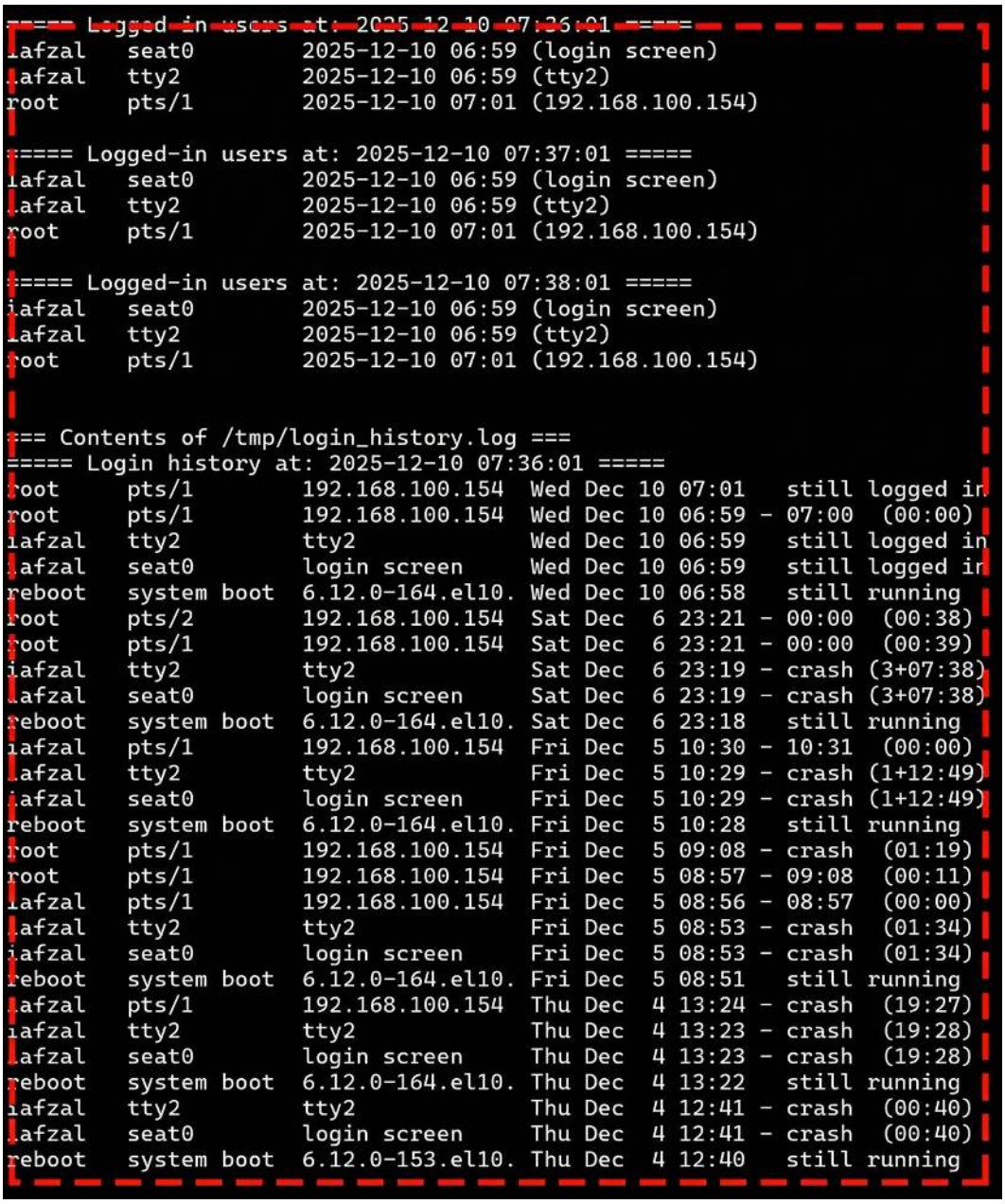


Figure 7.3: Reviewing the timestamped audit output

The output begins with the contents of `/tmp/logged_in_users.log` and contains timestamped sections at `07:36:01`, `07:37:01`, and `07:38:01`. Each section is followed by the output of `who`, so we can see which user sessions existed when that run took place.

The next section displays `/tmp/login_history.log`. Its last output contains root and `iafzal` login sessions, together with system boot and reboot entries. The recorded remote session includes `192.168.100.154`, an address from the private

`192.168.0.0/16` range. It identifies the demonstrated local-network session rather than a publicly routable host. This information helps an administrator see not only who is connected now but also which recent sessions and system starts have been recorded.

The `/tmp/system_users.log` section contains the usernames extracted from `/etc/passwd`. The recorded system includes familiar built-in accounts such as `root`, `bin`, `daemon`, `adm`, `mail`, `nobody`, `dbus`, `sshd`, and `chrony`; service accounts such as `cockpit`, `tcpdump`, `apache`, and `gnome-remote-desktop`; and the demonstrated user accounts `iafzal` and `anderson`. The exact list will vary as packages and users are added to a machine.

At the end of the output, the script verifies the three files with `ls -l`. The demonstrated listing is:

```
-rw-r--r--. 1 root root  630 Dec 10 07:38 /tmp/logged_in_users.log
-rw-r--r--. 1 root root 6183 Dec 10 07:38 /tmp/login_history.log
-rw-r--r--. 1 root root 1068 Dec 10 07:38 /tmp/system_users.log
```

The root owner and group reflect the account under which the cron job ran. The sizes show that the files now contain data, and the 07:38 modification time matches the most recent execution shown in the output. Because the script uses >>, each scheduled run adds another timestamped section to the three `/tmp` files.

> **NOTE**
>
> The demonstrated `-rw-r--r--` permissions allow every local user to read these files. Because the files are stored in the shared `/tmp` directory and contain account and session information, treat this exact layout as a demonstration. On an operational system, review the storage location and restrict access according to the sensitivity of the audit data.

There is one more detail worth noticing. Each run appends a timestamped section to the three `/tmp` logs. The `cron` entry uses a single > for `user_audit_cron.out`, so that output file is replaced on each run rather than appended to. However, the script displays the complete contents of all three growing logs each time, so `user_audit_cron.out` still contains repeated, cumulative information and grows as the source logs grow. That behavior is

part of the demonstrated script and explains why several timestamps appear together.

Removing the every-minute test schedule

The one-minute interval makes the result easy to observe, but the root job will continue running until it is removed. After verifying the output, reopen the root user's crontab:

```
crontab -e
```

Delete the following test entry, then save and close the editor:

```
*/1 * * * * /root/user_audit.sh > /root/user_audit_cron.out 2>&1
```

Confirm that the entry is no longer installed:

```
crontab -l
```

Removing the cron entry stops future executions. The script and the generated log files remain available for inspection until they are removed separately.

This project began with a long English description rather than Bash syntax. ChatGPT CLI identified the separate requirements, created variables for the file paths, grouped the commands, added timestamps, selected append redirection, displayed the results, and proposed a cron schedule. We then moved from the generated answer back to ordinary Linux administration: we created the file, assigned permission, installed the schedule, waited for cron, and inspected the real output.

That is the larger lesson from the book. We do not need to reduce every request to one short sentence, and we do not need to know every command before asking for help. We can describe the outcome carefully, give the tool the paths and constraints it needs, and then use our Linux knowledge to review and verify the result. The AI assistant helps us reach the command or script, but the administrator remains responsible for what runs on the system.

Summary

In this final chapter, we brought together the roles of ChatGPT CLI, Shell Genie, and Aider: ChatGPT CLI for flexible questions and multi-step tasks, Shell Genie for quick command generation, and Aider for working directly

with scripts and code. We then used ChatGPT CLI to generate a user-audit script, created and scheduled it as root, verified the timestamped login and account data written to the three /tmp logs and the captured cron output, and removed the every-minute test schedule. The project reinforced the book's central practice: describe the required outcome clearly, read the generated commands and scripts, and confirm their real effects with standard Linux tools.

Knowledge check

Test your understanding of the tools and the final mini project.

1. Which tool is presented as the central assistant for questions, problems, and detailed tasks?
    A. VirtualBox
    B. ChatGPT CLI
    C. crond
    D. Vim
2. When is Shell Genie most useful in the workflow described in this chapter?
    A. When a quick Linux command is needed from an everyday-language request
    B. When installing the CentOS operating system
    C. When replacing the root account
    D. When editing a Git commit history
3. What does the who command contribute to user_audit.sh?
    A. A list of all files in /tmp
    B. The current date only
    C. A list of currently logged-in users
    D. The cron service status
4. Why do the three log files appear automatically if they do not already exist?
    A. The >> append redirection creates a missing destination file
    B. The ls command creates each file

C. The root account creates every file at login
D. Vim creates files in /tmp automatically

5. What does cut -d: -f1 /etc/passwd display?
   A. The password of each user
   B. The first colon-separated field, which contains each username
   C. Only currently logged-in users
   D. The last login time for root
6. Which cron entry is installed in the demonstrated root workflow?
   A. */10 * * * * /usr/sbin/system_check
   B. @reboot /root/user_audit.sh
   C. */1 * * * * /root/user_audit.sh > /root/user_audit_cron.out 2>&1
   D. 0 0 * * * /tmp/system_users.log
7. Why does user_audit_cron.out contain repeated information over time?
   A. cron runs the script repeatedly, and the script displays cumulative log contents
   B. The who command always runs three times
   C. chmod duplicates the file
   D. crontab copies /etc/passwd every second
8. What final habit does the project emphasize after AI generates a command or script?
   A. Delete the terminal history immediately
   B. Run every suggestion without reading it
   C. Verify the real files, permissions, schedules, services, or output with Linux commands
   D. Replace Bash with Python for every task

Answers

1. **b. ChatGPT CLI**. It is presented as the central interface for flexible questions, troubleshooting, and detailed tasks.
2. **a. When a quick Linux command is needed from an everyday-language request**. Shell Genie complements the longer ChatGPT CLI workflow.

3. **c. A list of currently logged-in users**. The script appends the who output beneath a timestamped heading.
4. **a. The >> append redirection creates a missing destination file**. It then adds new output to the end on later runs.
5. **b. The first colon-separated field, which contains each username**. This includes regular users and system or service accounts.
6. **c. */1 * * * * /root/user_audit.sh > /root/user_audit_cron.out 2>&1**. It runs the root-owned script every minute and captures normal and error output in one file.
7. **a. cron runs the script repeatedly, and the script displays cumulative log contents**. Each run therefore captures earlier entries together with the newest one.
8. **c. Verify the real files, permissions, schedules, services, or output with Linux commands**. The generated explanation is only the starting point for system validation.

Agentic Linux Administration

1. Welcome to Packt Early Access
    1. Agentic Linux Administration: Automate, Troubleshoot, and Secure Linux with ChatGPT, Claude, Aider, and AI CLI Tools
2. Chapter 1: Foundations of AI-Driven Linux System Management
    1. Understanding AI and Linux
    2. How AI can help system administrators
    3. How AI is transforming IT and DevOps
        1. Netflix
        2. Amazon
        3. Google
    4. Summary
    5. Knowledge check
        1. Answers
3. Chapter 2: Preparing Your System Workspace
    1. Technical requirements
    2. Installing VirtualBox
    3. Creating and installing the CentOS virtual machine
        1. Downloading the CentOS Stream ISO
        2. Creating the virtual machine
        3. Configuring the virtual machine settings
        4. Installing CentOS Stream
    4. Connecting through PuTTY
    5. Exploring the Linux environment
        1. Checking the user and system identity

        2. Exploring the Linux filesystem
        3. Navigating between directories
        4. Creating and reading a file
        5. Checking basic internet connectivity
        6. Reading system logs
        7. Working with elevated privileges
        8. Inspecting and modifying permissions
    6. Verifying network connectivity
        1. Testing DNS resolution
        2. Restarting and checking NetworkManager
    7. Reviewing AI tools for system administrators
    8. Summary
    9. Knowledge check
        1. Answers
4. Chapter 3: Using ChatGPT/Claude from the Linux Command Line
    1. Technical requirements
    2. Understanding ChatGPT CLI
        1. Why a command-line AI workflow is useful
    3. Installing Python, pip, and the OpenAI SDK
    4. Creating an OpenAI API key
        1. Exporting the API key in Linux
        2. For those using Claude instead of ChatGPT
        3. Reviewing the OpenAI Python example
    5. Sending the first request from the terminal
    6. Making the program interactive
    7. Summary
    8. Knowledge check
        1. Answers

5. Chapter 4: Using ChatGPT CLI for Linux Administration
    1. Technical requirements
    2. Generating a system-health script
    3. Troubleshooting a permission error
    4. Creating and scheduling a disk-monitoring job
    5. Interpreting system logs with AI
    6. Reviewing security-hardening recommendations
        1. Trying one focused control with AIDE
    7. Creating a user-account script safely
    8. Summary
    9. Knowledge check
        1. Answers
6. Chapter 5: Generating Linux Commands with Shell Genie
    1. Technical requirements
    2. Understanding Shell Genie
        1. How Shell Genie complements ChatGPT CLI
    3. Installing and configuring Shell Genie on CentOS
        1. Checking Python and pip
        2. Installing pipx and Shell Genie
        3. Resolving the Click compatibility error
        4. Selecting the backend and saving the configuration
        5. Testing Shell Genie by installing Apache
    4. Generating commands from English
        1. Compressing the Documents directory
        2. Copying an archive before deleting the original
        3. Changing the user password
    5. Creating scripts and interacting with the system

        1. Creating and running an EPEL installation script
        2. Creating and running a system-check script
    6. Summary
    7. Knowledge check
        1. Answers
7. Chapter 6: Using Aider for Bash Scripting and Linux Automation
    1. Technical requirements
    2. Understanding Aider and when to use it
        1. Using Aider with Git
        2. Working without Git
    3. Installing and configuring Aider
        1. Installing Aider with pipx
        2. Providing the OpenAI API key
        3. Starting Aider without Git
        4. Testing the connection
    4. Creating and running a Bash snippet
        1. Approving the file and shell commands
    5. Generating loops, cron jobs, and script templates
        1. Verifying the cron entry and script
    6. Summary
    7. Knowledge check
        1. Answers
8. Chapter 7: Bringing It All Together
    1. Technical requirements
    2. Recapping the AI-assisted Linux toolkit
        1. ChatGPT CLI as the central assistant
        2. Shell Genie for quick commands

    3. Aider for working inside scripts
3. Mini project: Creating and scheduling a user-audit script
    1. Starting ChatGPT CLI as root
    2. Reviewing the generated script
    3. Creating user_audit.sh
    4. Scheduling the script
    5. Verifying the cron output
    6. Removing the every-minute test schedule
4. Summary
5. Knowledge check
    1. Answers