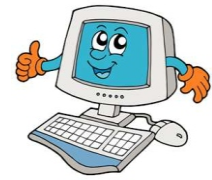


7

Vim Editor and Basic Scripting



In the previous chapter, we have discussed commands that can be used to work with Ubuntu Linux. The commands were executed one at a time. Though we could execute multiple commands by using the pipes, the process would become tedious as number of commands increases. A better way of executing multiple commands at one go is to type the sequence of commands in a text file. Then give this file to Linux shell for execution. The Linux shell will execute all the commands available within the text file in the specified sequence. This text file is known as shell script. A shell script can be defined as series of commands written in a plain text file. The shell scripts are commonly used by the users to perform routine individual tasks and system administration. In this chapter we will look at an editor that assists us in writing the shell script along with some sample shell scripts.

Working with Vim Editor

We have learned how to create a file using the cat command. The cat command although it allows us to create a file is not a good option to use when creating a shell script. We need a good text editor to perform such operations. Text editors like nano, pico, vi or Vim, ed and others are generally best suited for creating text file. Gedit is a graphical editor available with GNOME desktop environment. Kwrite is a graphical editor available with KDE desktop environment. We will use the Vim editor which is a visual display editor to write the shell script. This editor is available with almost all Unix and Linux flavors.

The Vim (Vi Improved) is a text editor written by Bram Moolenaar and first released publicly in 1991. It is an enhanced version of the vi editor distributed with most UNIX systems. Vim is a highly configurable text editor built to enable efficient text editing. The Vim editor can be used from both a command line interface and as a standalone application in a graphical user interface.

To work with the Vim editor we will have to initiate it first. Open a new Terminal Window. We can open the Vim editor using two ways, first type *vi* at the prompt and press Enter key or type *vi* followed by a file name and press Enter key. Figure 7.1 shows the Vim editor interface when we don't specify a file name.

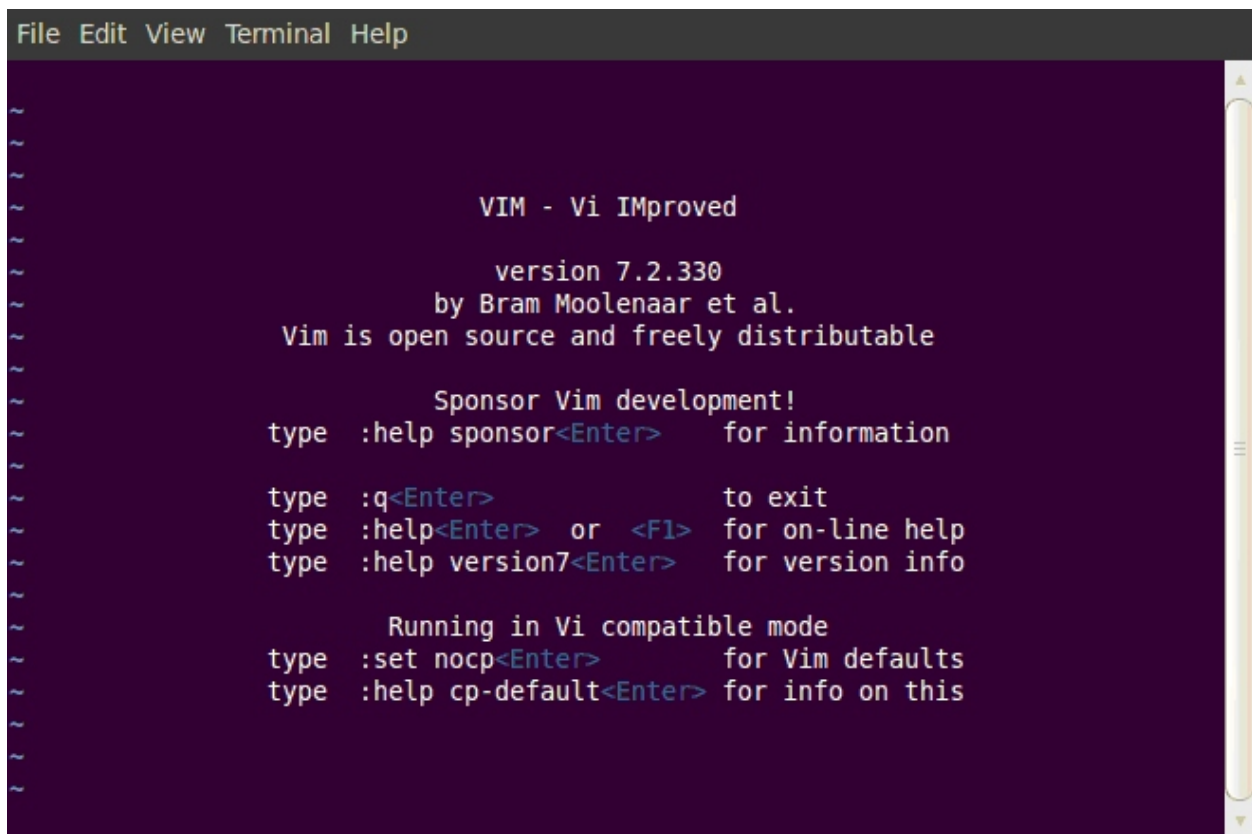


Figure 7.1 : Vim editor interface

It is a good option to start the Vim editor by specifying a file name. Type the command given below:

\$vi first_vim_file

and press Enter key, the command when executed will open the editor as shown in figure 7.2.



Figure 7.2 : Creating file using Vim editor

Observe that the screen in figure 7.2 is filled with tildes (~) on the left side of the screen. The tilde (~) symbol indicates that the lines are yet to be used by the editor. Notice that the cursor appears in the top left corner of the screen and some text is visible on the last line. The last line is known as command line and it displays the name of the file along with the information about the total number of lines and columns within the file.

Vim Modes

The Vim editor functions in three different modes namely, a command mode, an insert mode and a last line mode.

The command mode

When we first start editing a file using the Vim editor, the editor will be opened in a command mode. We can issue many commands that allow us to insert, append, delete text, or search and navigate within our file. Note that when using command mode we can't insert text immediately. We first need to issue an insert (i), append (a), or open (o) command to insert the text in the file.

An extension to the command mode is a visual mode. Visual mode is a flexible and easy way to select a piece of text from the file. It is the only way to select a block of text that needs to be modified. Table 7.1 shows us the characters that assist us in the visual mode.

Command	Usage
v	Switch to the visual mode (allows us to manipulate characters)
V	Switch to the visual mode (allows us to manipulate lines)
CTRL + v	Switch to the block-visual mode (allows us to manipulate rectangular blocks of text)

Table 7.1 : Characters that are used in visual mode

The insert mode

When we issue an insert, append, or open command, we will be in the insert mode. The current text editors show us the current mode of operation. Once in an insert mode we can type text into our file or navigate within the file.

We can toggle between the command mode and the insert mode by pressing the ESC key. This operation will be performed often when using the Vim editor. Table 7.2 shows us the characters that assist us in the insert mode.

Command	Usage
a	To insert text after the current cursor position.
i	To insert text before the current cursor position.
A	To append text at the end of the current line.
I	To insert text from the beginning of a line.
O	To insert in a new line above the current cursor position.
o	To insert in a new line below the current cursor position.

Table 7.2: Characters that are used in insert mode

The last line mode

The last line mode normally is used to perform operations like quitting the Vim session or saving a file. To go to the last line mode we first need to be in the command mode. From command mode we can go to last line mode by pressing the colon (:) key. After pressing this key, we will see a colon character at the beginning of the last line of our editor window with a cursor blinking near it. This indicates that the editor is ready for accepting a “last line command”.

It is possible to toggle back to the command mode from the last line mode by pressing the ESC key twice or by pressing the [Backspace] key until the initial “:” character is gone along with all the characters that we had typed or by simply pressing the ENTER key.

Creating a file in Vim

Let us now learn how to create a simple text file using the Vim editor. Execute the command given below to open the editor interface.

\$vi about_Gandhiji

To enter text within the file the editor needs to be in the insert mode. By default, the editor will start in the command mode. As seen in table 7.2 there are several commands that put the editor into the insert mode. The most commonly used commands to get into insert mode are ‘a’ and ‘i’.

Press ‘i’ and the editor will now be in the insert mode. Now type the contents as given in the box below.

Mahatma Gandhi was born on 2nd October 1869 in Porbandar, Gujarat.
His name was Mohandas Karamchand Gandhi.

Saving the file

Once we have written the contents shown we need to save this file. To save the file we need to switch to the last line mode from the insert mode. Press the ESC key and type colon (:), you will notice that colon is displayed in the bottom of the screen. Type *wq* (write and quit) as shown in figure 7.3 and press the Enter key. The Vim editor will now be closed and we will be able to see the shell prompt. To see the contents of the file we can use the cat command.

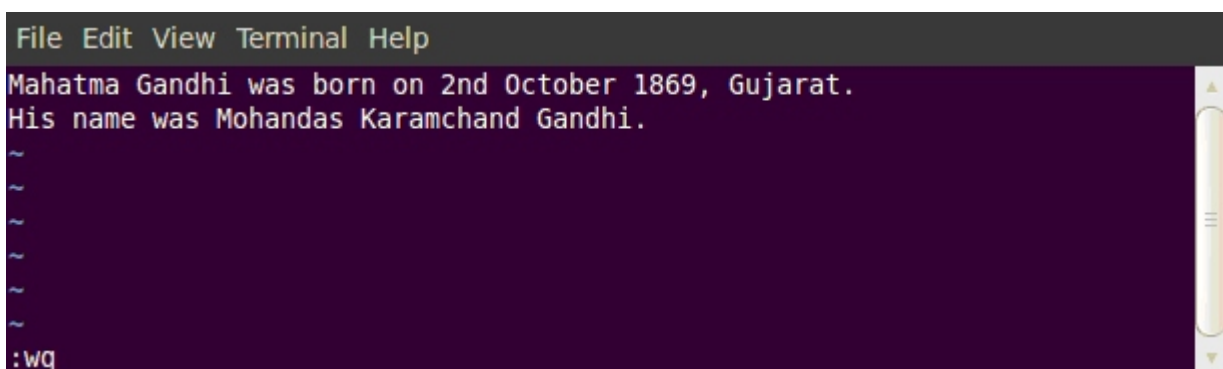


Figure 7.3: Saving a file and quitting the editor

There are several other commands available to save a file depending on the current status and usage. Table 7.3 shows commands that can be used in the last line mode along with their usage.

Command	Usage
:w	To save file and remain in editing mode
:wq	To save file and quit editing mode
x	To save file and quit editing mode (same as above)
:q	To quit editing mode when no changes are made
:q!	To quit editing mode without saving changes made in the file
:saveas FILENAME	To save existing file with new name and continue editing it under the new file name.

Table 7.3 : The last line mode commands to save the file

Note that if we open the Vim editor without typing file name initially the text will be directly stored in the system buffer (main memory). To transfer the contents from the buffer to a hard disk we need to type the file name along with the *wq* command.

Moving around in the document

You must have observed when creating the file about_Gandhiji the arrow keys (by any chance if you used them) were not working as per our expectations. Normally the arrow keys are used to move the cursor in up, down, left and right directions. In insert mode we cannot do anything except for typing the text. When using the Vim editor we need to use some special keystrokes to move within the document after going to the command mode. Once in command mode we can also use the arrow keys to move within the document. Table 7.4 lists the keystrokes that are used to navigate within the documents.

Command	Usage
h	Moves cursor left
l	Moves cursor right
j	Moves cursor down
k	Moves cursor up
Spacebar	Move cursor right one space

-/+ Keys	Move cursor down/up in first column
CTRL + d	Scroll down one half of a page
CTRL + u	Scroll up one half of a page
CTRL + f	Scroll forward one page
CTRL + b	Scroll back one page
M	Move the cursor middle of the page
H	Move the cursor to top of the page
L	Move the cursor to bottom of page
\$	Move the cursor to end of line
)	Move the cursor to beginning of next sentence
(Move the cursor to beginning of current sentence
G	Move the cursor to end of file
W	Move the cursor one word at a time
Nw	Move the cursor ahead by N number of words
B	Move the cursor back a word at a time.
b	Move the cursor back a word at a time.
Nb	Move the cursor back by N number of words
e	Move the cursor to end of word
gg	Move to first line of file
0	Move to the beginning of the line.

Table 7.4 : Keys to navigate in file

Try and work on these keystrokes to become familiar with them.

Editing the Document

Editing the document is one of the most common operations that a user would perform once the document is created. It is possible to insert or delete any data at a specific position as per our

needs. We can also replace contents or change the case of individual characters if required. A user needs to toggle between the command and the insert mode when we edit a document. The characters in table 7.2 showed us how to initiate different insert options. Let us try to add the contents given in the box below to the file `about_Gandhiji`.

His mother, Putlibai, was a very religious lady and used to tell him stories from the scriptures and mythology.

Little Gandhi grew up to be an honest and a decent student. At the age of 13 he was married to Kasturba.

To edit the document we need to again open it using the `vi` command, hence execute the command `vi about_Gandhiji` again. You will observe that the blinking cursor is visible on the first character at top left. In normal cases we would have used the `'I'` option to enter the insert mode, but we need to append the contents at the end.

Type `G` and you will see that the cursor gets positioned at last line. The position of the cursor will depend upon how the file was initially saved. Typing `G` may place the cursor in a new line after the last line (case when `Enter` key was pressed before saving the file) or the cursor will be placed on the first character of the last line (case when `Enter` key was not pressed before saving the file).

If the cursor is placed at new line we press `ESC` and then `'I'` and start typing the contents. In case we are at the first character of the last line, then press `'o'`. This action will take you to a new line. Now start typing the contents, in case of any errors in typing we may use the `Backspace` key and go back one cursor position at a time to correct the error. Once the editing is over press `ESC` key and type `:wq` to go the last line mode, save the file and end the Vim session. Figure 7.4 shows the look of the editor after the new contents have been added.

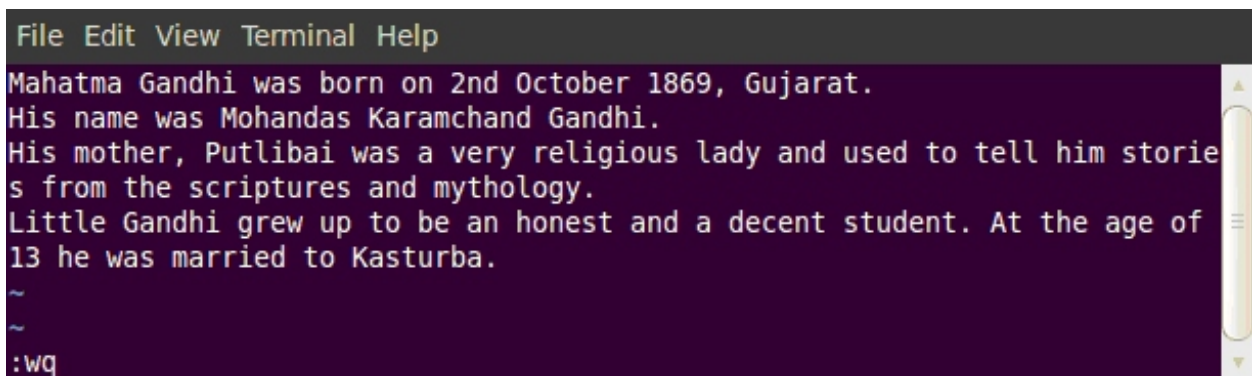


Figure 7.4 : Appending data in existing file

We may use the commands shown in table 7.5 to perform various editing operations on any document.

Command	Usage
u	Undo last change
U	Undo all changes to entire line
dd	Delete single line
Ndd	Delete N number of lines
D	Delete contents of line after cursor
C	Delete contents of line after cursor and insert new text. Press ESC key to end the insertion
dw	Delete one word
Ndw	Delete N number of words
cw	Change word
x	Delete the character under the cursor.
X	Delete the character before the cursor (Backspace).
r	Replace single character
R	Overwrite characters from cursor onward
s	Substitute one character under cursor and continue to insert
S	Substitute entire line and begin to insert at beginning of line
~	Change case of individual character
.	Repeat last command action.

Table 7.5 : Commands to perform editing

In addition to the commands given in table 7.5 the Vim editor also allows us to copy text from our file into temporary buffers and vice-versa. Each buffer acts like temporary memory, more commonly known as “clipboard”. Table 7.6 lists some of the commands that are used for capturing and pasting data.

Command	Usage
yy	Copy single line (defined by current cursor position) into the buffer
Nyy	Copy N lines from current cursor position into the buffer
p	Place (paste) contents of buffer after current line defined by current cursor position.

Table 7.6 : Commands to capture and paste

Searching and replacing text

Searching for content and replacing it is another common operation performed by users. The Vim editor allows us to use special commands to search text or a regular expression within the file. We can also substitute a word in place of another using command. Table 7.7 lists various commands that are useful for performing search or replace operation within a file.

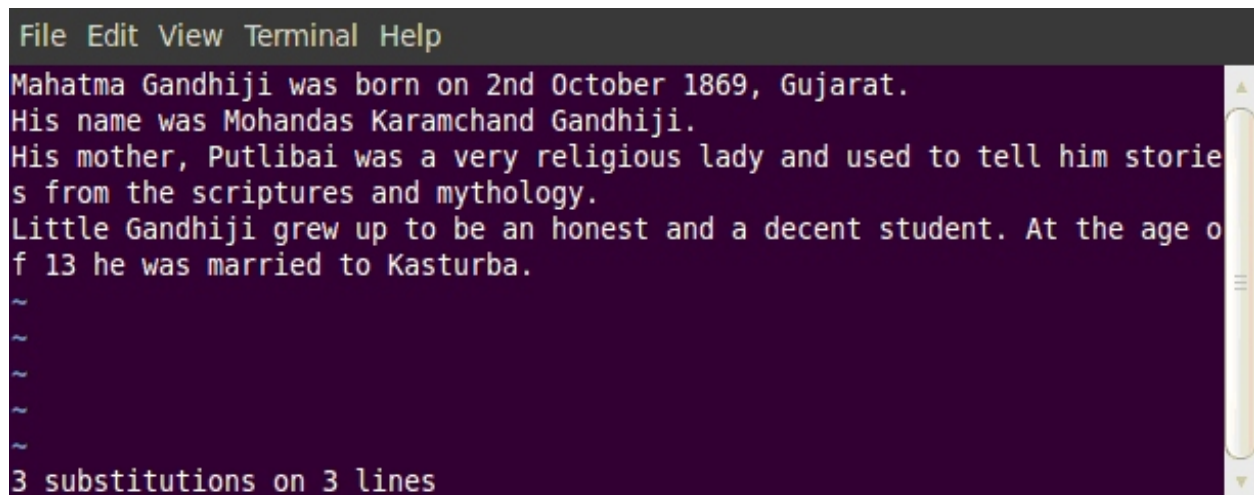
Command	Usage
/	Search for text in a forward direction
?	Search for text in a backwards direction
n	Search again in the same direction
SHIFT + n	Search again in the opposite direction
f	Press <i>f</i> and type the character to be searched. The cursor will move to that character on the current line.
SHIFT + f	Similar to <i>f</i> but searches in backward direction
t	Similar to <i>f</i> except that it moves the cursor one character before the specified character.
SHIFT + t	Similar to <i>t</i> but searches in backward direction
:s/old_string/new_string	Substitutes <i>new_string</i> for the first occurrence of the <i>old_string</i> in the current line
:s/old_string/new_string/g	Substitutes <i>new_string</i> for all the occurrences of the <i>old_string</i> in the current line
:%s/old_string/new_string/g	Substitutes <i>new_string</i> for all the occurrences of the <i>old_string</i> in the whole file
:%s/old_string/new_string/gc	Substitutes <i>new_string</i> for all the occurrences of the <i>old_string</i> in the file, but asks for confirmation before substituting the <i>new_string</i>

Table 7.7 : Commands to perform search and replace operation

Let us try to use some of these commands in the file `about_Gandhiji`. Assume that we wanted to replace all the occurrence of the word “Gandhi” with word “Gandhiji”. To perform this operation we need to first open the file using the command `vi about_Gandhiji`, then go to the last line mode by pressing the `ESC` key and execute the command given below:

:%s/Gandhi/Gandhiji/g

Here %s indicates we are trying to replace a string. The term “Gandhi” refers to the old string that is to be replaced while the term “Gandhiji” refers to the new string. The option “g” indicates that we have to substitute all the occurrences of the term “Gandhi” with the term “Gandhiji” in the whole file. The output of this command is shown in figure 7.5. Observe that it also shows how many occurrences have been replaced. We need to save the file if we need this change to be reflected in it. If we quit without saving then the changes will not be reflected in the file.



```
File Edit View Terminal Help
Mahatma Gandhiji was born on 2nd October 1869, Gujarat.
His name was Mohandas Karamchand Gandhiji.
His mother, Putlibai was a very religious lady and used to tell him stories
from the scriptures and mythology.
Little Gandhiji grew up to be an honest and a decent student. At the age of
13 he was married to Kasturba.
~
~
~
~
~
3 substitutions on 3 lines
```

Figure 7.5 : Search and replace operation

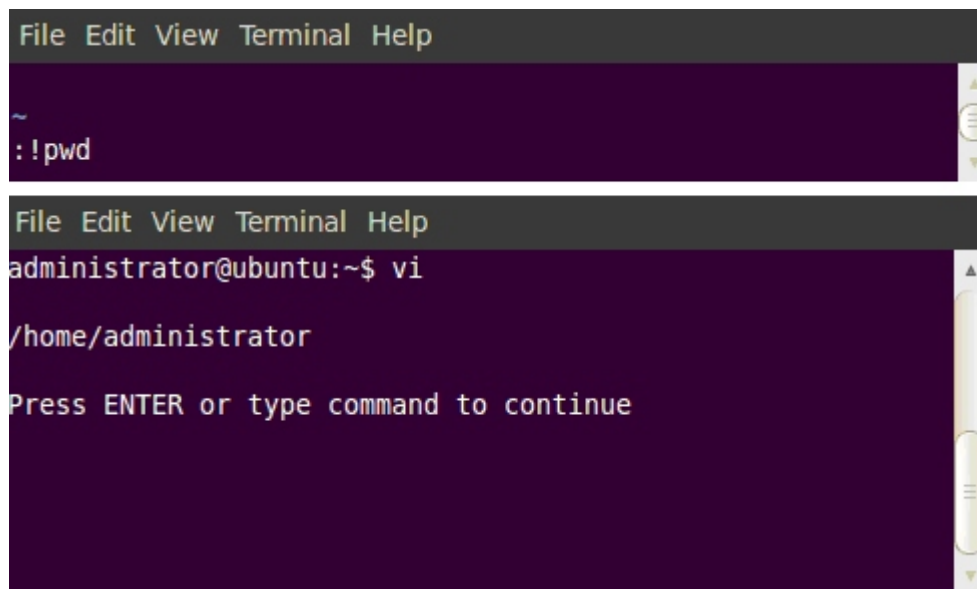
Executing Linux commands through Vim

It is also possible to execute the Linux commands from within the Vim editor. To execute any Linux command we need to type the exclamation (!) symbol before the command.

For example if we want to see the current working directory then perform the following steps :

- Open the Vim editor by typing vi on the command prompt.
- Go to the last line mode by pressing ‘ESC :’
- Type !pwd
- Press Enter key.

We will be able to see the current working directory. Figure 7.6 shows the operation and its output.



```
File Edit View Terminal Help
~
:!pwd

File Edit View Terminal Help
administrator@ubuntu:~$ vi
/home/administrator
Press ENTER or type command to continue
```

Figure 7.6 : Linux command in Vim editor

Similarly if we want to add the current date in a new line from the current cursor position within a file we may execute the following steps:

- Open the file using Vim editor.
- Go to the last line mode by pressing ‘ESC :’
- Type r !date
- Press Enter key.

The r option allows us to insert data in the file or buffer.

Shell Script

We saw how a command can be executed from a command prompt as well as using the Vim editor. Both these mechanisms allow us to execute commands one at a time. A shell script allows us to execute more than one command at one go in a better way. Thus instead of spending time in typing the commands on the prompt every time a task needs to be performed, we can create a shell script and execute the given set of commands by typing a single line command.

Shell script can be defined as “Set of commands written in plain text file that performs a designated task in a controlled order.” Shell scripts can be designed to be interactive; such a script may accept input from the user and perform different tasks based on the input provided.

Creating and Executing a Shell Script

We can create the shell script using any text editor. As we have learned to work with the Vim editor we will create scripts in it. Let us create a small shell script that welcomes a user. Type the script shown in the box below in Vim editor and save it with the name *script1.sh*. The extension

“sh” is basically used to specify that the file is a shell script. Note that it is not mandatory to create a file with an extension “sh”, a file without extension can also be used as a shell script. But it is always a good practice to give extension as it will help us differentiate between normal files and shell script files.

```
#Script 1: Script to welcome the user who has logged into the system
```

```
clear
```

```
echo Hello
```

```
who am i
```

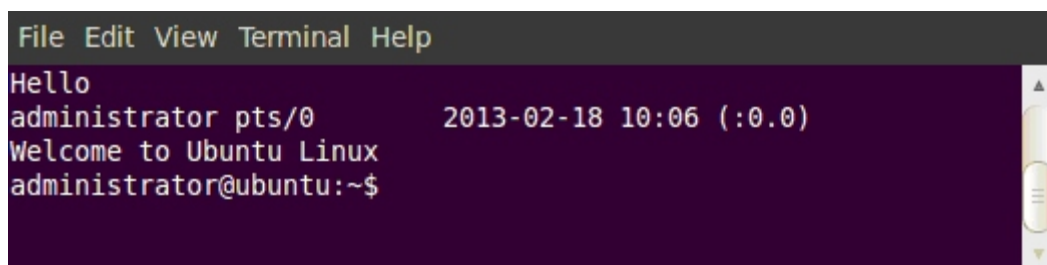
```
echo Welcome to Ubuntu Linux
```

Observe the first line in the script; it begins with symbol ‘#’. Any line preceded by the ‘#’ symbol is considered as a comment. The comments when part of the script are not executed; they are messages that help user understand the usage or meaning of the script. The second line is a command that clears the screen contents before giving the output of the script. The third line displays a message “Hello”, the fourth line executes a command “*who am i*” that gives us the name and some additional details of the user currently logged into the system. The last line again displays a message “Welcome to Ubuntu Linux”.

To execute the script we need to use *sh* or *bash* command. If the script is stored in current directory then type the command mentioned below:

\$bash script1.sh or **\$sh script1.sh**

Sometimes we might come across issues related to file privileges. For a script to be executed it needs to have execute permission explicitly set. By any chance if such a problem occurs we will have to use the *chmod* command to set the desired privileges. For Example issuing the command ***chmod +x script1.sh*** will make the file executable. If everything goes fine we will get the output similar to the one shown in figure 7.7.



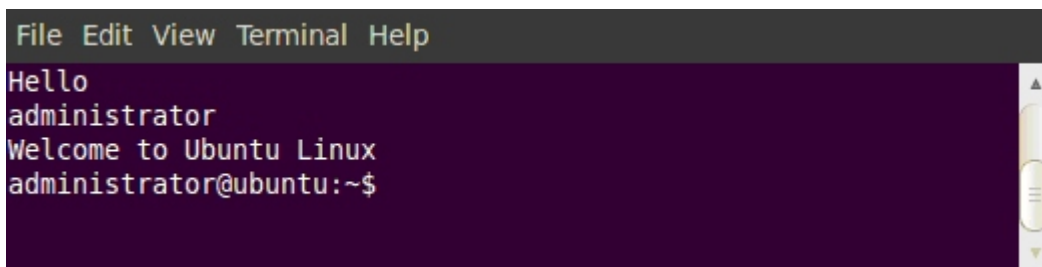
```
File Edit View Terminal Help
Hello
administrator pts/0          2013-02-18 10:06 (:0.0)
Welcome to Ubuntu Linux
administrator@ubuntu:~$
```

Figure 7.7 : Output of Script 1

Observe that in figure 7.7 we are getting some additional contents along with the user name. Let us use our knowledge of filters and try to remove the additional contents. The modified script is shown below:

```
#Script 2: Modified script to welcome the user who has logged into the system  
clear  
echo Hello  
echo ``who am i | cut -d " " -f 1``  
echo Welcome to Ubuntu Linux
```

Observe that we have used the filter *cut* along with the command *who am i*. We further have joined the two commands using the pipe. To make sure that the contents within the double quotes after the echo command are not treated as message we enclose them in back quotes (‘ ‘). The back quotes are printed on the key with ~ sign on the keyboard. Type the modified script using Vim editor and save it as *script2.sh*. Execute this script and you will observe that we are only able to see the contents that we want. Figure 7.8 shows the output of the modified script.



```
File Edit View Terminal Help  
Hello  
administrator  
Welcome to Ubuntu Linux  
administrator@ubuntu:~$
```

Figure 7.8 : Output of Script 2

Let us further modify the script to display current date and time. The script is given in the box below .

```
#Script 3: Script to welcome the user and display login date and time  
clear  
echo Hello  
echo ``who am i | cut -d " " -f 1``  
echo Welcome to Ubuntu Linux  
echo The current date and time is  
date
```

Create a file named *script3.sh* and type the contents of the script 3 in it. Execute it and observe the output. Once we are comfortable using the shell scripts we will find them very helpful in performing repetitive tasks.

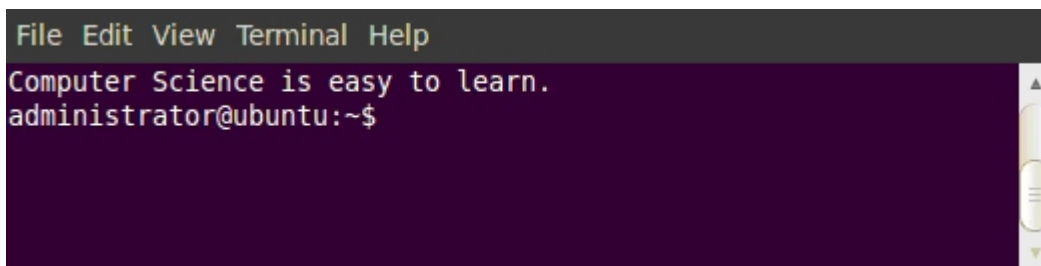
Shell Script Variables

The process of shell scripting is almost similar to the process of writing programs in a higher level language. One of the most common features of higher level programming is provision of variables.

As the name indicates variables are entities wherein we can store or edit a value. The value stored in the variable can also be reused or changed as per users need. Shell script variables like any other programming language variables are integral part of shell scripting. A variable when used in a shell script allows us to assign a value to it or accept its value from the user. We can also display the value assigned to this on the screen using echo command. Let us write a small script that shows the use of variable. The code of the script is given in the box below:

```
# Script 4: Shell script to show use of variables  
clear  
subject="Computer Science"  
echo $subject is easy to learn.
```

Type the contents of the script 4 in a file named *script4.sh*. Let us try to understand the script just saved. Similar to all the other scripts the first line indicates a comment. The second line clears the screen of any previous contents. In the third statement we have defined a variable called *subject* and assigned it value “Computer Science” using a simple assignment operator. As the string contains white space in between two words we need to enclose it within double quotes. The fourth line displays the message on the screen. The ‘\$’ symbol preceding the variable name *subject* instructs the shell to extract the value stored or assigned in the variable. Figure 7.9 displays the output of the script when it is executed.



```
File Edit View Terminal Help  
Computer Science is easy to learn.  
administrator@ubuntu:~$
```

Figure 7.9 : Output of Script 4

A user needs to take care that there should not be any space on either side of the equal to (=) symbol at the time of assigning a value. If due to some reasons a space occurs at either side, the shell will interpret the string after the space as a command. This may give unexpected outputs. The statement *subject="Computer Science"* first creates a variable named *subject* and then assigns it a value “Computer Science”. If we reuse this variable again, the old value stored in it will be overwritten. Let us try to understand the last statement by writing a simple script given in the box:

```
# Script 5: Shell script to show use of variables  
clear  
subject="Computer Science"  
echo $subject is easy to learn.  
subject="Economics"  
echo $subject is easy to learn.
```

Save the script as `script5.sh` and observe its output after executing it.

We need to follow the rules mentioned below when defining a variable in a shell script.

- A variable name can consist of alphabets, digits or an underscore (`_`).
- No special character other than underscore allowed as part of variable name.
- The first character of a variable name must either be an alphabet or an underscore.

Note :

If the shell is unable to understand a word as a variable it will interpret it as a Linux command.

User Interaction and Shell Script

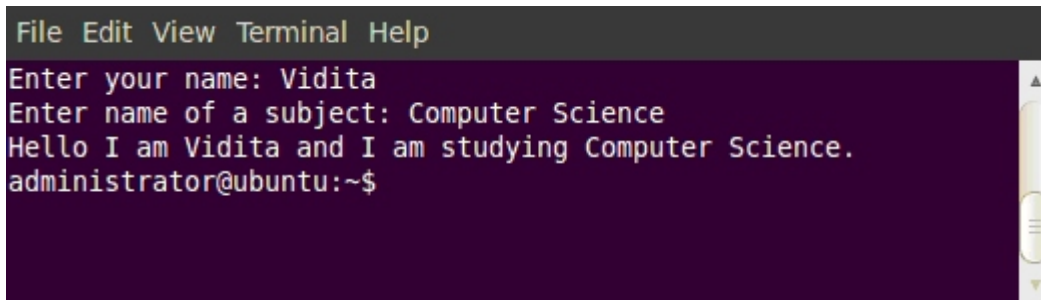
You must have observed in the previous example that we have assigned a value to the variable within the script itself. A variable when used in such a manner does not have much significance. A variable is generally used so that it can be further used in some operations with change in its value if needed. This property can be achieved only if we are able to accept the value of the variable from the user. It is possible to assign value to variables defined in the shell script using the `read` command. The `read` command expects the user to key in the data on the standard input device, it then takes all the contents that we type and stores it in the variable name supplied to it as an argument. Let us rewrite a script 5 to accept the subject names from the user. Save the code in the box with a file name `script6.sh`.

```
# Script 6: Shell script to accept value of variable from user  
clear  
echo -n "Enter your name: "  
read name  
echo -n "Enter name of a subject: "  
read subject  
echo Hello I am $name and I am studying $subject.
```

When we execute the script, the first executable statement will first clear the screen contents. Then it will display a message “Enter your name:”, the next statement waits for the user to enter its name. Pressing of Enter key indicates end of entry, so be careful that it is pressed only after the name has been typed. In case we press Enter key without typing anything the script will assign NULL value to the variable and move to next command. The third and fourth

statement also does the same task of displaying message and waiting for the user to key in a value for subject. The last statement displays the values of both the variables along with an appropriate message.

Observe that we have used `-n` option along with the `echo` command. This option instructs the `echo` command not to print a new line after the message is displayed. The `echo` command by default inserts a new line after displaying the message passed to it as an argument. Figure 7.10 shows output of the script.



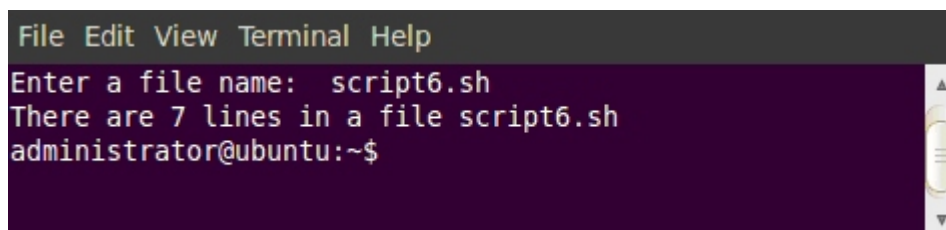
```
File Edit View Terminal Help
Enter your name: Vidita
Enter name of a subject: Computer Science
Hello I am Vidita and I am studying Computer Science.
administrator@ubuntu:~$
```

Figure 7.10 : Output of Script 6

As we are accepting the values of both the variables from the user, the output will change according to what user enters every time we execute the script. Let us write one more script that accepts a file name from the user and display the total number of lines in that file. The code of the script is shown herewith save it as *script7.sh*.

```
#Script 7: Shell script to display total number of lines in a file
clear
echo -n "Enter a file name: "
read fname
echo "There are `cat $fname | wc -l ` lines in a file $fname"
```

Figure 7.11 shows the output of the script when it is executed.



```
File Edit View Terminal Help
Enter a file name: script6.sh
There are 7 lines in a file script6.sh
administrator@ubuntu:~$
```

Figure 7.11 : Output of Script 7

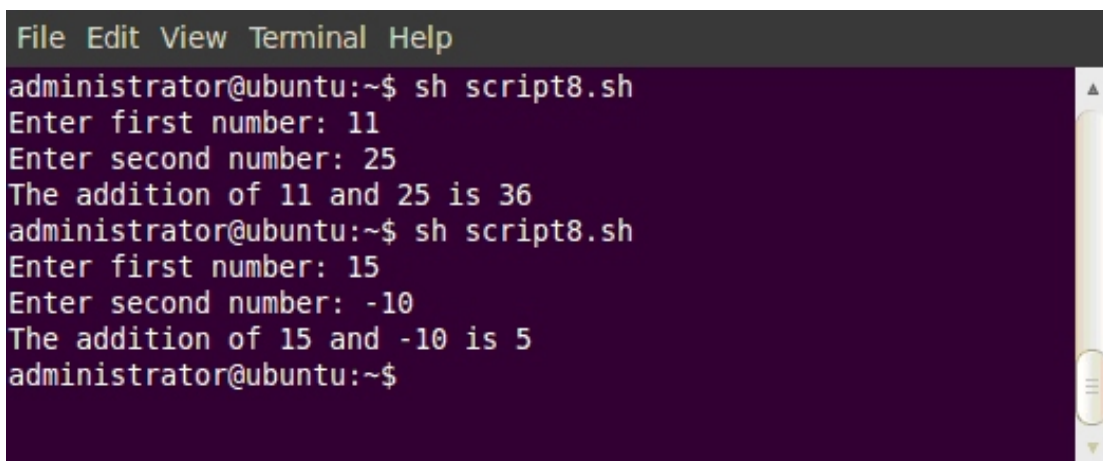
Shell Arithmetic

In the previous section we saw how to define a variable, assign value to it and how to retrieve the stored value. The value assigned so far were all strings (set of alphabet, digit or special characters). We can also assign only numeric values to the variable and perform operation with them. Let us write a shell script that accept two numbers and perform addition of these numbers. The code of the script is shown in the box below:

```
# Script 8: Script to add two numbers
echo -n "Enter first number: "
read num1
echo -n "Enter second number: "
read num2
sum=`expr $num1 + $num2`
echo "The addition of $num1 and $num2 is $sum"
```

Type the script and save it as *script8.sh*. In this script the term *expr* means expression, the contents written after *expr* are assumed to be operands and operators of an expression. Note that there should be one space between operator (+) and operands (\$num1, \$num2). Additionally, there should be no space before or after the assignment operator (=).

Figure 7.12 shows the different output of the script when it is executed twice. We are able to see the different outputs in one screen as we have not used the clear command in this script.



```
File Edit View Terminal Help
administrator@ubuntu:~$ sh script8.sh
Enter first number: 11
Enter second number: 25
The addition of 11 and 25 is 36
administrator@ubuntu:~$ sh script8.sh
Enter first number: 15
Enter second number: -10
The addition of 15 and -10 is 5
administrator@ubuntu:~$
```

Figure 7.12 : Output of Script 8

We can also perform subtraction, multiplication, division and modular division by using the -, *, / and % operators respectively. The expressions are evaluated as per the general norms of mathematics.

In case of tie between operators of same priority, preference is given to the operator which occurs first. To force one operation to be performed earlier than the other, we can enclose the operation in parenthesis.

For example, in the expression `$num1 * \($num2 + $num3 \) / $num4`, the operation `$num2 + $num3` will be evaluated first as it is enclosed within parentheses. Observe that we have preceded the `*` symbol as well as the left and right parentheses by a back slash character (`\`).

Note :

We need to prefix the multiplication (`*`) symbol with backslash (`\`) character when finding product of two numbers. Otherwise the shell will treat the (`*`) symbol as a wildcard character.

Let us create one more script that will accept a birth year from the user and display users current age in years. The code of the script is shown in the box below.

```
# Script 9: Script to calculate age of user in years
echo -n "Enter year of your birth: "
read byear
cyear=`date | tr -s ' ' | cut -d " " -f 6`
age=`expr $cyear - $byear`
echo "You are $age years old as of today."
```

Observe that in script 9 to make sure that all the multiple spaces in output of date command are squeezed to single space we have used the `tr` command with `-s` option. As we want only the year value which appears in the 6th column of the output when the date command is executed we have used the `cut` filter. Figure 7.13 shows the different output of the script when it is executed.

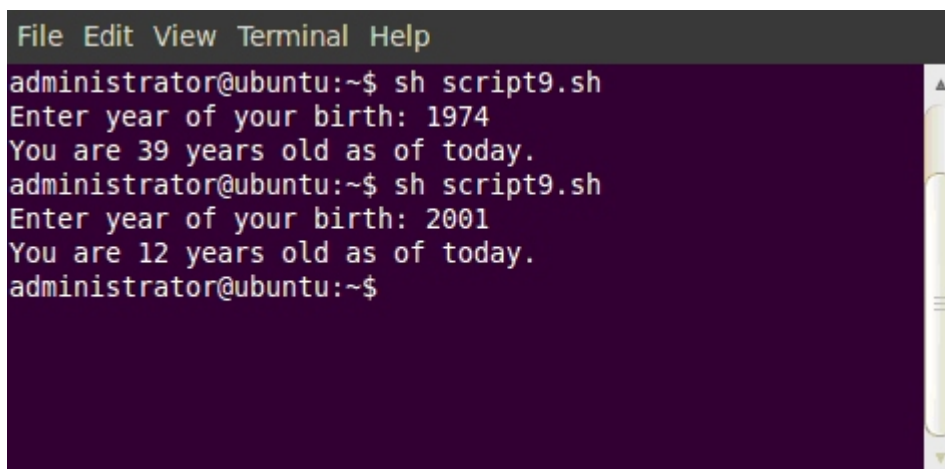


Figure 7.13 : Output of Script 9

Use of Shell Scripts

The shell script is a very powerful tool of Linux. It has almost all the capabilities of any higher level programming language. Once familiar with it we can perform and automate many tasks easily. Generally the repetitive task should be done using the shell scripts. Some uses of shell scripts are mentioned below:

- Create a new command using multiple set of commands.
- For automating many aspects of computer maintenance, for example create 1000 user accounts; delete all size 0 files, installation of new software etc.
- Data backup

As such there are no limitations on its usage; a user may use it for any purpose that he wants.

Summary

In this chapter we learned how to use the Vim text editor provided with Ubuntu Linux. The text editor is very powerful; it allows creating, updating and deleting contents of a file. Further we can search for required contents within a file. We also learned how to create a simple text file as well as a shell script using this editor. The shell script is a text file that contains sequence of commands that can be executed by simply using the shell scripts file name. Finally we saw how a shell script can be made equivalent to a high level program by making use of a variable and then using it in an expression.

EXERCISE

1. Explain different modes available in Vim editor.
2. List and explain the working of different save options of Vim editor.
3. Explain the difference between using dd and 2dd command.
4. What is a shell script?
6. List at least three uses of shell script.
7. **Choose the most appropriate option from those given below :**
 - (1) In how many modes Vim editor works?
 - (a) One
 - (b) Two
 - (c) Three
 - (d) Four
 - (2) Which of the following statement is true for Gedit?
 - (a) It is a Command line editor.
 - (b) It is a Graphical editor.
 - (c) It is not an editor.
 - (d) It is available with KDE Desktop environment.

- (3) :wq in Vim editor is used for which of the following activities?
- (a) To save file and remain in editing mode
 - (b) To save file and quit editing mode
 - (c) To quite editing mode without saving changes made in the file
 - (d) All of the above
- (4) Which of following keys is not used to go into insert mode of the Vim editor?
- (a) o
 - (b) i
 - (c) a
 - (d) cw
- (5) Which of the following keys are used to delete a line?
- (a) ce
 - (b) ge
 - (c) dd
 - (d) d\$
- (6) Which of the following statements is used to search for phrase in the file?
- (a) :set is
 - (b) :help cmd
 - (c) :!cmd
 - (d) /phrase<ENTER>
- (7) Which of the following syntax is used to substitute all occurrences of phrase1 with phrase2 in the entire file without asking for user confirmation?
- (a) :%s/phrase1/phrase2/g
 - (b) :%s/phrase1/phrase2/gc
 - (c) :s/phrase1/phrase2/g
 - (d) :s/phrase1/phrase2/gc
- (8) Which of the following character is used for commenting a line in a shell script?
- (a) *
 - (b) %
 - (c) \$
 - (d) #
- (9) Which of the following symbol instructs a shell script to extract the value of a variable?
- (a) *
 - (b) %
 - (c) \$
 - (d) #

Laboratory Exercises

Write a script to perform following operations :

- (a) To display the date and time in the given format:
“Today is February 15, 2013 and current time is 12:10:23”

(b) To display the login details of current user in the following format:

Name of the user:

Login date:

Login time:

(c) To display the date, time, username and current directory.

(d) To accept a string and a filename from the user. Search all occurrences of the string inside a given file.

(e) To accept a file name from the user and count number of lines in it.

(f) To accept two file names from the user and creates a new file containing the contents of both the files provided as input.

To accept two file names from the user and compare them.

