Learning Linux Shell Scripting

Shell scripting is used to automate day-to-day administration and for testing or product development tasks.

This book covers Bash (GNU Bourne-again shell) preparing you for the exciting world of Linux Shell scripting.

Firstly, we introduce you to the shell environment and explain its basic commands. Next we move on to checking, killing, and controlling the execution of processes in Linux OS. After this, we teach you about the filter tools available in Linux and explain standard output and standard error devices.

Next, you’ll get to grips with some real-world essentials, such as debugging and performing shell arithmetic fluently. Then you’ll take the next step and learn advanced topics in Shell scripting, such as starting up a system and customizing a Linux system.

Finally, you’ll get to know the capabilities of scripting and learn about Grep, stream editor, and Awk.

Who this book is written for
This book is ideal for those who are proficient at working with Linux and who want to learn about shell scripting to improve their efficiency and practical skills. By the end of this book, you will be able to confidently use your own Shell scripts in the real world.

What you will learn from this book
- Familiarize yourself with the various text filtering tools available in Linux
- Combine the fundamental text and file processing commands to process data and automate repetitive tasks
- Understand expressions and variables and how to use them practically
- Automate decision-making and save a lot of time and effort spent revisiting code
- Get to grips with advanced functionality, such as using traps and signals, and using dialogs to develop screens
- Start up a system and customize a Linux system
- Take an in-depth look at regular expressions and pattern matching to understand the capabilities of scripting

Who this book is written for
This book is ideal for those who are proficient at working with Linux and who want to learn about shell scripting to improve their efficiency and practical skills. By the end of this book, you will be able to confidently use your own Shell scripts in the real world.

What you will learn from this book
- Familiarize yourself with the various text filtering tools available in Linux
- Combine the fundamental text and file processing commands to process data and automate repetitive tasks
- Understand expressions and variables and how to use them practically
- Automate decision-making and save a lot of time and effort spent revisiting code
- Get to grips with advanced functionality, such as using traps and signals, and using dialogs to develop screens
- Start up a system and customize a Linux system
- Take an in-depth look at regular expressions and pattern matching to understand the capabilities of scripting

Unleash the power of Shell scripts to solve real-world problems by breaking through the practice of writing tedious code
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Getting Started and Working with Shell Scripting'
- A synopsis of the book’s content
- More information on Learning Linux Shell Scripting
Ganesh Sanjiv Naik is an author, consultant, and corporate trainer for embedded Android, embedded Linux, and Internet of Things related product development.

He completed his computer engineering in 1988. Since then, he has worked in this industry. He has worked on projects including micro-controller based projects to advanced Embedded Android projects. He has more than 20 years of professional experience and project accomplishment in information technology.

Ganesh has a passion and deep desire for teaching. He has trained 1,000 engineers in Linux and Android product development. He has developed a lot of training material as well as curriculum for various universities and training institutes.

He has an interest in spiritual study and practices such as meditation. He is a certified yoga teacher. His hobbies include yoga and martial arts.

He has worked as a corporate trainer for Indian Space Research Organization, Intel, GE, Samsung, Motorola, Penang Skill Development Center (Malaysia), various companies in Singapore as well as various other corporates in India and other countries. He has started a company called Levana Technologies, which works with the Penang Skill Development Center (Malaysia) for consulting and training activities. If you would like to send feedback, suggestions, or corrections in the book, he can be contacted at https://in.linkedin.com/in/naikganesh.

This book is his real-life experience....
He has worked as a consultant and corporate trainer in the following skills:

- Internet of Things
- Embedded Android, Android internals, and device driver development
- USB and PCI device driver development in Linux
- Embedded Linux and device driver development
- Unix Shell scripting with sed and awk
- Embedded C++ and C programming
- Operating systems, software engineering, and networking
- Problem solving — analysis, reasoning, and solution techniques for software engineers
Preface

Shell scripts are an essential part of any modern operating system, such as UNIX, Linux, Windows, and similar. The scripting language or its syntax may vary from OS to OS; but the fundamental principles remain the same. I first encountered Linux Shell scripts during the development of embedded Linux product development. Shell scripts were initializing the complete product from the basic booting procedure until users logged in and a complete operating system was initialized. Another situation was in the automation of regular activities, such as the build and release management of source codes of very complex products, where more than 10,000 files were a part of a single project. Similarly, another very common requirement comes while using the make utility, which is used to compile and build complex product source codes.

Initially, I had learned scripts to solve practical problems and customize already existing products. This book is the summary of what I have learned over the years in Linux Shell scripting during project development work, consultancy, and corporate trainings and their Q&A sessions.

In this book, you will learn about the very basics of Shell scripting to complex, customized automation. By the end of this book, you will be able to confidently use your own Shell scripts for the real-world problems out there. The idea is to be as practical as possible and give you the look and feel of what real-world scripting looks like.

This book covers bash, the GNU Bourne-Again Shell scripting. You can use the knowledge gained by reading this book for any shell of any of the UNIX flavors or distributions. You will need to take care of a few syntax changes if you are working in other shells, such as Korn, and similar. You should be able to read this book cover to cover, or you can just pick it up and read anything that you might find interesting. But perhaps most importantly, if you have a question about how to solve a particular problem or you need a hint, you will find it easy to find the right solution or something close enough to save your time and energy.
What this book covers

Chapter 1, Getting Started and Working with Shell Scripting, you will learn different ways to write and run Shell scripts. You will also learn ways to handle files and directories, and you will learn about working with permissions.

Chapter 2, Drilling Deep into Process Management, Job Control, and Automation, you will learn about basic process management. You will learn about command ps and job management using commands such as jobs, fg, bg, kill, and pkill. Later on, you will learn about process monitoring tools: top, iostat, vmstat and sar.

Chapter 3, Using Text Processing and Filters in Your Scripts, you will learn about using more, less, head, and tail commands. You will also learn text processing tools such as, cut, paste, comm, and uniq. You will learn about standard input, output, and error. Later on, you will learn about metacharacters and pattern matching using vi and grep.

Chapter 4, Working with Commands, you will learn about how shell interprets any command entered on the command line. You will also learn command substitution, separators, and pipes in detail.

Chapter 5, Exploring Expressions and Variables, you will learn about variables — environment variables. This will include how to export environment variables, set, shift, read-only variables, command-line arguments, and about creating and handling arrays.

Chapter 6, Neat Tricks with Shell Scripting, you will learn about debugging, the here operator, interactive Shell scripts for taking input from the keyboard, and file handling.

Chapter 7, Performing Arithmetic in Shell Scripts, you will learn about doing arithmetic operations in various ways, such as using declare, let, expr, and arithmetic expressions. You will also learn about representing numbers in different bases, such as hex, octal, and binary. You will learn about using the bc utility for doing floating point or fractional arithmetic.

Chapter 8, Automating Decision Making in Scripts, you will learn about using decision making in scripts working with test, if...else, and switching case. You will also use select for loop with menu. For repeating tasks such as processing lists, you will learn about using for loop, while loop and do while. You will also learn about how to control loops using break and continue statements.
Chapter 9, Working with Functions, we will understand about functions in Shell scripts. You will learn about the definition and display of functions by removing the function from the shell. You will also learn about passing arguments to functions, sharing data between functions, declaring local variables in functions, returning results from functions, and running functions in background. You will finally learn about using source and . commands. We will use these commands to use the library of functions.

Chapter 10, Using Advanced Functionality in Scripts, you will learn about using traps and signals. You will also learn about creating menus with the help of the dialog utility.

Chapter 11, System Startup and Customizing a Linux System, you will learn about Linux system startup from power on until the user login and how to customize a Linux system environment.

Chapter 12, Pattern Matching and Regular Expressions with sed and awk, you will learn about regular expressions and using sed (stream editor) and awk for text processing. You will learn about the usage of various commands and options along with a lot of examples for using sed and awk.
Getting Started and Working with Shell Scripting

Whoever works with Linux will come across shell as the first program to work with. The Graphical user interface (GUI) usage has become very popular due to the ease of use. Those who want to take advantage of the power of Linux will use the shell program by default.

Shell is a program which provides the user direct interaction with the operating system. Let’s understand the stages in the evolution of the Linux operating system. Linux was developed as a free and open source substitute for UNIX OS. The chronology can be as follows:

- The UNIX operating system was developed by Ken Thomson and Dennis Ritchie in 1969. It was released in 1970. They rewrote the UNIX using C language in 1972.
- In 1991, Linus Torvalds developed the Linux kernel for the free operating system.

In this chapter, we will cover the following topics:

- Comparison of shells
- Working in shell
- Learning basic Linux commands
- Our first script—Hello World
- Compiler and interpreter—difference in process
- When not to use scripts
- Various directories
- Working more effectively with shell—basic commands
- Working with permissions
Comparison of shells

Initially, the UNIX OS used a shell program called Bourne Shell. Then eventually, many more shell programs were developed for different flavors of UNIX. The following is brief information about different shells:

- Sh — Bourne Shell
- Csh — C Shell
- Ksh — Korn Shell
- Tcsh — enhanced C Shell
- Bash — GNU Bourne Again Shell
- Zsh — extension to Bash, Ksh, and Tcsh
- Pdksh — extension to KSH

A brief comparison of various shells is presented in the following table:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Bourne</th>
<th>C</th>
<th>TC</th>
<th>Korn</th>
<th>Bash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Command-line editing</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Advanced pattern matching</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Filename completion</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Directory stacks (pushd and popd)</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>History</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Functions</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Key binding</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Job control</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Spelling correction</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Prompt formatting</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

What we see here is that, generally, the syntax of all these shells is 95% similar. In this book, we are going to follow Bash shell programming.
Tasks done by shell
Whenever we type any text in the shell terminal, it is the responsibility of shell to execute the command properly. The activities done by shell are as follows:

- Reading text and parsing the entered command
- Evaluating metacharacters such as wildcards, special characters, or history characters
- Process io-redirection, pipes, and background processing
- Signal handling
- Initializing programs for execution

We will discuss the preceding topics in the subsequent chapters.

Working in shell
Let's get started by opening the terminal, and we will familiarize ourselves with the Bash Shell environment:

1. Open the Linux terminal and type in:
   ```
   $ echo $SHELL
   /bin/bash
   ```
2. The preceding output in terminal says that the current shell is /bin/bash such as BASH shell:
   ```
   $ bash --version
   GNU bash, version 2.05.0(1)-release (i386-redhat-linux-gnu)
   Copyright 2000 Free Software Foundation, Inc.
   ```

Hereafter, we will use the word Shell to signify the BASH shell only. If we intend any other shell, then it will be specifically mentioned by name such as KORN and similar other shells.

In Linux, filenames in lowercase and uppercase are different; for example, the files Hello and hello are two distinct files. This is unlike Windows, where case does not matter.

As far as possible, avoid using spaces in filenames or directory names such as:

- Wrong file name—Hello World.txt
- Correct file name—Hello_World.txt or HelloWorld.txt
This will make certain utilities or commands fail or not work as expected, for example, the `make` utility.

While typing in filenames or directory names of the existing files or folders, use the tab completion feature of Linux. This will make working with Linux faster.

## Learning basic Linux commands

The following table lists a few basic Linux commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$ ls</code></td>
<td>This command is used to check the contents of the directory.</td>
</tr>
<tr>
<td><code>$ pwd</code></td>
<td>This command is used to check the present working directory.</td>
</tr>
<tr>
<td><code>$ mkdir work</code></td>
<td>We will work in a separate directory called <code>work</code> in our home directory. Use this command to create a new directory called <code>work</code> in the current folder.</td>
</tr>
<tr>
<td><code>$ cd work</code></td>
<td>This command will change our working directory to the newly created directory <code>work</code>.</td>
</tr>
<tr>
<td><code>$ pwd</code></td>
<td>This command can be used to verify whether we moved to the expected directory.</td>
</tr>
<tr>
<td><code>$ touch hello.sh</code></td>
<td>This command is used to create a new empty file called <code>hello.sh</code> in the current folder.</td>
</tr>
<tr>
<td><code>$ cp hello.sh bye.sh</code></td>
<td>This command is used to copy one file into another file. This will copy <code>hello.sh</code> as <code>bye.sh</code>.</td>
</tr>
<tr>
<td><code>$ mv bye.sh welcome.sh</code></td>
<td>This command is used to rename a file. This will rename <code>bye.sh</code> as <code>welcome.sh</code>.</td>
</tr>
<tr>
<td><code>$ ll</code></td>
<td>This command will display detailed information about files.</td>
</tr>
<tr>
<td><code>$ mv welcome.sh .welcome.sh</code></td>
<td>Let's see some magic. Rename the file using the <code>mv</code> command and the run the <code>ls</code> command. Now, the <code>ls</code> command will not display our file <code>welcome.sh</code>. That file gets hidden. Any filename or directory name starting with &quot;.&quot; (dot) becomes hidden.</td>
</tr>
<tr>
<td><code>$ ls -a</code></td>
<td>This command is used to see hidden files.</td>
</tr>
<tr>
<td><code>$ rm .welcome.sh</code></td>
<td>This command is used to delete the file.</td>
</tr>
</tbody>
</table>
If we delete any file from GUI such as Graphical User Interface, then it will be moved to the /home/user/.local/share/Trash/files/ all deleted files folder.

Our first script – Hello World

Since we learned basic commands to use Linux OS, we will now write our first Shell script called hello.sh. You can use any editor of your choice such as vi, gedit, nano, and other similar editors. I prefer to use the vi editor.

1. Create a new hello.sh file as follows:
   ```bash
   #!/bin/bash
   # This is comment line
   echo "Hello World"
   ls
   date
   ```

2. Save the newly created file.

The #!/bin/bash line is called the shebang line. The combination of the characters # and ! is called the magic number. The shell uses this to call the intended shell such as /bin/bash in this case. This should always be the first line in a Shell script.

The next few lines in the Shell script are self explanatory.

- Any line starting with #, will be treated as a comment line. An exception to this would be the first line with #!/bin/bash
- The echo command will print Hello World on the screen
- The ls command will display directory content on the console
- The date command will show the current date and time

We can execute the newly created file by the following commands:

- Technique one:
  ```
  $ bash hello.sh
  ```
- Technique two:
  ```
  $ chmod +x hello.sh
  ```
By running any of the preceding commands, we are adding executable permission to our newly created file. You will learn more about file permissions in later in this same chapter.

$ ./hello.sh

By running the preceding command, we are executing hello.sh as the executable file. By technique one, we passed filename as an argument to Bash shell.

The output of executing hello.sh will be as follows:

Hello World
hello.sh
Sun Jan 18 22:53:06 IST 2015

Since we have successfully executed our first script, we will proceed to develop a more advanced script, hello1.sh. Please create the new script hello.sh as follows:

#!/bin/bash
# This is the first Bash shell
# Scriptname : Hello1.sh
# Written by: Ganesh Naik
echo "Hello $LOGNAME, Have a nice day!"
echo "Your are working in directory `pwd`.
echo "You are working on a machine called `uname -n`.
echo "List of files in your directory is.
ls  # List files in the present working directory
echo "Bye for now $LOGNAME. The time is `date +%T`!"

The output of executing hello.sh will be as follows:

Hello student, Have a nice day!.
Your are working in directory /home/student/work.
You are working on a machine called ubuntu.
List of files in your directory is.
hello1.sh  hello.sh
Bye for now student. The time is 22:59:03!

You will learn about the LOGNAME, uname, and other similar commands as we go on with the book.
Compiler and interpreter – difference in process

In any program development, the following are the two options:

• **Compilation**: Using a compiler-based language such as C, C++, Java, and other similar languages

• **Interpreter**: Using interpreter-based languages such as Bash Shell scripting.

When we use a compiler-based language, we compile the complete source code, and as a result of compilation, we get a binary executable file. We then execute the binary to check the performance of our program.

On the contrary, when we develop the Shell script, such as an interpreter-based program, every line of the program is input to Bash shell. The lines of Shell script are executed one by one sequentially. Even if the second line of a script has an error, the first line will be executed by the shell interpreter.

When not to use scripts

Shell scripts have certain advantages over compiler-based programs, such as C or C++ language. However, Shell scripting has certain limitations as well.

The following are the advantages:

• Scripts are easy to write

• Scripts are quick to start and easy for debugging

• They save the time of development.

• Tasks of administration are automated

• No additional setup or tools are required for developing or testing

Shell script

The following are the limitations of Shell scripts:

• Every line in Shell script creates a new process in the operating system. When we execute the compiled program such as C program, it runs as a single process for the complete program.

• Since every command creates a new process, Shell scripts are slow as compared to compiled programs.

• Shell scripts are not suitable if heavy math operations are involved.
Getting Started and Working with Shell Scripting

- There are Problems with cross-platform portability.
- We cannot use Shell scripts in the following situations when:
  - Extensive file operations are required
  - We need data structures, such as linked lists or trees
  - We need to generate or manipulate graphics or GUIs
  - We need direct access to system hardware
  - We need a port or socket I/O
  - We need to use libraries or interface with legacy code
  - Proprietary, closed source applications are used (Shell scripts put the source code right out in the open for the entire world to see)

Various directories
We will explore the directory structure in Linux so that it will be useful later on:

- `/bin/`: This contains commands used by a regular user.
- `/boot/`: The files required for the operating system startup are stored here.
- `/cdrom/`: When CD-ROM is mounted, the CD-ROM files are accessible here.
- `/dev/`: The device driver files are stored in this folder. These device driver files will point to hardware-related programs running in kernel.
- `/etc/`: This folder contains configuration files and startup scripts.
- `/home/`: This folder contains a home folder of all users except the administrator.
- `/lib/`: The library files are stored in this folder.
- `/media/`: External media such as a USB pen drive is mounted in this folder.
- `/opt/`: The optional packages are installed in this folder.
- `/proc/`: This contains files which give information about kernel and every process running in OS.
- `/root/`: This is the administrators home folder.
- `/sbin/`: This contains commands used by the administrator or root user.
- `/usr/`: This contains secondary programs, libraries, and documentation about user-related programs.
- `/var/`: This contains variable data such as http, tftp, and similar other.
- `/sys/`: This dynamically creates the `sys` files
Working more effectively with shell – basic commands

Let us learn a few commands, which are required very often, such as `man`, `echo`, `cat` and similar:

- Enter the following command. It will show the various types of manual pages displayed by the `man` command:
  ```bash
  $ man man
  ```

  From the following table, you can get an idea about various types of `man` pages for the same command:

<table>
<thead>
<tr>
<th>Section number</th>
<th>Subject area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User commands</td>
</tr>
<tr>
<td>2</td>
<td>System calls</td>
</tr>
<tr>
<td>3</td>
<td>Library calls</td>
</tr>
<tr>
<td>4</td>
<td>Special files</td>
</tr>
<tr>
<td>5</td>
<td>File formats</td>
</tr>
<tr>
<td>6</td>
<td>Games</td>
</tr>
<tr>
<td>7</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>8</td>
<td>System admin</td>
</tr>
<tr>
<td>9</td>
<td>Kernel routines</td>
</tr>
</tbody>
</table>

- We can enter the `man` command to display corresponding manual pages as follows:
  ```bash
  $ man 1 command
  $ man 5 command
  ```

- Suppose we need to know more about the `passwd` command, which is used for changing the current password of a user, you can type the command as follows:
  ```bash
  $ man command
  man -k passwd // show all pages with keyword
  man -K passwd // will search all manual pages for pattern
  $ man passwd
  ```
This will show information about the `passwd` command:

```bash
$ man 5 passwd
```

The preceding command will give information about the file `passwd`, which is stored in `/etc/passwd`.

- We can get brief information about the command as follows:
  ```bash
  $ whatis passwd
  
  Output:
  passwd (1ssl)        - compute password hashes
  passwd (1)           - change user password
  passwd (5)           - the password file
  
  • Every command we type in the terminal has an executable binary program file associated with it. We can check the location of a binary file as follows:
    ```bash
    $ which passwd
    /usr/bin/passwd
    
    The preceding line tells us that the binary file of the `passwd` command is located in the `/usr/bin/passwd` folder.
    
    • We can get complete information about the binary file location as well as manual page location of any command by following:
      ```bash
      $ whereis passwd
      
      The output will be as follows:
      passwd: /usr/bin/passwd /etc/passwd /usr/bin/X11/passwd /usr/share/man/man1/passwd.1.gz /usr/share/man/man1/passwd.1ssl.gz /usr/share/man/man5/passwd.5.gz
      
      • Change the user login and effective user name:
        ```bash
        $ whoami
        
        This command displays the user name of the logged in user:
        ```bash
        $ su
        
        The `su` command (switch user) will make the user as the administrator; but, you should know the administrators, password. The `sudo` command (superuser do) will run the command with administrator's privileges. It is necessary that the user should have been added in the `sudoers` list.
        ```bash
        # who am i
        
        This command will show the effective user who is working at that moment.
        # exit
• Many a times, you might need to create new commands from existing commands. Sometimes, existing commands have complex options to remember. In such cases, we can create new commands as follows:

```bash
$ alias ll='ls -l'
$ alias copy='cp -rf'
```

To list all declared aliases, use the following command:

```bash
$ alias
```

To remove an alias, use the following command:

```bash
$ unalias copy
```

• We can check about the operating system details such as UNIX/Linux or the distribution that is installed by the following command:

```bash
$ uname
```

Output:

```
Linux
```

This will display the basic OS information (UNIX name)

• Linux kernel version information will be displayed by the following:

```bash
$ uname -r
```

Output:

```
3.13.0-32-generic
```

• To get all the information about a Linux machine, use the following command:

```bash
$ uname -a
```

Output:

```
Linux ubuntu 3.13.0-32-generic #57-precise1-Ubuntu SMP Tue Jul 15 03:50:54 UTC 2014 i686 i686 i386 GNU/Linux
```

• The following commands will give you more information about the distribution of Linux:

```bash
$ cat /proc/version   // detailed info about distribution
$ cat /etc/*release
# lsb_release -a      // will tell distribution info for Ubuntu
```

The command `cat` is used for reading files and displayed on the standard output.
• Sometimes, we need to copy a file or directory in many places. In such situations, instead of copying the original file or directory again and again, we can create soft links. In Windows, a similar feature is called as creating a shortcut.

$ ln -s file file_link

• To learn about the type of file, you can use the command file. In Linux, various types of files exist. Some examples are as follows:
  - Regular file (-)
  - Directory (d)
  - Soft link (l)
  - Character device driver (c)
  - Block device driver (b)
  - Pipe file (p)
  - Socket file (s)

• We can get information about a file using the following command:
  $ file   fil_name  // show type of file

• Printing some text on the screen for showing results to the user or to ask details is an essential activity.
  - The following command will create a new file called file_name using the cat command:
    $ cat > file_name
    line 1
    line 2
    line 3
    < Cntrl + D will save the file    >

But this is very rarely used, as many powerful editors are already existing, such as vi or gedit.

  - The following command will print Hello World on the console. The echo command is very useful for Shell script writers:
    $ echo "Hello World"

  - The following command will copy the string Hello World to the hello.c file:
    $ echo "Hello World" > hello.c
The command `echo` with `>` overwrites the content of the file. If content already exists in the file, it will be deleted and new content will be added in the file. In a situation, when we need to append the text to the file, then we can use the `echo` command as follows:

```
$ echo "Hello World" >> hello.c  will append the text
```

The following command will display the content of the file on screen:

```
$ cat hello.c
```

## Working with permissions

The following are the types of permissions:

- **Read permission**: The user can read or check the content of the file
- **Write permission**: The user can edit or modify the file
- **Execute permission**: The user can execute the file

## Changing file permissions

The following are the commands for changing the file permissions:

To check the file permission, give the following command:

```
$ ll file_name
```

The details of file permissions are as seen in the following image:
In the preceding diagram, as we can see, permissions are grouped in owner-user and group and other users' permissions. Permissions are of three types such as read, write, and execute permissions. As per the requirement, we may need to change permissions of the various files.

Command chmod
We can change the file or directory permissions by the following two ways:

Technique one – the symbolic method
The following command will add the read/write and execute permissions to the file wherein, u is for user, g is for group, and o is for others:

```bash
$ chmod ugo+rwx file_name
```
Alternatively, you can use the following command:

```bash
$ chmod +rwx file_name
```

Technique two – the numeric method
The following command will change the file permissions using the octal technique:

```bash
$ chmod +rwx file_name
```

The file permission 777 can be understood as 111 111 111, which corresponds to the rwx.rwx.rwx permissions.

Setting umask
We will see how Linux decides the default permissions of the newly created file or folder:

```bash
$ umask
0002
```

The meaning of the preceding output is that, if we create a new directory, then from the permissions of +rwx, the permission 0002 will be subtracted. This means that for a newly created directory, the permissions will be 775 or rwx rwx r-x. For a newly created file, the file permissions will be rw- rw- r-x. By default, for any newly created text file, the execute bit will never be set. Therefore, the newly created text file and directory will have different permissions even though the umask is same.
Setuid
Another very interesting functionality is the setuid feature. If the setuid bit is set for a script, then the script will always run with the owner's privileges irrespective of which user is running the script. If the administrator wants to run script written by him by other users, then he can set this bit.

Consider either of the following situations:
$ chmod u+s file_name
$ chmod 4777 file

The file permissions after any of the preceding two commands will be drwsrwxrwx.

Setgid
Similar to setuid, the setgid functionality gives the user the ability to run scripts with group owner's privileges, even if it is executed by any other user.

$ chmod g+s filename

Alternatively, you can use the following command:
$ chmod 2777 filename

File permissions after any of the preceding two commands will be drwxrwsrwt.

Sticky bit
Sticky bit is a very interesting functionality. Let's say, in the administration department there are 10 users. If one folder has been set with sticky bit, then all other users can copy files to that folder. All users can read the files, but only the owner of the respective file can edit or delete the file. Other user can only read but not edit or modify the files if the sticky bit is set.

$ chmod +t filename

Alternatively, you can use the following command:
$ chmod 1777

File permissions after any of the preceding two commands will be drwxrwxrwt.
Summary
In this chapter, you learned different ways to write and run Shell scripts. You also learned ways to handle files and directories as well as work with permissions.

In the next chapter, you will learn about process management, job control, and automation.
Where to buy this book

You can buy Learning Linux Shell Scripting from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.