CHAPTER 4

More Commands

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

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

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

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

A. Pajankar, *Practical Linux with Raspberry Pi OS*,
https://doi.org/10.1007/978-1-4842-6510-9_4

Configuring the RPi Board

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

```
sudo raspi-config
```

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

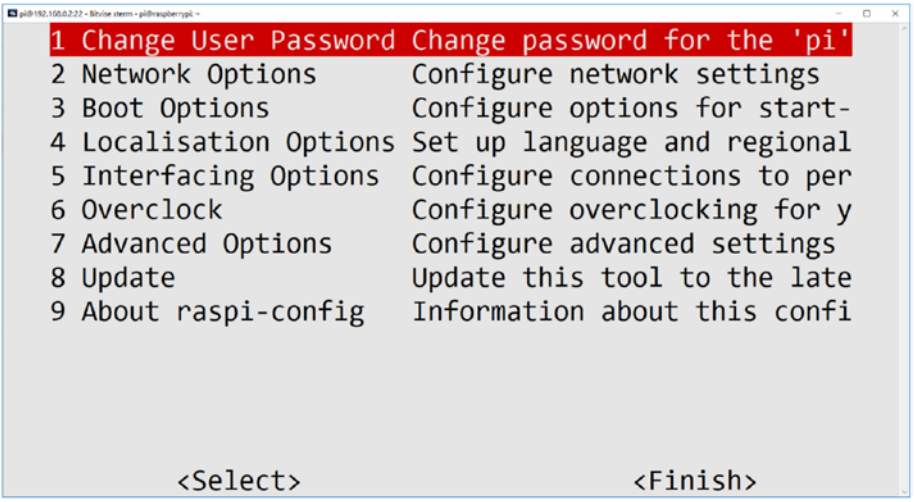

Figure 4-1. *Raspberry Pi configuration utility at the command prompt*

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

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

What Is sudo?

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

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

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

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

Getting Help on Commands

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

```
man ls
```

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

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

```
info ls
```

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

Network-Related Commands

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

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

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

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

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

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

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

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

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

lo        no wireless extensions.

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

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

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

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

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

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

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

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

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

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

Commands: File Operations

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

```
touch abc
mkdir practice
```

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

```
cp abc abc1
```

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

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

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

We can copy it into a folder as follows:

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

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

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

```
mv abc1 abc2
```

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

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

```
cat abc
```

It will show the contents of the file:

```
head abc
```

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

```
head -5 abc
```

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

```
tail abc
tail -5 abc
```

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

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

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

Printing a String

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

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

Control Operators

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

Run the following commands in sequence:

```
ls
echo $?
```

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

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

```
echo test1 ; echo test2
```

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

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

```
leafpad &
```

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

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

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

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

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

In this case, both the commands are not executed.

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

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

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

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

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

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

Filename Globbing

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

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

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

```
ls a?c
```

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

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

```
ls a*
```

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

```
ls *.txt
```

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

Command: History

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

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

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

Pipes

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

```
history | tail -10
```

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

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

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

Summary

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

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