System Administration (CSNB113) – Lab 8

Topics: Shell scripts and programming.

Sec:

Student-ID:

Name:

This lab exercise is to be submitted **at the end** of the lab session!

We will do a lot of shell programming from here onwards. Hopefully, you are able by now to use vi!?!

Create a small file with vi, with the file name `my_first`. The command to do that is

```
vi my_first
#!/bin/bash
# This  is  my  first  shell  program!
echo "Hello  World"  #  This  is  another  comment
```

The **first line** is the 'shebang' that tells us which command interpreter we want our small program to use. In this case, it will be /bin/bash. How does one know? Simple:

```
which  bash
```

'which' tells us which program will be called when you type a command.
The **second line** is a comment. It is not necessary, but helpful: it is a comment. The third line says what to do: type ('echo') the string "Hello World" on the prompt.

This is not much of a program, since you get the same when you type

```
echo "Hello World"
```

Now you want to **execute**, that is **run** it. This is being done with ./ (DotSlash):

```
./my_first
```

Why would this not work? ls -l my* helps you:

```
ls -l my*
```

The permissions are wrong; the 'x' is missing throughout. What is the correct command to give e'x'ecute permission to owner, group and world?

Try again to run it now:

```
./my_first
```

If your lecturer asks you to type "I will never be naughty" 10 times, it makes a lot of sense to write a program. Call it `my_second`:

```
#!/bin/bash
for ROUND in 1 2 3 4 5 6 7 8 9
10 do
echo "I will never be

naughty" done
```

Run this program, and debug it, if necessary, until it works.

Be careful about the '1 2 3 4 5 6 7 8 9 10'. These are NOT numbers (integers) by default. On the contrary, by default shell script variables are strings.
You can prove that easily by changing one line in your program:

```
#!/bin/bash
for ROUND in Today is a
Wednesday do
echo "I will never be

naughty" done
```

You see that the command interpreter just takes one 'item' after the other when executing a `for`-loop.
Of course, it is also possible to use the loop variable:

```
#!/bin/bash
for ROUND in Today is a
Wednesday do
echo "$ROUND"

done
```

But when you have to write the sentence 100 times? Call it `my_third`:

```
#!/bin/bash
#
number=0
while [ $number -lt 100 ]; do echo
    "I will never be naughty"
    number=$((number + 1))
done
```

*This is a so-called '**while- loop**', because it is repeated (it loops) as long as 'number' is less than 100. Whenever it is looped, the number is increased.*
The ((…)) is specific to shell script: It means, to force the process as arithmetic expression. Shell script is not often used to make mathematical calculations, therefore expressions and variables are by default considered to be strings, as shown above.
On the other hand, you can see that the numeric relationship 'less than' (-lt) is considering the loop variable number *as a numeral.*

Unfortunately, you have been naughty in a number of courses. So you decide to write it for different lecturers. What to do? Simple: copy `my_third` to `my_prog`, using the following command:

---

and then simply edit `my_prog` by inserting a `$1` and a `$2`:

```
#!/bin/bash
#
number=0
while [ $number -lt 100 ]; do
    echo "I will never be naughty $1
    $2" number=$((number + 1))
done
```

Call this program with an argument, like

```
./my_prog  Mr  Johns
```

You see, Mr is the first, Johns the second argument. So `$1` will be Mr, `$2` will be Johns. Once you have to write the sentence 100 times for Puan Eliza, you call it like

```
./my_prog  Puan  Haliza
```

Let's check if it actually echos the lines 100 times:

```
./my_prog  Puan  Haliza  |  wc  -l
```

(wc stands for 'word count' and actually counts characters, words, and lines. Since we only wanted to know the number of lines, option '-l' gives the number of lines only.)

All these programs were doing **repetitions**, and you have seen how a repetition can be programmed.

Next we will look into programs making a **selection**. Name the program '`first_selection`'. It should contain the following lines:

```
#!/bin/bash
if  [  "$1"  =  "1"  ];  then
   echo "The first choice is
nice" elif [ "$1" = "2" ]; then
   echo "The second choice is just as
nice" elif [ "$1" = "3" ]; then
   echo "The third choice is
excellent" else
   echo "I see you were wise enough not to
   choose" echo "You win"
fi
```

Call it by passing the arguments 1 2 3 and nothing:

```
./first_selection  1
./first_selection  2
./first_selection  3
./first_selection
```

*Here, a **comparison** was used: The **parameter passed** (**$1**) is compared with 1, then with 2 and finally with 3. If it **equals** either 1, 2 or 3, the next command is executed. If it equals neither 1 nor 2 nor 3, the **statement 'else'** is executed.*

It is also possible, to insert the output of a shell command into a program. Make sure to name it **`date_and_time`** You will need this program later in this exercise.

```
#!/bin/bash
#
echo  "The  current  date  is:  "  `date`
```

Those strange, backward apostrophes, called ***backticks***, indicate to shell script, that the command **within** these apostrophes (here: date) must be run.

System Administration (CSNB113) – Lab 8

System Administration (CSNB113) – Lab VII

The three major concepts for programming have been introduced:
*1. **Sequence** (one command is executed after the other, from top to bottom)*
*2. **Selection** (it is possible to skip some lines depending on some situation)*
*3. **Repetition** (a certain number of lines are repeated as often as one desires)*

Often, we need to use a variable in programming; and likewise in shell script. In shell script, we have agreed (convention) to use UPPERCASE for variables. In the following, we will write one small program using a variable.

```
#!/bin/bash
#
# our first program containing a shell
variable FIRSTVAR="Uniten"
echo  "I  am  student  of  $FIRSTVAR"
```

Write and run this program.

You can see, that the Dollar-Sign ($) replaces the variable FIRSTVAR. What happens if you remove this Dollar sign? (Edit the file!)

_____

_____

Last, make your program **`date_and_time`** available ***system-wide*** (check the lecture slides on the HowTo).

*Hint: Usually, we place programs that are **added** to the system by the user (system administrator), and to be used by all users, into* `/usr/local/bin/`.

Before you continue, make sure that your program is now a ***system-wide*** program! (By going to any other directory and check if you can still run the program.)

In order to prove that you did the right thing,

```
$ script
Lab_seven cd /
$ which  date_and_time
$ date_and_time
$ exit
$ echo  sn012345  |  mailx  -a  Lab_seven  \
          -s  "Lab  seven"  surizal@metalab.uniten.edu.my
```

The explanation here is rather simple:

1. `echo  sn012345` *prints* `sn012345` *to the standard output, but here to the* pipe
2. `mailx` *sends a mail, with an attachment (*`'-a'`*) of file* `Lab_seven`*, and*
3. `Subject:  Lab  seven` *to*
4. `surizal@metalab.uniten.edu.my`

Don't forget to submit your lab sheet!