School of Computer Science Engineering and Technology
Bennett University (The Times Group)

Program- BTech-3rd Semester
Course Code- CSET213
Year- 2024
Date- 11/11/2024

Type- Sp. Core-I
Course Name-Linux and Shell Programming
Semester- Odd
Batch- Cyber Security (B9-16)

Lab Assignment 10

Exp No	Name	CO1	CO2	CO3
10	Shell Functions	✓	✓	

Objective: To learn and use shell functions for the development of applications.

Outcomes: After implementing shell functions, the students will be able to create their own user defined functions that can be invoked by other users.

Hands-on Learning on shell functions (60 minutes)

- **Why shell functions?**

  Shell functions are particularly useful if you have certain tasks which need to be performed several times. So, instead of writing out the same code again and again, you may write it once in a function then call that function every time.

  It is a small chunk of code which you may call multiple times within your script. Function is a small script within a script.

- How to create a function?

  To create a function, you must write proper syntax mentioned below:

```
function_name () {
<commands>
}

or

function function_name {
<commands>
}
```

A few points to note:

1. Either method described above for specifying the function is valid. Both functionalities are same and there is no advantage or disadvantage to one over the other. It is just personal preference.
2. In other programming languages it is common to have arguments passed to the function listed inside the brackets (). In Bash they are there only for decoration, and you never put anything inside them.
3. The function definition (the actual function itself) must appear in the script before any calls to the function.

Note: You should pick function names that are descriptive. That way it is obvious what task the function serves.

To do example_1:

```
1. #!/bin/bash
2. Todays_message () {
3. echo Hello Today `date` I am $whoami learning shell function
4. }
5. Todays_message
6. Todays_message
```

Break it down:

- Line 2 - We start defining the function by giving it a name.
- Line 3 - Within the curly brackets we may have as many commands as we like.
- Lines 5 and 6 - Once the function has been defined, we may call it as many times as we like and it will execute those commands.

- **Passing Arguments**

  Sometimes we like the function to process some data for us. You can send data to the function in a similar way like passing command line arguments to a script. You can supply the arguments directly after the function name. Inside the function they are accessible as $1, $2, etc.

  Hands-on example_2

```
1. #!/bin/bash
2. # Passing arguments to a function
3.
4. Todays_message () {
5. echo If you fail, never give up because FAIL
   means "First Attempt in Learning: By $1
6. }
7.
8. Todays_message Abdul Kalam
```

- **Return Values**

Most of the other programming languages have the concept of a return value for functions, a means for the function to send specified data back to the original calling location. Bash functions don't allow us to do this. However, they allow us to set a return status like a program or command exits with an exit status which indicates whether it succeeded or not. Keyword return is used to indicate a return status.

```
1. #!/bin/bash
2. # Setting a return status for a function
3.
4. print_message () {
5. echo $1 scripting is not meant for application program development
   like web. It is generally used for hardware/kernel level
   programming and so return value from function is the status $1
6. return 6
7. }
8. print_message Shell

#exit status of function using $?

10. echo The previous function has a return value of $?
```

Break it down

Line 6 - The return status does not have to be hardcoded. It may be a variable.
Line 10 - The variable $? contains the return status of the previously run command or function.
Note: *Using the default exit status of a function can be a dangerous practice.*

1. Typically, a return status of 0 indicates that everything went successfully. A non-zero value indicates an error occurred.
2. If you want to return a number like the result of a calculation, then you can use the return status to achieve this. However, it is not meant for this purpose but can work.
3. You can also use Command Substitution and have the function print the result.

Hands-on example_3

returnApproach.sh

```
1. #!/bin/bash
2. # Setting a return value to a function
3.
4. countingOfLinesInFile() {
5. cat $1 | wc -l
6. }
7.
8. totalNumberOfLines=$( countingOfLinesInFile $1 )
9.
10.  echo The file $1 has $totalNumberOfLines lines in
```

Break it down:

Line 5 - This statement will print the number of lines of $1 on the stdout file.

Line 8 - Command substitution is used to take what would normally be printed to the stdout file and assign it to the variable totalNumberOfLines.

Execute the below in terminal

user@bash: ./returnApproach.sh myfile.txt

Variable Scope

Scope refers to which parts of a script can see which variables. By default, a variable is global which means that it is visible everywhere in the script. You may also create a variable as a local variable. When you want to create a local variable within a function, it is only visible within that function. To do that you can use the keyword local in front of the variable the first time we set its value.

```
local var_name=<var_value>
```

Note: Generally, it is good to use local variables within functions to keep everything within the function contained. In this way variables are safer from being accidentally modified by another part of the script which happens to have a variable with the same name (or vice versa).

Hands-on example_4

```
1. #!/bin/bash
2. # Experimenting with variable scope
3.
4. varChange () {
5. local var1='local 1'
6. echo Inside function: var1 is $var1: var2 is $var2
7. var1='changed again'
8. var2='2 changed again'
9. }
10.
11.  var1='global 1'
12.  var2='global 2'
13.
14.  echo Before function call: var1 is $var1: var2 is $var2
15.
16.  varChange
17.
18.  echo After function call: var1 is $var1: var2 is $var2
```

Overriding Commands: In shell scripting, it is allowed to name a function as the same name as a command you would normally use on the command line. This allows us to create a wrapper. For example, may be every time we call the command ls in our script, what we want is ls -lh. We could do the following mentioned in the below example_5:

```
1. #!/bin/bash
2. # Create a wrapper around the command ls
3.
4. ls () {
5. command ls -lh
6. }
7.
8. ls
```

Break it down:

Line 5 - When we have a function with the same name as a command, we need to put the keyword command in front of the name when we want the command as opposed to the function as the function normally takes precedence.

Note: In the example above, if you didn't put the keyword command in front of ls on line 5 you would end up in an endless loop. Even though you are inside the function ls when we call ls it would have called another instance of the function ls which in turn would have done the same and so on. If you encounter this then you can cancel the script from running by pressing the keys CTRL c at the same time on your keyboard.

Scripting Problems for Assessment (40 Minutes)

1. Develop a basic calculator that would do the following functions: (20 minutes) **(Odd Batch)**
    a. Addition
    b. Subtraction
    c. Multiplication
    d. Division
    e. Square roots
    f. Percentage

    Requirement:

    i. The values on which the operation to be applied should be taken from the command line arguments.
    ii. Each of the operation should be implemented as function.
2. Develop a shell script to implement function driven program which can (15 minutes) **(Even Batch)**
    a. Display list of users who are currently working in the system
    b. Copying files
    c. Rename a file
    d. List of files in the directory

    Note: All arguments should be taken from command line

Submission Instructions:

1. Submission requires the screen shots of all the incurred steps to execute a shell script or a video showing the whole process.
2. All these files are in single zip folder.
3. Use the naming convention: Prog_CourseCode_RollNo_LabNo.docx (Example: BCA3rdSem_CBCA221_ E21BCA002_Lab1.1)
4. Submission is through LMS only