Seminar #5

Summary (and not only)

Shell functions

Like in C, in Bash we have the opportunity to write user defined functions in order to enable modularity to our programs. Functions enable us to break down the overall functionality of a script into smaller, logical subsections, which can then be called in order to perform some individual task(s) when it is needed. Moreover, using functions to perform repetitive tasks is a good way to create code reuse; code reuse represents an important part of the modern programming principles. A function is also a faster way to run a program, comparative to running a different shell-script.

The general format of a function:

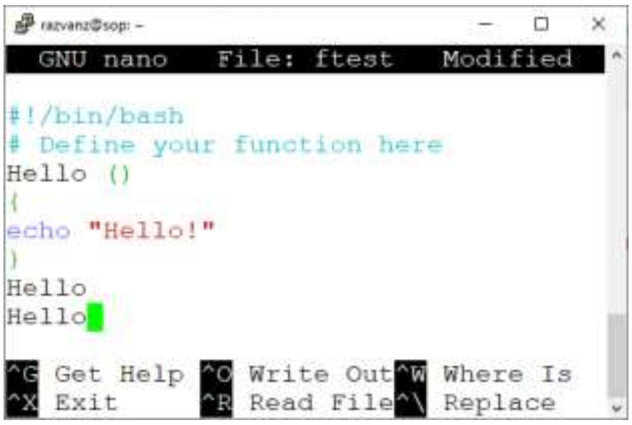
function_name ()

{

list of commands

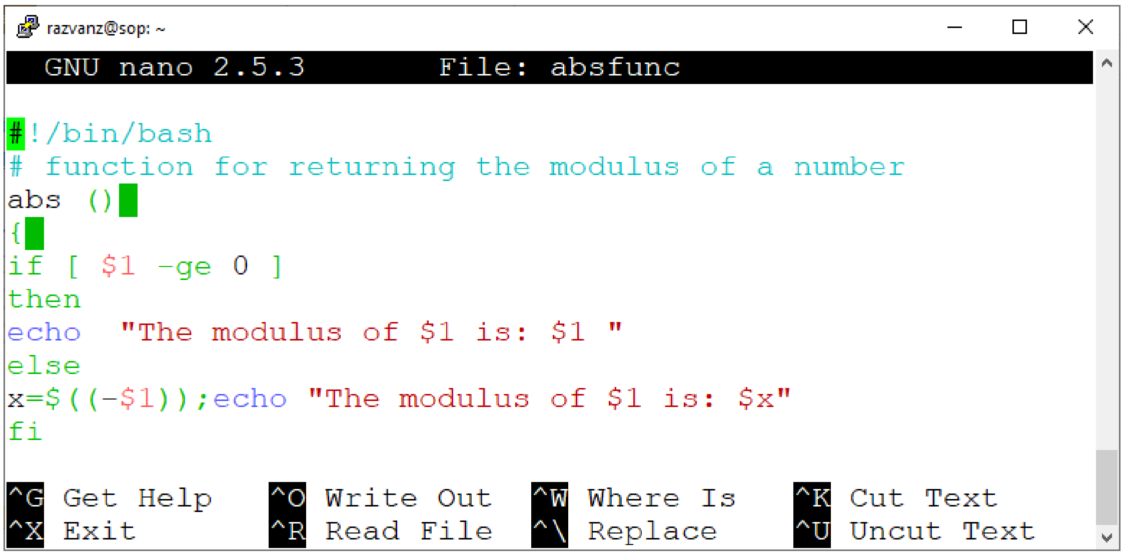
}

The brackets after the function's name indicate to the shell that is about a function declaration. In the following example the ***Hello*** function is defined and it is called two times:



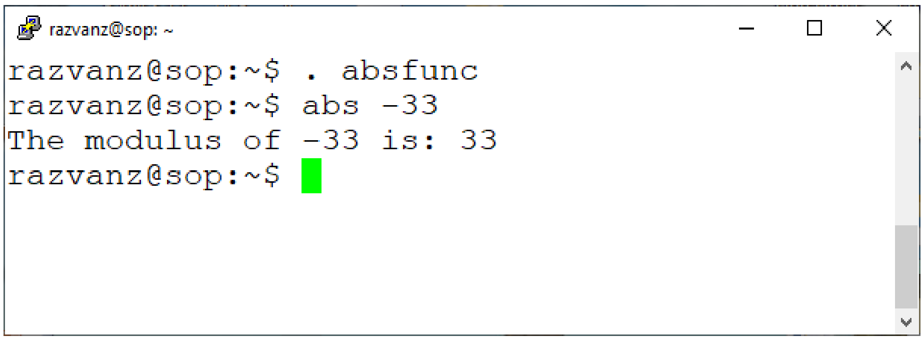
```
  GNU nano    File: ftest    Modified

#!/bin/bash
# Define your function here
Hello ()
{
echo "Hello!"
}
Hello
Hello
```

Let's consider another example, where we define a function ***abs*** that computes the modulus of a given number:



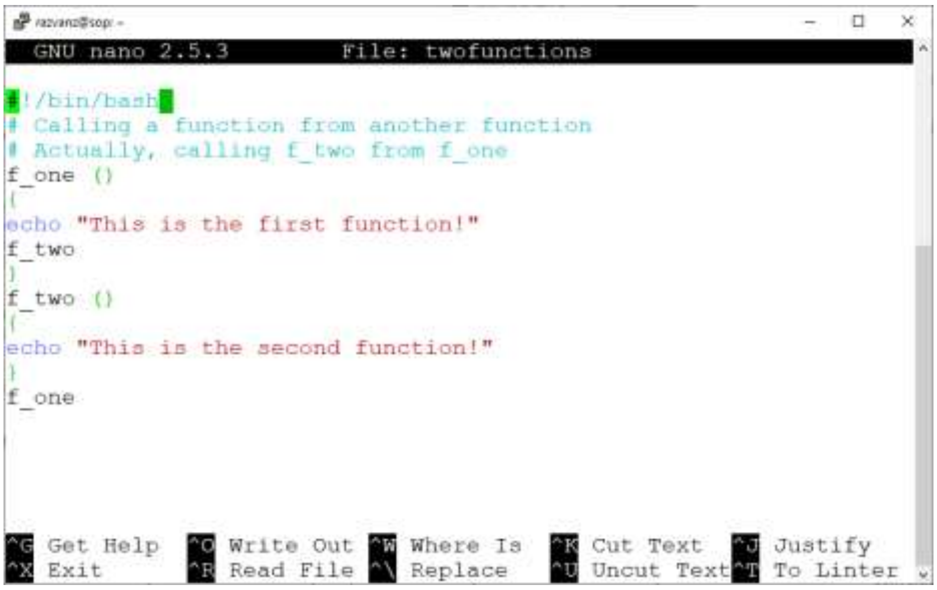
```
razvanz@sop: ~
  GNU nano 2.5.3            File: absfunc

#!/bin/bash
# function for returning the modulus of a number
abs  ()
{
if [ $1 -ge 0 ]
then
echo  "The modulus of $1 is: $1 "
else
x=$((-$1));echo "The modulus of $1 is: $x"
fi

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text
```

In this example $1 is the value of the first positional parameter. Assuming that the shell-script is saved as ***absfunc***, we may load into memory the ***abs*** function using the dot (.) command and after that we call the ***abs*** function like this:



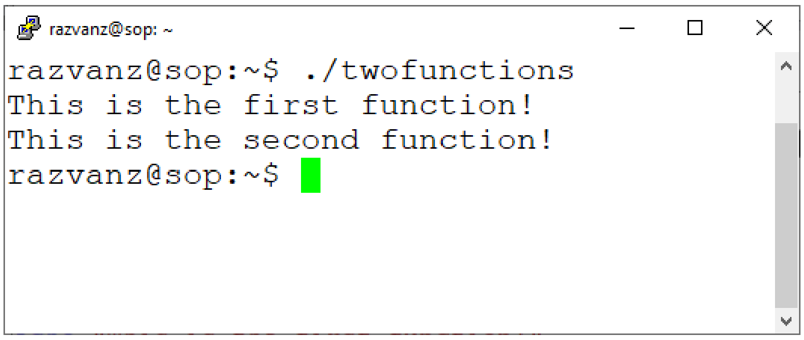
```
razvanz@sop:~$ . absfunc
razvanz@sop:~$ abs -33
The modulus of -33 is: 33
razvanz@sop:~$
```

In the next example we call a function from another function:



```
razvanz@sop: ~
  GNU nano 2.5.3            File: twofunctions

#!/bin/bash
# Calling a function from another function
# Actually, calling f_two from f_one
f_one ()
{
echo "This is the first function!"
f_two
}
f_two ()
{
echo "This is the second function!"
}
f_one

^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify
^X Exit       ^R Read File   ^\ Replace    ^U Uncut Text ^T To Linter
```

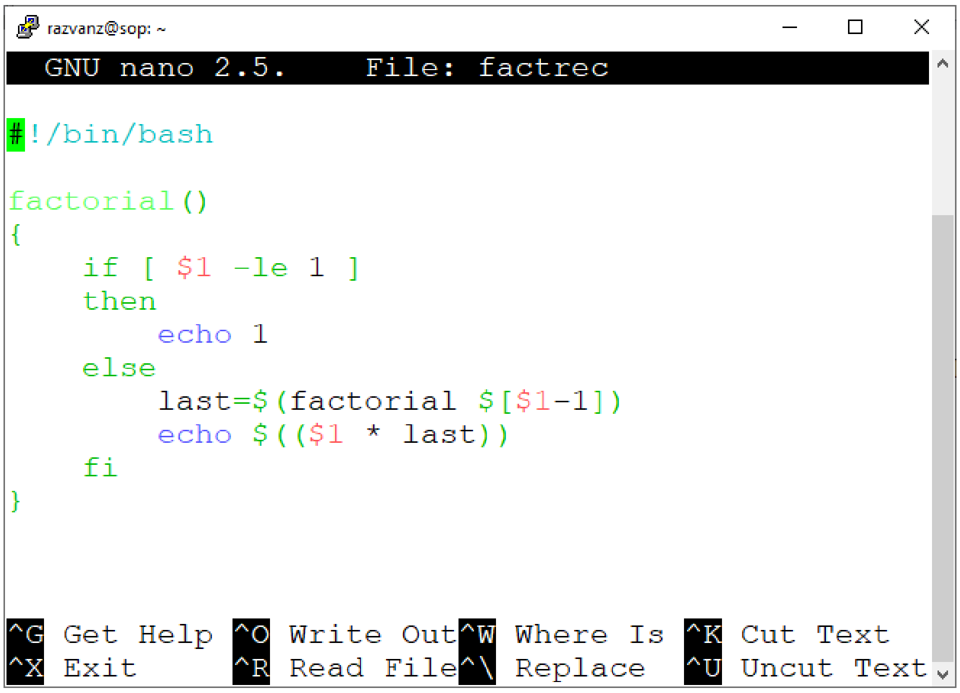
The call of ***f_one*** will provide the following output:



razvanz@sop: ~

```
razvanz@sop:~$ ./twofunctions
This is the first function!
This is the second function!
razvanz@sop:~$
```

We may also have recursive functions. Here is an example of a recursive factorial:



razvanz@sop: ~

```
  GNU nano 2.5.       File: factrec

#!/bin/bash

factorial()
{
    if [ $1 -le 1 ]
    then
        echo 1
    else
        last=$(factorial $[$1-1])
        echo $(($1 * last))
    fi
}



^G Get Help  ^O Write Out^W Where Is  ^K Cut Text
^X Exit      ^R Read File^\ Replace   ^U Uncut Text
```