# Laboratory 3: Writing Shell Scripts

## 1. Text Editors in Linux

A **text editor** is a program used to create and edit plain text files. The main purpose of a text editor is to create a file that can be used by another program. For instance, text files can contain Hypertext Markup Language (HTML) for a web browser to display, or source code that a compiler can process.

Another purpose is to create a **shell script** – i.e. a text file containing a sequence of commands.

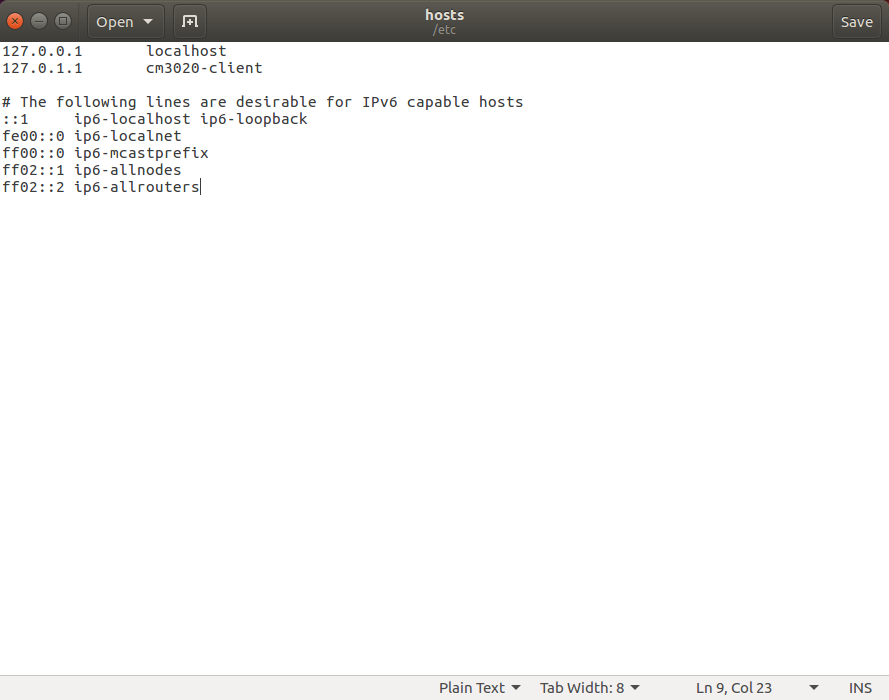
Linux has two types of text editors:

* command-line editors (**Vim** – stands for “vi (visual) improved”)
* GUI editors – e.g. **KWrite** (for KDE) and **Gedit** (for GNOME)

Linux GUI text editors are similar to Notepad in Windows. They have a main window where you can enter and edit text and are easier to manage than their command-line counterparts.

**1.1 gEdit: A GUI Text Editor for GNOME/Unity**

gedit is a GUI text editor, also known as the programmer’s editor for the **Unity/GNOME** (see Figure 2-1).



-Type gedit at a command prompt to open a new empty file in the text editor.

-Add a filename as an argument to open an existing file for editing.

-If the file does not exist, this command will create an empty file with the specified name.

Another way to start **gedit** is to use the **Unity Dash** button at the top left corner and search for gedit. To access the menu on applications in Ubuntu, hover over the application panel (United document – gedit) at the top of the screen and the menu will appear.

gedit has some advanced features, including **syntax highlighting** to display text in different colours and fonts for programming languages, such as C, C++, Java, HTML, and Perl. You can select syntax highlighting from the **View** menu – highlight mode and select which language to highlight for.

### Activity 3.1

### Exploring the gedit Text Editor

**Approx. Time Required:** 20 minutes.

**Objective:** Explore the features of gedit.

**Description:** In this activity, you use gedit to create a simple HTML Web page. You will learn how to display line numbers and enable syntax highlighting to improve readability. You will also learn to add bookmarks and navigate the file. To finalise, you will save the file and open it as an HTML Web page.

1. To enable advanced features, gedit plugins must be installed first. Run the following commands from terminal:
   1. sudo apt-get –f install

If error “Could not get lock /var…” is encountered, try the following commands first:

* + 1. - sudo rm /var/lib/dpkg/lock
    2. sudo dpkg --configure -a
  1. Enter password to run as root.
  2. sudo apt-get install gedit-plugins
  3. Press y to continue installation.

1. Start gedit.
2. To enable bookmarks, go to **Edit** **🡪 Preferences 🡪 Plugins** and tick **Bookmarks** (this menu is located at the very top of the screen).
3. To enable the display of line numbers, go to **Edit** **🡪 Preferences -** **Display Line Numbers**.
4. In the first line, write <!--***your first and last name-->*** and press Enter. On the second line, write <!--***Activity 3.1***--> and press Enter. On the third line, write <!--***today’s date***--> and press Enter two times.

The <!--> symbol in the first three lines indicates a **comment**, so the Web browser does not read these lines.

1. To save the file, click the **Save** button. Name the file act3-1\_Lastname. Exit gedit by clicking **File**, **Quit** from the menu.
2. Recover your file by typing gedit ~/act3-1\_Lastname to test that the file can be opened using the terminal command.
3. You can select syntax highlighting for HTML code by clicking **View**, pointing to **Highlight** **mode** and clicking **HTML**. Pay attention to the changes of the text in the first three lines.
4. On line 5, type the following HTML code, press Enter after each line (twice to create blank lines shown):

<!--your first and last name-->

<!--Activity 3.1-->

<!--today's date-->

<html>

<body>

My test web page

</body>

</html>

1. Click **View** on the menu bar and make sure the **Show Icon Border** option is selected.
2. Put the cursor over *line 5*, and then click **Search 🡪 Toogle Bookmark** from the menu. Do the same thing on *line 10*. Observe the changes.
3. Save the file by clicking **File🡪Save As** from the menu, typing act3-1\_Lastname.html for the filename, and clicking **Save**. Exit gedit by clicking **File🡪Quit**.
4. Open the **Nautilus** file manager by clicking the **Files Manager** on the Launcher. Go to the directory where the html file was saved and open it. What results do you see?

## 

## 2. Writing basic Linux shell scripts

A **shell script** contains a sequence of commands to be executed line by line. Understanding the basics of shell scripting is essential to becoming proficient at system administration because scripts are useful to task automation. When you need to issue several commands to carry out an administrative task, for instance, you can save all the related commands in one executable file.

* Scripts are also used for troubleshooting.
* Some scripts run when the Linux system starts, and you need to know how to manage these scripts if problems occur during the boot process

The shell scripts you will create in this lab are examples of interpreted programs, and the interpreter used is the **BASH shell**.

Creating a shell script is as simple as creating a file and then assigning execute permission for it. By default, files cannot be executed; not even root can execute a file unless the file owner adds execute permission manually. This security feature is in place to ensure that only scripts and programs are executed. As mentioned previously, trying to execute a “regular” file causes an error.

The reason you cannot simply type the script name to run it is that when you run a program in this manner, the kernel looks in the directory paths defined by the PATH variable.

After creating a shell script and assigning permission, you must enter the absolute or relative path to where the file is stored to run the script. (the ***relative*** path starts at your current directory.) For example, to run a script called scr\_1 that is stored in your current directory, you use the command ./scr\_1.

Activity 2-2 walks you through creating and running a shell script.

### Activity 3.2

### Creating a Shell Script

**Time Required:** 10 minutes.

**Objective:** Create and run a shell script.

**Description:** In this activity, you use the vim editor to create a shell script, assign the necessary permission, and then run the script.

1. Open a terminal window, and at the command prompt, type mkdir scripts and press **Enter**. Recall that the mkdir command is used to create directories.
2. To go to the **scripts** directory, type cd scripts and press **Enter**.
3. Type gedit scr\_3-2 and press **Enter** to open a new empty file in the text editor.
4. Enter the following code, pressing **Enter** after each line:

1 #!/bin/bash

2 # This is a comment line and can be

3 # reconfigured by the preceding pound symbol.

4 clear

5 echo Your current directory is

6 pwd

7 echo "You are changing your parent directory, which is..."

8 cd ..

9 pwd

10 echo "You are moving back to the scripts directory"

11 cd ~/scripts

12 pwd

13 echo "Here is a long listing of all your files and directories."

14 ls –l

1. Save and close gedit and then in the terminal type ls –l (while in the scripts directory). Does the user have execute permissions?

By default the user should not have permission as the command ls –l should output the following long format information:

*- r w* ***-*** *r w - r - - 1 administrator administrator*

Recall from “Learning shell scripting” (last week’s lab) that the *long format* *listing* has the following structure:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | **4** | 5 | 6 | 7 | 8 | 9 | 10 |

1. File directory

2-4) Read/Write/Execution rights of file owner.

5-7) Read/Write/Execution rights of file group.

8-10) Read/Write/Execution rights of everyone else.

Notice that the fourth position has a -, which translates into no execution rights for the user.

1. To give the user permission to run the script type chmod u+x scr\_3-2 and press **Enter**. In this case, you used the symbolic method to assign the permission.
2. Run the script by typing ./scr\_3-2 and pressing **Enter**. What do you see?

Script scr\_3-2is used to move between directories. **Line 1** specifies that the script shall be run in the BASH shell, which ensures that all users, regardless of the shell they are using, have the same results. **Lines 2** and **3** are comments, so the interpreter ignores them. The purpose of a comment is to add documentation information for users and anyone else who might modify the script. The clear command **in line 4** clears the screen. The echo command in lines **5, 7, 10** and **13** has a dual purpose: to display the text you type or to display the value of a variable, discussed next. Notice the quotation marks used in these lines – they are needed if you want to display an apostrophe.

**2.1 Variables**

An **environment variable** is a placeholder for data that can change; typically, it gets its value automatically from the OS startup or the shell being used. Linux is a multiuser OS, so more than one user can log in at the same time. For this reason, each user has environment variables with different values to define his or her working environment. For instance, the HOME environment variable stores the absolute pathname to a user’s home directory, so it varies each user. Other environment variables are the same for all users logged in to a machine, such as the HOST environment variable that specifies the computer name.

Table 3.1 describes a few common environment variables. The env command is used to display a list of environment variables and their stored values. To see a particular variable’s stored value, also use the echo command and add a $ symbol before the variable name. For instance, echo $HOME returns the value of the HOME variable.

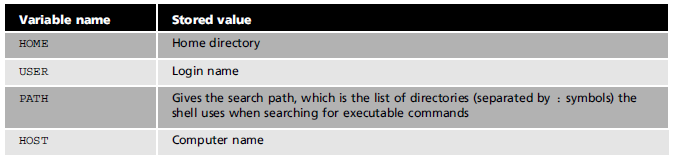


Table 3.1

A **shell variable** is similar to an environment variable, but its value is usually assigned in a shell script. These variables are related to a particular script, not necessarily the global environment, as environment variables are. Variables are also important for specifying how shell scripts run. For instance, you can create a script that greets users by name. There are three ways to store information in a variable:

* direct assignment
* the prompt method
* positional parameters.

**2.2 Direct Assignment**

You use the **direct assignment** method when you want to specify the variable’s value in the command. For example, COLOUR=blue specifies the value of the COLOUR variable.

**Activity 3.3**

### 

### Using the Direct Assignment Method

**Time Required:** 10 minutes.

**Objective:** Use the direct assignment method to store a value in a variable.

**Description:** In this activity, you create a shell script that searches for the scr\_3-2 file, starting in your home directory. You then use the direct assignment method with two variables and use them as arguments for the find command.

1. Open a terminal window, type cd scripts, and press **Enter**.
2. Type gedit scr\_3-3 and press **Enter** to open a new empty file in the gedit editor.
3. Type the following information, pressing **Enter** after each line:

1 #! /bin/bash

2 STARTLOCATION=$HOME

3 FILENAME=scr\_3-2

4 echo "Searching for the file named $FILENAME"

5 echo "in the $STARTLOCATION directory"

6 find $STARTLOCATION -name $FILENAME

1. Save and close the file.
2. Give the user permission to run the script by typing chmod 744 scr\_3-3 and pressing **Enter**. In this case, you used the numeric method to assign the permission.
3. Remember that you can confirm the permissions enabled for each file (scr\_3-2 and scr\_3-3) by typing ls –l.
4. Run the script by typing ./scr\_3-3 and pressing **Enter**.

Script scr\_3-3is used to find a file. **Line 2**creates the ***STARTLOCATION*** variable and uses the direct assignment method to assign your home directory as the value. **Line 3**creates the ***FILENAME*** variable and uses direct assignment to assign scr\_3-2 as the value. **Lines 4** and **5** use the echo command to display the variables you have assigned values to. **Line 6** introduces the find command, which searches for files in the directory tree starting from the location specified in the command.

You may notice that some directories (such as dbus, gvfs or cache/dconf) may output a “Permission denied” message.

In this activity, the -name option is used to specify searching for a file based on its name, which is the value of the ***FILENAME*** variable in this example, defines this value as scr\_3-2.

Table 3.2 describes some options that can be used with the command find:

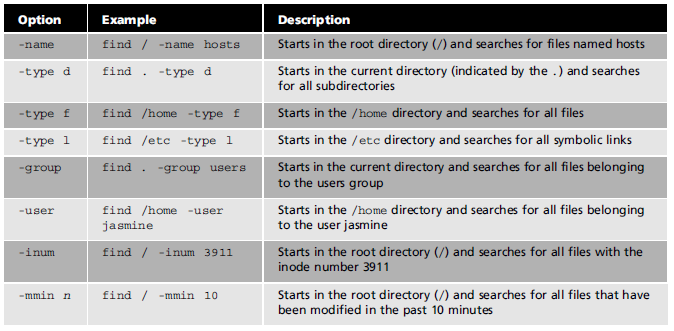


Table 3.2

**2.3 The Prompt Method**

With the **prompt** method, the user is asked to enter a value for the variable. This method is useful when you need information from the user to complete the script, such as a script that asks for the user’s first name. In Activity 3-4, you use this method to ask the user to enter the value of a variable.

### Activity 3.4

### Using the Prompt Method

**Time Required:** 10 minutes.

**Objective:** Create a script with the prompt method for storing a value in a variable.

**Description:** In this activity, you create a shell script that prompts the user for a starting location and filename, and then searches for the filename starting at the location the user chooses.

1. Open a terminal window, type cd scripts, and press **Enter**.
2. Type gedit scr\_3-4 to open a new empty file in the gedit editor.
3. Type the following information, pressing **Enter** after each line:

1 #! /bin/bash

2 clear

3 echo “Welcome to the FIND script”

4 echo -n "Enter the location (such as /home) where the search should start:"

5 read STARTLOCATION

6 echo -n "What is the name of the file to search for?"

7 read FILENAME

8 echo "Starting search for the $FILENAME file in the $STARTLOCATION directory..."

9 find $STARTLOCATION -name $FILENAME 2> /dev/null

1. Save and close the file.
2. Give the user execute permission to run the script by typing chmod u+x scr\_3-4 and pressing **Enter**.
3. Run the script by typing ./scr\_3-4 and pressing **Enter**. At the first prompt, type / (this allows all directories to be searched) and press **Enter**. At the second prompt, type scr\_3-4 (for the file to search for) and press **Enter**. You should see the location of the file outputted in the terminal.

Script scr\_3-4is used to query any file. **Lines 4** and **6** introduce the –n option for the echo command, which specifies **not starting** a new line so that the user can respond on the same line. The read command in **lines 5** and **7** prompts the user to enter values for the variables ***STARTLOCATION*** and ***FILENAME***. At the end of **line 9**, the 2> /dev/null is used to redirect all error messages (represented by the “2”) to the /dev/null directory instead of displaying them onscreen.

When running this script, the shell tries to examine directories you might not have permission to access – to avoid having to see several mission denied” errors, you can have error messages redirected to this directory (you shouldn't always have error messages redirected, they can give you helpful information when you need to know why some­thing is not working).

**2.4 Positional Parameters**

The **positional parameter** method uses the order of arguments in a command to assign values to variables on the command line. Variables from $0 to $9 are available, and their values are defined by what the user enters. This method is useful in a script when you might want it to run differently each time it is used.

For example, when you issue the command to run a script, its name as well as all arguments following it are stored in positional parameters. For instance, to run the scr\_3-2 script, you used the command . /scr\_3-2

The filename is considered position 0 in the command, and the text ./scr\_3-2 becomes the value of the $0 variable. In this example, there are no other arguments, so $1, $2, and so forth have no value. If you want the value of $1 to be /home, the user running specifies this argument as . /scr\_3-2 /home

If you create a script that searches for a file the user specifies, you can assign $1 as the value of the first argument and use it as the filename to search for. When a user needs to search for a specific file, the user enters it on the command line with the name of the script, and then runs the script. As an example, say the user is looking for a file called file-l, and the scr\_3-2script uses the $1 variable to find a file. Your script would look like this:

#! /bin/bash clear

echo "Searching for $1" find $1

When the user enters the ./scr\_3-2 file-1 command, the output is as follows:

Searching for file-l

file-l

In this example, the first argument is $0 or ./scr\_3-2, and the second argument is file-1. Table 3.3 describes the positional parameters:

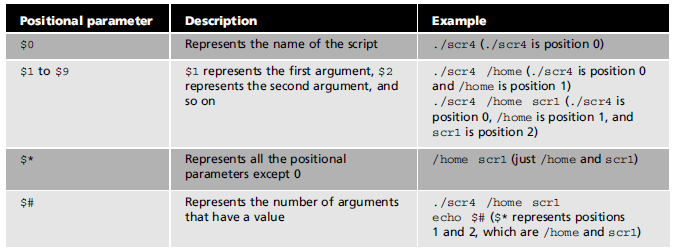


Table 3.3

### Activity 3.5

### Using the Positional Parameters

**Time Required:** 10 minutes.

**Objective:** Create a script that uses positional parameters to assign values to variables.

**Description:** In this activity, you create a shell script that uses positional parameters to deter­mine where to start a search and which file to search for.

1. Open a terminal window, type cd scripts, and press **Enter**.
2. Type gedit scr\_3-5 to open a new empty file in the gedit editor.
3. Type the following information, pressing **Enter** after each line:

1 #! /bin/bash

2 clear

3 echo "Searching for $2 starting in the $1 directory"

1. find $1 -name $2 2> /dev/null
2. Save and close the file.
3. Give the user execute permission to run the script by typing chmod u+x scr\_3-5 and pressing **Enter**.
4. Run the script, giving position 1 ($1) the value /home and position 2 ($2) the value scr\_3-5. This is done by typing:

./scr\_3-5 /home scr\_3-5

and pressing **Enter**. You should see the result of the query of file scr\_3-5.

Remember to add arguments when using a script that requires positional parameters. For instance, if you use ./scr\_3-5 without any parameters on the command line, the script will not run correctly because $1 and $2 will have no value.

**2.5 Exit Status Codes**

Before working with conditions, you need to understand exit status codes. When you quit a program or a command, a numeric code called an **exit status code** is sent to the shell. These codes differ, depending on the Linux distribution or the shell. However, successful commands usually return the code 0, and failures return a value greater than 0; the code is not actually displayed onscreen, but you can reference it via a script or at the command line with the $? variable, as shown in the following example:

echo $?

0

cd baddir

bash: cd: baddir: No such file or directory

echo $?

1

In this example, the first echo $? command is successful and returns the exit status code 0. The second time this command is used, the exit status code 1, indicating failure, is returned because the user tried to change to a directory that does not exist.

**2.6 Conditions**

Although commands in shell scripts are often carried out in order, sometimes you need the interpreter to skip commands based on a condition. For instance, you might want a portion of the script to run if the user is in the Marketing Department and have another portion run if the user is in Human Resources. The if statement is used to carry out certain based on testing a condition.

The following list describes common condition statements used in scripts:

* if statement— starts the condition being tested;
* then statement — starts the portion of code specifying what to do if the condition evaluates to true;
* else statement — starts the portion of code specifying what to do if the condition evaluates to false;
* fi statement — indicates the end of the condition being tested.

The next activity shows how to use conditional statements.

### Activity 3.6:

### Using Conditional Statements

**Time Required:** 10 minutes

**Objective:** Create a script with if, then, and else statements.

**Description:** In this activity, you create a shell script that tests whether a file the user specifies exists.

1. Open a terminal window, type cd scripts, and press **Enter**.
2. Type gedit scr\_3-6 to open a new empty file in the gedit editor.
3. Type the following information, pressing **Enter** after each line:

1 #!/bin/bash

2 clear

3 echo "Enter the name of a file you think is in your current directory."

4 read FILENAME

5 if [ -a $FILENAME ]

6 then

7 echo “You're right! $FILENAME is in your current directory.”

8 else

9 echo “Sorry, $FILENAME is not in your current directory.”

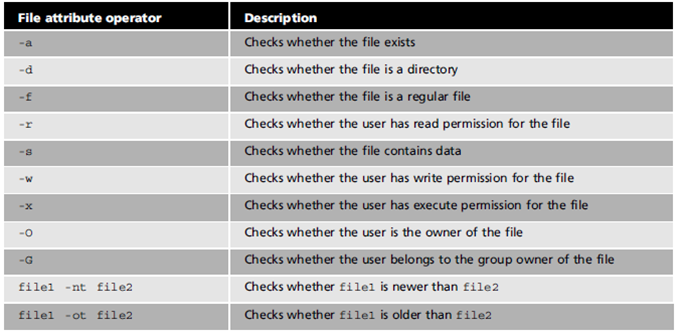
10 fi

11 echo “Finishing script...”

When using the if statement, you must use an open bracket ([) followed by a space and then type the condition, followed by another space and a closed bracket (]). In other words, there must be a space before and after the condition.

1. Give the user execute permission to run the script by typing chmod u+x scr\_3-6 and pressing **Enter**.
2. Run the script by typing ./scr\_3-6. At the prompt, try for valid and invalid file names.

Within the conditional statements in **line 5**, the -a is a file attribute operator that checks whether a file exists. Table 3.4 lists file attribute operators available in the BASH shell.

Table 3.4

**2.7 Menu Scripts**

You can also use condition statements to create **menu scripts** that allow users to choose from a list of options. The next activity of this lab shows you how to create a menu with if-then statements.

### Activity 3.7

### Creating a Menu Script

**Time Required:** 10 minutes.

**Objective:** Create a menu script with if and then statements.

**Description:** In this activity, you create a menu script with if and then statements, giving users **three** menu items to choose from.

1. Open a terminal window, type cd scripts, and press **Enter**.
2. Type gedit scr\_3-7 to open a new empty file in the gedit editor.
3. Type the following information, pressing **Enter** after each line:
   1. #!/bin/bash
   2. clear
   3. echo “Please select an option:”
   4. echo
   5. echo "1) Display your current directory.”
   6. echo "2) Display your home directory.”
   7. echo "3) List the contents of your current directory.”
   8. echo
   9. read CHOICE
   10. if [ $CHOICE == 1 ]
   11. then
   12. pwd
   13. fi
   14. if [ $CHOICE == 2 ]
   15. then
   16. echo $HOME
   17. fi
   18. if [ $CHOICE == 3 ]
   19. then
   20. ls
   21. fi
   22. echo
   23. echo “Finishing script...”
4. Give the user execute permission to run the script by typing chmod u+x scr\_3-7 and pressing Enter.
5. Run the script several times by typing ./scr\_3-7 and selecting the different available options.
6. Finally, run the script and choose an option that does not exist (e.g. 4).

**BONUS ACTIVITY 3.7.1**: DISPLAY AN ERROR MESSAGE LIKE “INVALID OPTION” FOR ANY OPTION OTHER THAN THE ONES SPECIFIED, AND LOOP THE SCRIPT UNTIL A VALID OPTION IS SELECTED. ALSO, ADD A FOURTH OPTION TO EXIT THE SCRIPT (Hints: you can use nested ifs or the commands case, while and exit).