Introduction

Unix-like systems are amazing operating system designs. Even after many decades, Unix-style architecture for operating systems serves as one of the best designs. One of the important features of this architecture is the command-line interface, or the shell. The shell environment helps users to interact with and access core functions of the operating system. The term **scripting** is more relevant in this context. Scripting is usually supported by interpreter-based programming languages. Shell scripts are files in which we write a sequence of commands that we need to perform and are executed using the shell utility.

In this book we are dealing with **Bash** (**Bourne Again Shell**), which is the default shell environment for most GNU/Linux systems. Since GNU/Linux is the most prominent operating system on Unix-style architecture, most of the examples and discussions are written by keeping Linux systems in mind.

The primary purpose of this chapter is to give readers an insight into the shell environment and become familiar with the basic features that the shell offers. Commands are typed and executed in a shell terminal. When a terminal is opened, a prompt is available which usually has the following format:

```
username@hostname$
```

Or:

```
root@hostname #
```

or simply as `$` or `#`.

`$` represents regular users and `#` represents the administrative user root. Root is the most privileged user in a Linux system.

It is usually a bad idea to directly use the shell as the root user (administrator) to perform tasks. This is because typing errors in your commands have the potential to do more damage when your shell has more privileges. So, it is recommended to log in as a regular user (your shell will denote that as `$` in the prompt, and `#` when running as root), and then use tools such as `` `sudo' `` to run privileged commands. Running a command such as `sudo <command> <arguments>` will run it as root.

A shell script is a text file that typically begins with a shebang, as follows:

```
#!/bin/bash
```

Shebang is a line on which `#!` is prefixed to the interpreter path. `/bin/bash` is the interpreter command path for Bash.

Execution of a script can be done in two ways. Either we can run the script as a command-line argument to `bash` or we can grant execution permission to the script so it becomes executable.

The script can be run with the filename as a command-line argument as follows (the text that starts with # is a comment, you don't have to type it out):

```
$ bash script.sh # Assuming script is in the current directory.
```

Or:

```
$ bash /home/path/script.sh # Using full path of script.sh.
```

If a script is run as a command-line argument for `bash`, the shebang in the script is not required.

If required, we can utilize the shebang to facilitate running the script on its own. For this, we have to set executable permissions for the script and it will run using the interpreter path that is appended to `#!` to the shebang. This can be set as follows:

```
$ chmod a+x script.sh
```

This command gives the `script.sh` file the executable permission for all users. The script can be executed as:

```
$ ./script.sh #./ represents the current directory
```

Or:

```
$ /home/path/script.sh # Full path of the script is used
```

The kernel will read the first line and see that the shebang is `#!/bin/bash`. It will identify `/bin/bash` and execute the script internally as:

```
$ /bin/bash script.sh
```

When a shell is started, it initially executes a set of commands to define various settings such as prompt text, colors, and much more. This set of commands are read from a shell script at `~/.bashrc` (or `~/.bash_profile` for login shells) located in the home directory of the user. The Bash shell also maintains a history of commands run by the user. It is available in the `~/.bash_history` file.

`~` denotes your home directory, which is usually `/home/user` where `user` is your username or `/root` for the root user.

A login shell is the shell which you get just after logging in to a machine. However, if you open up a shell while logged in to a graphical environment (such as GNOME, KDE, and so on), then it is not a login shell.