# A simple-to-hard step by step shell scripting tutorial

By: Elnaz Amanzadeh

Contents

[A simple-to-hard step by step shell scripting tutorial 1](#_Toc524304689)

[Introduction 2](#_Toc524304690)

[Exercise 1: “read” & “touch” 5](#_Toc524304691)

[Exercise 2: Difference between “touch” and “cat” commands: 5](#_Toc524304692)

[Exercise 3**:** For Loops- A simple One 6](#_Toc524304693)

[Exercise 4: for loop- Part 2 7](#_Toc524304694)

[Exercise 5: While looping 8](#_Toc524304695)

[Exercise 6: “grep” 9](#_Toc524304696)

[Exercise7: “if .. else statement” 10](#_Toc524304697)

[Exercise 8: Multiple if statements 11](#_Toc524304698)

[Exercise 8: sed 13](#_Toc524304699)

[Exercise 10: “sed” command- part2 14](#_Toc524304700)

[Exercise 11: “awk” command 16](#_Toc524304701)

[Exercise 12: “awk”- Part2 17](#_Toc524304702)

# Introduction

For beginners in Linux systems, shell scripting seems unfamiliar. Normally, they may ask about the necessity and advantages of shell scripting. As a short answer, shell scripting is a simple way to achieve reproducibility for all actions in Linux system. Simply, you may download some files into a specified folder. After a while, you don’t know where is that file, and if you are using only terminal, it will be a time consuming process to find the file. In more complex activities, same thing is true. In addition, everything is performed using command lines in Linux system, but you don’t want to retype complex and long commands every time you need to use them. You always will need a shell script as a background.

How to create and open a shell script?

Normally, after starting your Linux system, press [Alt+Ctrl+t] to open a new terminal. I want to direct you to Desktop directory and create shell script file there.

For this, type [cd Desktop]. Now, you are in Desktop directory.

Now, check your Desktop. Make sure that the name you are selecting for your script doesn’t already exist: [ls]

And, create your shell script: [cat > Shell1.sh]

Check your Desktop again: [ls]. File, Shell1.sh should be there.

Type; [gedit shell1.sh]. Like its name, shell script is a platform that you can type everything you want to execute later. Therefore, you will need a program to open your script and make any change you want. Such programs are called text editors. Sublime Text, Atom, Vim, Gedit, Nano, GNU Emacs, etc (You can obtain more information about text editors in <https://fossbytes.com/9-best-text-editors-linux-programming-2017/>).

Let’s get back to our script.

First of all, if you want to make your script executable, you should tell this to your system by typing [#! /bin/bash].



Before starting to write please note that a shell script should be as readable as possible and no unnecessary commands should be used. Because when you aim to write a complex shell script with several commands following each other and there were some errors, not only yourself but also other people should people to understand it and correct it or make any change in it.

Let’s begin with a cliché! We want to say Hello to World. In nest lines, type [echo ”Hello World”]

Get back to your terminal and type [./Shell1.sh]

There is an error about Permission. To be sure that you want to execute this script, you should type [chmod +x Shell1.sh]. Now execute your script again. you will see this:



Therefore, use [echo] any time you want to print any phrase which it comes before. How about giving a name for everything want to execute? It will be helpful for calling the same directory in a script for several times. Let’s do this. Open your script. I you want save your first command and also not execute it every time you run ./Shell1.sh, just put a # before it. Hashtag will change every line you point to a comment.

Now, go to the next line and type:

First="Hello World"

echo $First

When you run your script using terminal, it will show the same result as last command. There is a point here. Try the same script without $ sign. Using this sign turns the term into a variable which has been introduced before in the shell.

## Exercise 1: “read” & “touch”

This command lest you to have interactions with users. Open your script and type:

echo “What is your name?”

read USER\_NAME

echo “Hello $USER\_NAME”

echo “I will create a file for you called ${USER\_NAME}\_file”

touch ${USER\_NAME}\_file

I am sure that you know application of read and touch commands but if you don’t know just check your Desktop directory. “read” is a built in function and reads a line, while touch helps you to create your favorite files.

You will probably ask what is the difference between cat and touch commands. To receive your answer, perform Exercise 2.

## Exercise 2: Difference between “touch” and “cat” commands:

Please run following commands and you will find out:

$ls -l file3

$cat file3

$cat > file3

hi

$cat file3

$cat >> file3

hihi

$cat file3

Principally, you won’t be able to run more than two above lines with touch. Please try it and find differences.

## Exercise 3**:** For Loops- A simple One

For Loops are considered as one of the most used commands. They simply many actions in script for us. Don’t wait. Just run the following:

for i in 1 2 3 4 5

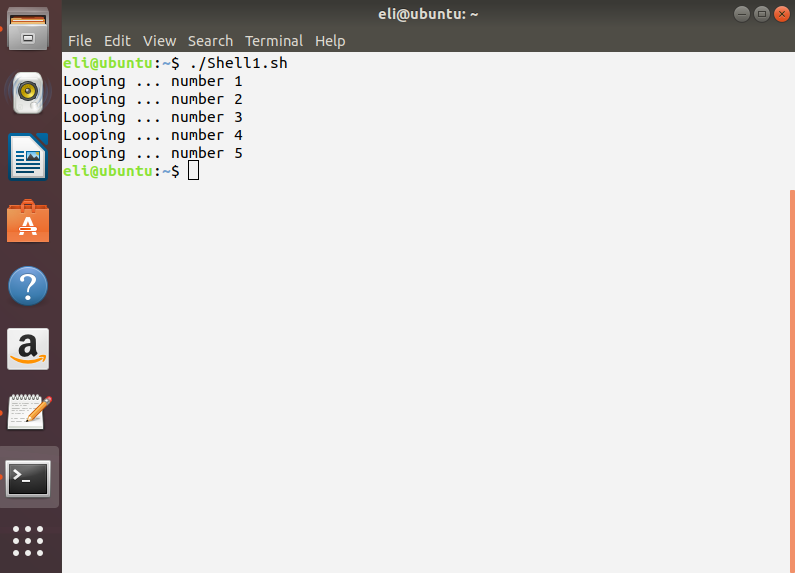
do

echo "Looping ... number $i"

done

As you can see, “do” and “done” are important because they direct initiation and end of a loop.

If you want to writ line 2 and line 3 in the same line, just put a (;) immediately after do and write the next line.



## Exercise 4: for loop- Part 2

If you have a string, you should check whether you have defined all details true or not. Please run the following script:

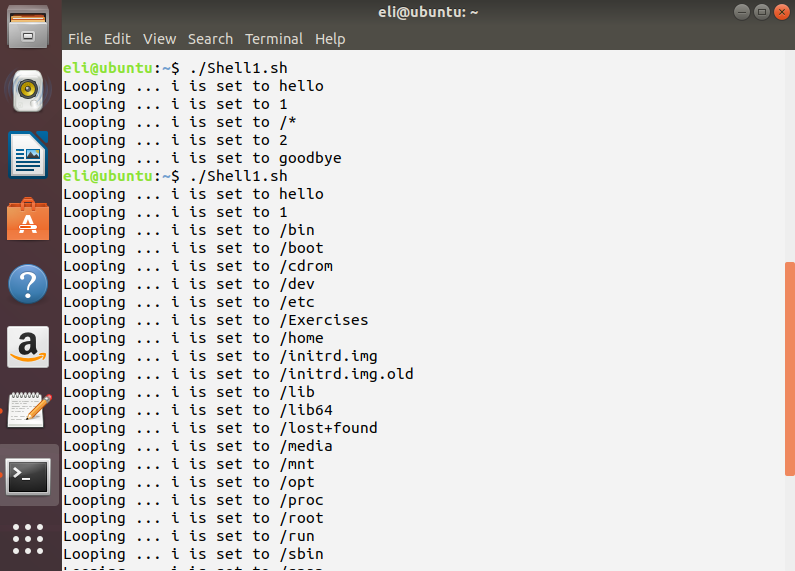
for i in hello 1 /”\*” 2 goodbye

do

echo "Looping ... i is set to $i"

done

Everything seems normal. Now, try it again with no “” around \*. Here is the difference in results:



Note: \* means all. You can use this parameter to point all variables or all files in a special format. Therefore, it is a very useful parameter but you should know where it should be in quotes and where shouldn’t.

## Exercise 5: While looping

While loop is another type of loops which acts like for loop. You also can detect the difference between them by a simple exercise which can you see it and other examples in (https://www.shellscript.sh/loops.html):

INPUT\_STRING=hello

while [ "$INPUT\_STRING" != "bye" ]

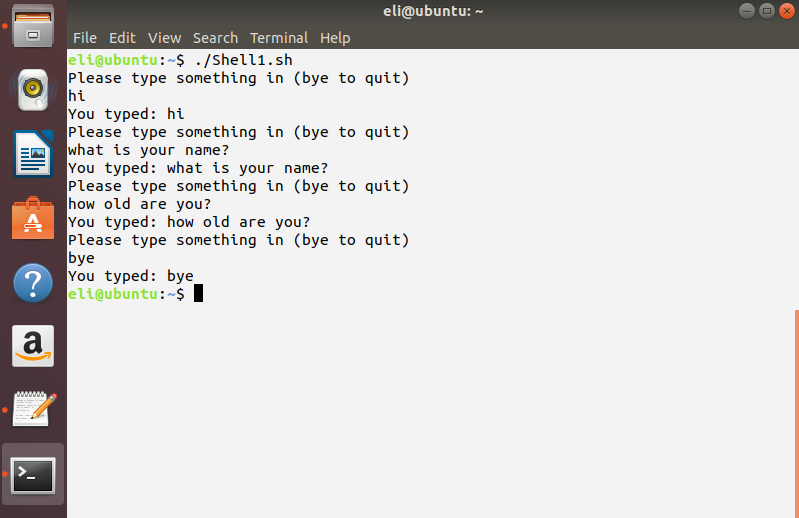
do

echo "Please type something in (bye to quit)"

read INPUT\_STRING

echo "You typed: $INPUT\_STRING"

done



You can continue a while loop until you type “bye” word.

## Exercise 6: “grep”

Similarly, you can also use all mentioned commands in one script. For example, you can run this:

cd ~/Desktop

My\_String=Eli

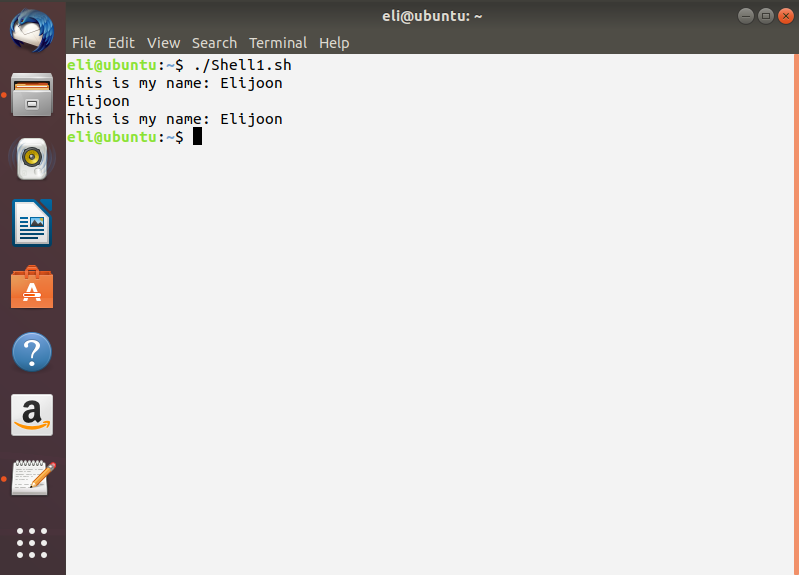
for i in \*.txt

do

cat $i | grep $My\_String

done

You can see the results as below:



This is also a review on what you can do with what you have learned until here. In this exercise, (\*) has been used to point all the text files you have created in Desktop directory. Two commands of (cat) and (grep) have also been in the same line and separated by a (|). In many scripts, you need to write several commands in a tandem way. Therefore, you will need to use “and”. By this way, not only the script will be simpler and more understandable, also reduced the number of probable errors. In order to do this, first option is the use of (|). The second option is the use of (&&) which is often used in (if) statements to determine two conditions with one (if) statement.

\*\*For grep commands: <https://bencane.com/2013/08/19/grepping-a-file-without-using-cat-and-grep-other-tricks/>

\*\*For cat and grep and awk: <http://xahlee.info/linux/linux_shell_text_processing.html>

## Exercise7: “if .. else statement”

An important condition maker in shell scripting is “if .. else statement”. Using it, you can determine your favorite or required ranges. Let’s run this script:

echo "Please guess the number of cars: "

read X

if [ "$X" -le "100" ]; then

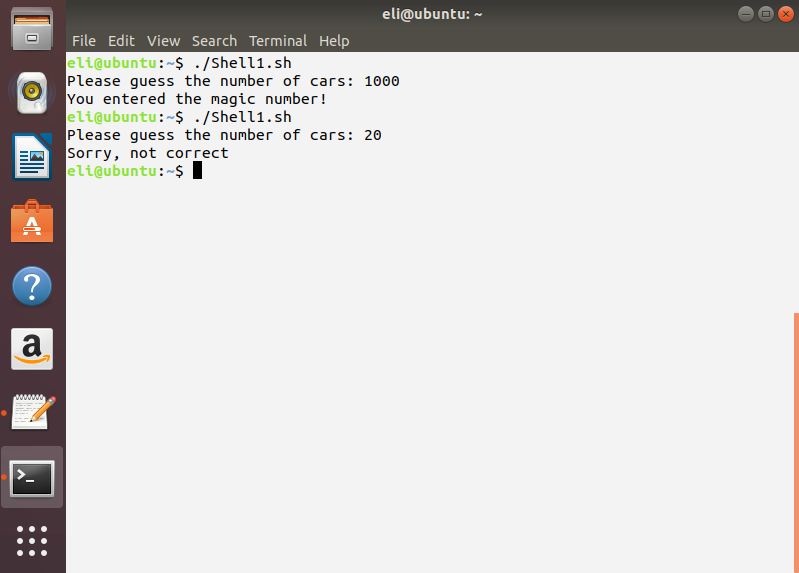
echo "Sorry, not correct"

else

echo "You entered the magic number!"

fi

Result:



Like “for” loops, if .. else statements should have accurate start and end. It is ended by “fi”. You should use “fi” as many as “if”s you are using in your script. In this example, “le” is used to say if $X is less than 100 or equal to 100 it is not correct. “else” means any integer more than 100. You can determine several conditions for any file or database you have and finally obtain your favorite results.

There are other options to determine the range in a script:

lt(a, b): a < (less than) b

le(a, b): a < (less or equal) b

eq(a, b): a = (equal) b

ne(a, b): a != (not equal) b

gt(a, b): a > (greater than) b

ge(a, b): a > (greater or equal) b.

\*\*For “if .. else statements”: https://www.tutorialspoint.com/unix/if-else-statement.htm

## Exercise 8: Multiple if statements

Run the following script several times and each time enter a different value. You will find out how it works and it will help you in your scripts to use several conditions:

echo -en "Please guess the number of cars: "

read X

echo $X | grep "[^0-9]" > /dev/null 2>&1

if [ "$X" -ge "100" ]; then

echo "Sorry, not correct"

else

if [ "$X" -lt "100" ]; then

echo "You are in right area"

if [ "$X" -eq "7" ]; then

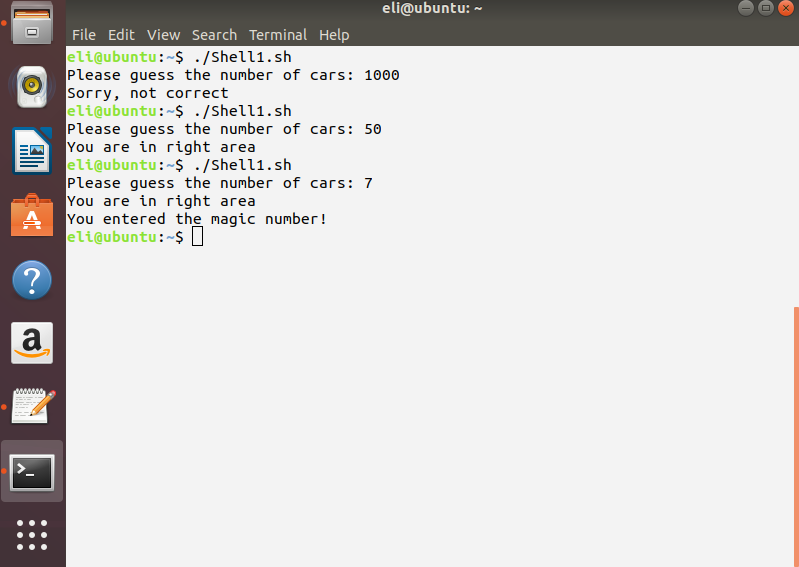
echo "You entered the magic number!"

fi

fi

fi

Results will be like following figure:



It is important to determine correct ranges. In this command first range includes all integers greater than 100, other if statements (conditions) will be applied only if you enter “else” term in the right place.

To read more about if statement and all flags in this command please refer to: <https://linuxacademy.com/blog/linux/conditions-in-bash-scripting-if-statements/>

Another example of if statement in combination with a while loop:

cd ~/Desktop

currfile=1

while read line

do

if [ "$line" = "1" ]

then

(( currfile++ ))

else

echo "$line" >> ${currfile}.txt

fi

done < names.txt

## Exercise 8: sed

sed or stream editor is a very useful command. You see this command very commonly in scripts which work on input output files. You can use this command to replace, delete, add, or highlight terms. But, it is a tricky command with several flags that you can find some of them in html. Please run the following command.

cd ~/Desktop

sed (–e) 's/$/ 1/g' myname.txt

you can once execute it with –e flag and see the results:



The best thing about “sed” is that, changes aren’t saved except when you use (-e) flag. Accordingly, you can modify your file when you will be sure about the commands and changes. In addition, you can perform any changes in your target file using flags. However, when you search about “sed” you will find commands with slashes and back slashes in the style of (s/…/…/). s means substitution. The first part of the expression contains a basic regular expression (regex). The escaped parentheses \( and \) are used to group parts of matched text that can be back referenced by \1 in the second part of the s-command. At the end of the expression is the letter g, which is the "global" flag, which means that the operation should be repeated for every occurrence on the line.

## Exercise 10: “sed” command- part2

“sed” commands will more touchable for you after running the following script on the file you created before called “names.txt” and its content is:

This is your name: Elmirajoon

This is my name: ELi

This is yourname: Elmi

This is yourname: Elmira

This is yourname: Elmirajoon

This is my name: Elnaz

This is yourname: Elmi

This is yourname: Elmira

This is yourname: Elmirajoon

This is my name: Elijoon

This is yourname: Elmi

This is yourname: Elmira

This is yourname: Elmirajoon

This is my name: Elnazjoon

This is yourname: Elmi

This is yourname: Elmira

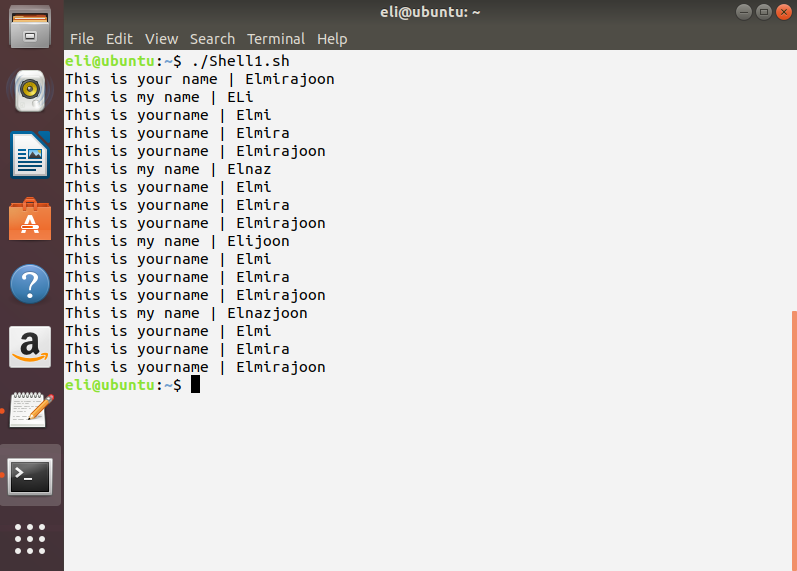
This is yourname: Elmirajoon

In order to change (:) to a pipe (|), tun the following script:

cd ~/Desktop

sed 's/:/ |/g' names.txt

and output is:



similarly, try to replace (:) with (|) in the string of: 101:8:43:4:72:14:41:69:85:3:137:4:3:0:4:0:9:3:0:3:12:3:

In order to do this, try following command:

sed 's/\([^:]\*:\*\):/\1|/g' mynamen.txt

and the following one:

cd ~/Desktop

sed 's/\([^:]\*:[^:]\*\):/\1|/g' mynamen.txt

Now, you can compare two scripts and obtained results. Please analyze details, this will help you to write your own scripts.

For this case, you can consider using other commands. For this, see <https://www.unix.com/shell-programming-and-scripting/228901-awk-sed-change-every-2nd-field.html>.

## Exercise 11: “awk” command

“awk” helps you to easily modify your files, read, parse them, and select special terms or lines. Try the following command on the “names.txt” file you have created before in your Desktop directory.

cd ~/Desktop

awk '{print $1}' names.txt

And you will see this:



The first column of “names.txt” file is “This”. Using “awk”, you can also delete or replace strings. For more details with “awk” you can read <https://likegeeks.com/awk-command/>.

## Exercise 12: “awk”- Part2

awk '

BEGIN {

i = 0

while ( i <= 61000 )

{

print i, 0

i += 50

}

} ' >> output.txt

You can easily combine awk command with other commands and conditions. In this example, while loop has been used to create a set of integers. Please regard all the lines and how each command and flag has been used.

Second example of “awk’ command contains a combination of several commands which have been introduced in this tutorial. Please run it:

cd ~/Desktop

ls -lrt | grep ^- | awk 'END{print $NF}' names.txt

And try the following script:

cd ~/Desktop

ls -lrt | grep ^- | awk 'END{print $OF}' names.txt

You can compare results and realize the difference and little details that you can use to achieve your favorite results. As a short explain, “ls –lrt” has been used to list files and consider newest of them, grep^ has been used to find all the line, and ultimately awk has been used to print “OF” or . More explanations can be found in <http://manpages.courier-mta.org/htmlman1/ls.1.html>, and <https://www.linuxnix.com/awk-scripting-learn-awk-built-in-variables-with-examples/>.

The present tutorial has been prepared to guide beginners of shell scripting. The simple difference of it is its simple but at the same time involving style of guidance. Any beginner who runs all the scripts of this tutorial will touch applications of all introduced functions and commands. Commands and functions have also been selected based on the frequency of their usages in all fields of programming and simplifying scripts. Many of examples here could be written using other commands; however, I tried to show how simply and shortly you can execute your desired modifications and calculations on your files or strings. I also preferred running solving problems in 12 exercises instead of explaining functions of each command because I believe by this way, every one even who is not an expert in computer science will be able to write some scripts. Combination of commands have always been my own problems. Accordingly, I decided to add commands which I introduce in each exercise to the next exercise to make the usage of combination of commands familiar to you. I am sure regarding details will help you to write more complexed scripts. You can contact me from: [elibio449@gmail.com](mailto:elibio449@gmail.com).

Best Wishes