


Tutorial12: Shell Scripting - Part 2

 **DO NOT USE THIS VERSION OF THE LAB. This page will no longer be updated.**
New version here: <https://seneca-ictoer.github.io/ULI101/A-Tutorials/tutorial12>
Andrew's students please go here: http://wiki.littlesvr.ca/wiki/OPS145_Lab_11

Contents

- 1 ADDITIONAL SHELL SCRIPTING
 - 1.1 Main Objectives of this Practice Tutorial
 - 1.2 Tutorial Reference Material
- 2 KEY CONCEPTS
 - 2.1 IF-ELIF-ELSE STATEMENT
 - 2.2 FOR LOOP USING COMMAND SUBSTITUTION
 - 2.3 WHILE LOOP
 - 2.4 EXIT & BREAK STATEMENTS
 - 2.5 START-UP FILES
- 3 INVESTIGATION 1: ADDITIONAL LOGIC STATEMENTS
- 4 INVESTIGATION 2: ADDITIONAL LOOPING STATEMENTS
- 5 INVESTIGATION 3: exit AND break STATEMENTS
- 6 INVESTIGATION 4: USING START-UP FILES
- 7 FURTHER STUDY
- 8 LINUX PRACTICE QUESTIONS

ADDITIONAL SHELL SCRIPTING

Main Objectives of this Practice Tutorial

- Use the **if-elif-else** control flow statement in a shell script.
- Use the **for** loop control using a list with **command substitution**.
- Use the **while** loop in a shell script.
- Use the **exit** and **break** statements in a shell script.
- Explain how to configure and use a **.bashrc** start-up file.

Tutorial Reference Material

Course Notes	Linux Command/Shortcut Reference	YouTube Videos
Slides: <ul style="list-style-type: none">▪ Week 12 Lecture 1 Notes:	Control Flow Statements: <ul style="list-style-type: none">▪ if-elif-else (https://www.tutorialspoint.com/unix/if)	Startup Files: <ul style="list-style-type: none">▪ Purpose (http://www.gnu.or Brauer Instructional Videos: <ul style="list-style-type: none">▪ Bash Shell Scripting - Part 2 (https://www.youtube.co

PDF (<https://wiki.cdot.senecacollege.ca/uli101/slides/ULI101-12.1.pdf>) | PPTX (<https://wiki.cdot.senecacollege.ca/uli101/slides/ULI101-12.1.pptx>)

- `-else-statement.htm`)
- `for Loop` (<https://www.cyberciti.biz/faq/bash-for-loop/#:~:text=A%20'for%20loop'%20is%20a,file%20using%20a%20for%20loop.>)
- `while Loop` (https://bash.cyberciti.biz/guide/While_loop)

- `Examples` (<http://www.linuxfromscratch.org/blfs/view/svn/postlfs/profile.html>)

[m/watch?v=XVTwbINXnk4&list=PLU1b1f-2Oe90TuYfifnWulINjMv_Wr16N&index=6](https://www.youtube.com/watch?v=XVTwbINXnk4&list=PLU1b1f-2Oe90TuYfifnWulINjMv_Wr16N&index=6))

Additional Statements:

- `exit` (<https://www.geeksforgeeks.org/exit-command-in-linux-with-examples/#:~:text=exit%20command%20in%20linux%20is,last%20command%20that%20is%20executed.&text=After%20pressing%20enter%2C%20the%20terminal%20will%20simply%20close.>)
- `break` (<https://www.geeksforgeeks.org/break-command-in-linux-with-examples/#:~:text=break%20command%20is%20used%20to,The%20default%20number%20is%201.>)

KEY CONCEPTS

IF-ELIF-ELSE STATEMENT

The **elif** statement can be used to perform additional conditional tests of the previous test condition tests **FALSE**. This statement is used to make your logic control-flow statement to be more adaptable.

How it Works:

If the test condition returns a **TRUE** value, then the Linux Commands between **then** and **else** statements are executed.

If the test returns a **FALSE** value, then a **new condition is tested again**, and action is taken if the result is **TRUE**, then the Linux Commands between **then** and **else** statements are executed. **Additional elif statements** can be used if additional conditional testing is required. Eventually, an action will be taken when the final test condition is **FALSE**.

Example:

```

num1=5
num2=10
if test $num1 -lt $num2
then
    echo "Less Than"
elif test $num1 -gt $num2
then
    echo "Greater Than"
else
    echo "Equal to"
fi

```

FOR LOOP USING COMMAND SUBSTITUTION

Let's issue the **for** loop with a **list** using **command substitution**. In the example below, we will use command substitution to issue the **ls** command and have that output (filenames) become arguments for the **for** loop.

Example:

```

for x in $(ls)
do
    echo "The item is: $x"
done

```

WHILE LOOP

The **while** loop is useful to loop based on the result from a test condition or command result. This type of loop is very useful for **error-checking**.

How it Works:

The condition/expression is evaluated, and if the condition/expression is **TRUE**, the code within ... the block is executed. This repeats until the condition/expression becomes **FALSE**.

Reference: https://en.wikipedia.org/wiki/While_loop

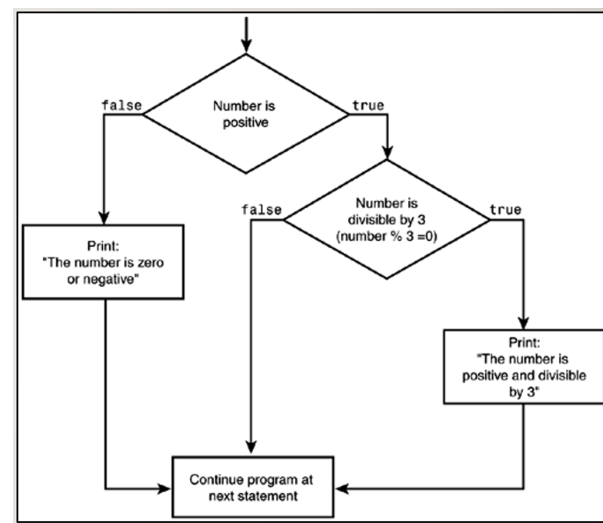
Example 1:

```

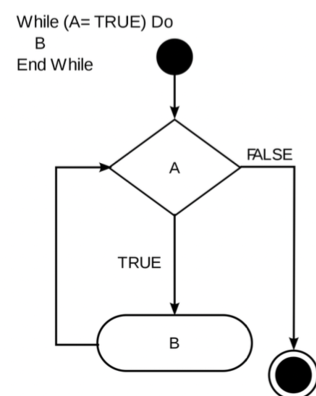
answer=10
read -p "pick a number between 1 and 10: " guess
while test $guess -eq 10
do
    read -p "Try again: " guess
done
echo "You are correct"

```

Example 2:



Example of how an **if-elif-else** statement works.
(Image licensed under cc (<https://creativecommons.org/licenses/by-sa/3.0/>))



Example of how a **while** loop works.

(Image licensed under cc (<https://creativecommons.org/licenses/by-sa/3.0/>))

```

value=1
while [ $value -le 5 ]
do
    echo "$value"
    ((value=value+1)) # could also use ((value++))
done
1
2
3
4
5

```

EXIT & BREAK STATEMENTS

exit Statement

The **exit** statement is used to **terminate** a shell script.

This statement is very useful when combined with logic in a shell script.

The exit command can contain an argument to provide the exit status of your shell script.

Example:

```

if [ $# -ne 1 ]
then
    echo "USAGE: $0 [arg]"
    exit 1
fi

```

break Statement

The **break** statement is used to **terminate a loop**.

Although the loop terminates, the shell script will continue running.

Example:

```

read -p "Enter a number: " number
while [ $number -ne 5 ]
do
    read -p "Try again. Enter a number: " number
    if [ $number -eq 5 ]
    then
        break
    fi
done

```

START-UP FILES

Shell configuration (start-up) files are **scripts** that are run when you log in, log out, or start a new shell.

The start-up files can be used, for example, to **set the prompt and screen display, create local variables,** or create temporary Linux commands (**aliases**)

The **/etc/profile** file belongs to the root user and is the first start-up file that executes when you log in, regardless of shell.

User-specific config start-up files are in the user's home directory:

- `~/.bash_profile` runs when you log in.
- The `~/.bashrc` runs when you start an interactive Bash shell.

Logout Files

There is a file that **resets or restores the shell environment** (including shut-down of programs running in the shell) when the user logs out of their shell. User-specific logout start-up files are in the user's home directory: `~/.bash_logout`

INVESTIGATION 1: ADDITIONAL LOGIC STATEMENTS

ATTENTION: This online tutorial will be required to be completed by **Friday in week 13 by midnight** to obtain a grade of **2%** towards this course

In this investigation, you will learn additional control-flow statements to allow your shell scripts to be even **more adaptable**.

Perform the Following Steps:

1. **Login** to your matrix account.
2. Issue a command to **confirm** you are located in your home directory.
3. Issue a Linux command to create a directory called **advanced**
4. Issue a Linux command to change to the **advanced** directory.
5. Issue a Linux command to confirm you are located in the **advanced** directory.

In **tutorial 10**, you learned how to use the **if** control-flow statement. You will now learn to use the **if-else** statement to take two different actions based on if the condition tests either TRUE or FALSE.

6. Use a text editor like vi or nano to create the text file called **if-4.bash** (eg. **vi if-4.bash**)
7. Enter the following lines in your shell script:

```
#!/bin/bash
clear
read -p "Enter the first number: " num1
read -p "Enter the second number: " num2
if [ $num1 -gt $num2 ]
then
    echo "The first number is greater than the second number."
elif [ $num1 -lt $num2 ]
then
    echo "The first number is less than the second number."
else
    echo "The first number is equal to the second number."
fi
```

8. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).

9. Issue the following linux command to add execute permissions for your shell script:

```
chmod u+x if-4.bash
```

10. Run your shell script by issuing: **./if-4.bash**

Try running the script several times with numbers different and equal to each other to confirm that the shell script works correctly.

A classic shell script to demonstrate many different paths or actions to take depending of multiple testing using an **if-elif-else** statement would be a **mark to letter grade converter**.

11. Use a text editor like vi or nano to create the text file called **if-5.bash** (eg. **vi if-5.bash**)

12. Enter the following lines in your shell script:

```
#!/bin/bash
clear
read -p "Enter a mark (0-100): " mark
if [ $mark -ge 80 ]
then
    echo "You received an A grade."
elif [ $mark -ge 70 ]
then
    echo "You received a B grade."
elif [ $mark -ge 60 ]
then
    echo "You received a C grade."
elif [ $mark -ge 50 ]
then
    echo "You received a D grade."
else
    echo "You received an F grade."
fi
```

13. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).

14. Issue the following linux command to add execute permissions for your shell script:

```
chmod u+x if-5.bash
```

15. Run your shell script by issuing: **./if-5.bash**

What do you notice? Run several times to confirm that the shell script runs correctly for all mark (grade) categories.

16. Issue the following to run a checking script:

```
-uli101/week12-check-1
```

17. If you encounter errors, make corrections and **re-run** the checking script until you receive a congratulations message, then you can proceed.

In the next investigation, you will learn more about the **for** loop and learn how to use the **while** loop for **error-checking**.

INVESTIGATION 2: ADDITIONAL LOOPING STATEMENTS

In this investigation, you will learn more about the **for** loop and learn how to use the **while** loop for **error-checking**.

Perform the Following Steps:

1. Issue a Linux command to confirm you are located in your **advanced** directory in your Matrix account.
2. Issue the following Linux command to view the **./for-1.bash** file:
`more ./for-1.bash`

As you should have noticed from **tutorial 10** that the **for** loop can use a **list**. You can also use the for loop with positional parameters stored as **arguments** from an executed shell script.

You can also use the **for** loop with a list using **command substitution**. Using command substitution is an effective method to loop within a shell script.

Before creating a new shell script, let's learn to use command substitution from the Bash Shell to store arguments as positional parameters and use them for practice.

3. Issue the following linux command to set positional parameters in your current shell:
`set apples oranges bananas pears`
4. Issue the following linux command:
`echo $#`

What do you notice? What does this value represent?

5. Issue the following linux command:
`echo $*`

What do you notice?

These positional parameters can be used with a for loop. To simplify things, let's create another shell script that uses **command substitution** within a **for** loop.

6. Use a text editor like vi or nano to create the text file called **for-3.bash** (eg. `vi for-3.bash`)
7. Enter the following lines in your shell script:

```
#!/bin/bash
clear
set 10 9 8 7 6 5 4 3 2 1
for x
do
    echo $x
done
echo "blast-off!"
```

8. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).

9. **Add execute permissions** for the owner of this script and **run this Bash shell script**.

What do you notice?

Let's create another shell script to **run a loop for each file** that is contained in your current directory using **command substitution**.

10. Use a text editor like vi or nano to create the text file called **for-4.bash** (eg. **vi for-4.bash**)

11. Enter the following lines in your shell script:

```
#!/bin/bash
clear
set $(ls)
echo "Here are files in my current directory:"
echo
for x
do
    echo "    $x"
done
```

12. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).

13. **Add execute permissions** and **run** this Bash shell script.

What do you notice?

We can reduce a line in our shell script by using **command substitution** in the for loop instead of using the **set** command. Let's demonstrate this in another shell script.

14. Use a text editor like vi or nano to create the text file called **for-5.bash** (eg. **vi for-5.bash**)

15. Enter the following lines in your shell script:

```
#!/bin/bash
clear
echo "Here are files in my current directory:"
echo
for x in $(ls)
do
    echo "    $x"
done
```

16. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).

17. **Add execute permissions** for this shell script and **run Bash shell script**

What do you notice? Does the output for this shell script differ from **for-4.bash**? Why?

We now want to introduce you to the use of **error-checking**.

18. Use the **more** command to view the previously-created Bash shell script **/if-5.bash** (eg. **more ./if-5.bash**)

Take a few moments to re-familiarize yourself with this shell script

19. Run your shell script by issuing: **./if-5.bash**

When prompted, enter a **letter** instead of a *number*. What happens?

Let's edit the **for-5.bash** shell script to perform **error-checking** to force the user to enter a numeric value between **0** and **100**.

NOTE: The **while** statement can be used with the **test** command (or a simple linux command or a linux pipeline command) for error checking. In our case, we will use a pipeline command with extended regular expressions. In order to loop while the result is TRUE (not FALSE), you can use the negation symbol (!) to set the test condition to the opposite.

20. Use a text editor like vi or nano to edit the text file called **./if-5.bash** (eg. **vi ./if-5.bash**)
21. Add the following lines in your shell script IMMEDIATELY AFTER the read statement to prompt the user for a mark:

```
while ! echo $mark | egrep "^[0-9]{1,}$" > /dev/null 2> /dev/null
do
    read -p "Not a valid number. Enter a mark (0-100): " mark
done
```
22. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).
23. Run your shell script by issuing:
./if-5.bash
24. When prompted, enter a **letter** instead of a *number*. What happens?
Does the shell script allow you to enter an invalid grade like **200** or **-6**?

Let's add an **additional error-checking loop** to force the user to enter a number between **0** and **100**.

Compound operators like **&&** and **||** can be used with the **test** command.

Let's use the **||** compound criteria to to NOT accept numbers **outside** of the range **0** to **100**.

25. Use a text editor like vi or nano to edit the text file called **./if-5.bash** (eg. **vi ./if-5.bash**)
26. Add the following lines in your shell script IMMEDIATELY AFTER the PREVIOUSLY ADDED error-checking **while** loop statement to **force** the user to enter a valid number (between 1 and 100):

```
while [ $mark -lt 0 ] || [ $mark -gt 100 ]
do
    read -p "Invalid number range. Enter a mark (0-100): " mark
done
```
27. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).
28. Run your shell script by issuing:
./if-5.bash
29. When prompted, enter a **letter** instead of a *number*. What happens?
Does the shell script allow you to enter an **invalid grade** like **200** or **-6**?

Let's reinforce **math operations** in a shell script (that you created in **tutorial 10**) and then incorporate math operations within a loop.

30. Use a text editor like vi or nano to create the text file called **for-6.bash** (eg. **vi for-6.bash**)
31. Enter the following lines in your shell script:

```
#!/bin/bash
value=1
while [ $value -le 5 ]
do
    echo "$value"
```

```
value=value+1
done
```

32. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).
33. Set execute permissions for this shell script and run your shell script by issuing: **./for-6.bash**

You should have noticed an error message.

34. To demonstrate what went wrong, issue the following **commands**:

```
num1=5;num2=10
result=$num1+$num2
echo $result
```

Notice that the user-defined variable stores the text "**10+5**" which is NOT the expected result of adding the number **10** and **5**.

As you may recall in **tutorial 10**, we need to convert a number stored as text into a **binary number** for calculations (in this case, advance the value by 1 for each loop).
We can accomplish this by using the math construct **(())**

35. To demonstrate, issue the following set of **commands**:

```
num1=5;num2=10
sum=$(( $num1+$num2 ))
echo $sum

((product=$num1*$num2))
echo $product
```

Let's correct our **for-6.bash** shell script to correctly use math operations.

36. Use a text editor like vi or nano to edit the text file called **for-6.bash** (eg. **vi for-6.bash**)
37. Edit **line 6** and replace with the following:
((value=value+1))

Note: For those familiar with other programming languages, you can achieve the same results by using: **((value++))**

38. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).
39. **Run** this Bash shell script again.

What do you notice this time?

40. Issue the following to run a checking script:
~uli101/week12-check-2

41. If you encounter errors, make corrections and **re-run** the checking script until you receive a congratulations message, then you can proceed.

In the next investigation, you will learn to use the **exit** statement to **terminate the execution of a shell script** if not run with the properly number of arguments and use the **break** statement that will **terminate a loop** but NOT terminate the running of the shell script.

INVESTIGATION 3: **exit** AND **break** STATEMENTS

In this investigation, you will learn to use the **exit** and **break** statements in your shell scripts.

THE EXIT STATEMENT

The **exit** statement is used to terminate a shell script.

This statement is very useful when combined with logic in a shell script to display an **error message** if the command was **improperly executed** and **terminate** the running of the shell script.

The *exit* command can contain return a *value* to provide the **exit status** of your shell script (i.e. TRUE or FALSE value).

Perform the Following Steps:

1. Make certain that you are logged into matrix account.
2. Confirm that you are currently located in the **advanced** directory.
3. Use a text editor like vi or nano to create the text file called **exit.bash** (eg. **vi exit.bash**)
4. Enter the following lines in the **exit.bash** shell script:

```
#!/bin/bash

if [ $# -ne 1 ]
then
    echo "USAGE: $0 [arg]" >&2
    exit 1
fi

echo "The argument is: $1"
```
5. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).
6. Add **execute permissions** for this Bash shell script.
7. Issue the following command (without arguments):
./exit.bash

What did you notice?

Since there are no arguments, the test within the running shell script returns FALSE, then an **error message** with feedback of how to properly issue the shell script with an argument and then **terminates** the Bash shell script.

Notice that the **\$0** positional parameter displays the **name** of the currently running shell script in the USAGE message. This is useful in case you decide to **change** the *name* of the shell script at a later time.

The symbol **>&2** redirects **standard output** from the USAGE message

to **standard error** making like a real error message.

This "*neat redirection trick*" will NOT be considered for evaluation for this coverage.

8. Issue the following Linux command:

```
echo $?
```

What does this **exit status** from the previously issued command indicate?

9. Issue the following command (with an argument):

```
./exit.bash uli101
```

What did you notice this time?

10. Issue the following Linux command:

```
echo $?
```

What does this **exit status** from the previously issued command indicate?

11. Issue the following command (with two arguments and redirecting stderr to a file):

```
./exit.bash uli101 Linux 2> error.txt
```

What did you notice this time?

12. Issue the following Linux command:

```
echo $?
```

13. Issue the following Linux command to confirm that stderr was redirected to a file:

```
cat error.txt
```

THE BREAK STATEMENT

The **break** statement is used to **terminate** a **loop** without terminating the running shell script.

Perform the Following Steps:

1. Make certain that you are logged into matrix account.
2. Confirm that you are currently located in the **advanced** directory.
3. Use a text editor like vi or nano to create the text file called **break-1.bash** (eg. **vi break-1.bash**)
4. Enter the following lines in the **break-1.bash** shell script:

```
#!/bin/bash
```

```
read -p "Enter an integer: " number
```

```
while ! echo $number | egrep "^[0-9]{1,}$" > /dev/null 2> /dev/null || [ $number  
-ne 5 ] 2> /dev/null
```

```
do
```

```
    if [ $number -eq 5 ] 2> /dev/null
```

```
    then
```

```
        break
```

```
    fi
```

```
    read -p "Try again. Enter a valid integer: " number
```

done

```
echo "The number is: $number"
```

5. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).
6. Add execute permissions for this Bash shell script.
7. Issue the following command (without arguments):

```
./break-1.bash
```

When prompted, enter several **invalid** and **valid** integers. Then enter **valid integers** NOT containing the value of **5**. Finally, enter the integer with the value of **5**.

What happens?

Let's use the **break** statement with the **for** loop.

8. Use a text editor like vi or nano to create the text file called **break-2.bash** (eg. **vi break-2.bash**)
9. Enter the following lines in the **break-2.bash** shell script:

```
#!/bin/bash
```

```
for x
do
```

```
    if [ $x = "uli101" ]
    then
        break
```

```
    fi
    echo "Argument is: $x"
done
```

```
echo
echo "Shell script has been completed"
```

10. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).
11. Add execute permissions for this Bash shell script.

12. Issue the following command (with arguments):
- ```
./break-2.bash hwd101 ipc144 uli101 apc100
```

What do you notice? How come **uli101** and **apc100** are NOT displayed but a message appeared at the end of the script that the script completed?

13. Issue the following to run a checking script:
- ```
~uli101/week12-check-3
```

14. If you encounter errors, make corrections and **re-run** the checking script until you receive a congratulations message, then you can proceed.

In the next investigation, we will learn to create / modify **startup files** to customize your Linux shell environment.

INVESTIGATION 4: USING START-UP FILES

In this investigation, you will learn how to **customize** your **Bash Linux shell environment** by creating and testing a **start-up** file.

Perform the Following Steps:

1. Issue a Linux command to change to your **home** directory.
2. Issue a Linux command to confirm you are located in the **home** directory.
3. Use the **more** command to view the contents of the **default start-up** file called **/etc/profile**

This file contains the **default settings** when you open your Bourne shell (eg. if issuing the command **sh**).

4. Use the **more** command to view the contents of the start-up file called **/etc/bashrc**

This file contains the **default settings** when you **open your Bash shell** (eg. if issuing the command **bash**).

Since we are using the **Bash shell** by default, let's create a **customized Bash start-up file**.
This startup file is located in your **home** directory using the name **".bash_profile"**

Let's move your **.bash_profile** file to prevent **accidental overwrite**.

5. Issue the following linux command:
mv ~/.bash_profile ~/.bash_profile.bk

If you experience an error message *"No such file or directory"*,
just ignore this command since there is no **~/.bash_profile** file in your home directory.

6. Use a text editor like vi or nano to create the text file called **~/.bash_profile** (eg. **vi ~/.bash_profile**)

If you are using the nano text editor, refer to notes on text editing in a previous week in the course schedule.

7. Enter the following lines in your shell script (the symbol "[" is the open square bracket symbol):

```
clear
echo -e -n "\e[0;34m"
echo "Last Time Logged in (for security):"
echo
lastlog -u $USER
echo
echo -e -n "\e[m"
```

NOTE: You will notice there is **NO she-bang line** since this is a **start-up** file.

8. Save your editing session and exit the text editor (eg. with vi: press **ESC**, then type **:x** followed by **ENTER**).
9. You can test run the startup file without exiting and re-entering your Bash shell environment.
Issue the following:
. ~/.bash_profile

What do you notice?

10. **Exit** your current Bash shell session.

11. **Login** again to your matrix account.

Did you start-up file customize your Bash shell environment with colours?

NOTE: This is where you can make your Linux shell environment values **persistent** (i.e. saved regardless of exit and login to your Bash Shell such as **aliases**, **umask**, etc.).

12. Issue the following linux command to **restore** your previous settings for your bashrc startup file:

```
mv ~/.bash_profile.bk ~/.bash_profile
```

If you experience an error message "*No such file or directory*", just ignore.

13. **Exit** your current Bash shell session.

14. **Login** again to your matrix account.

What did you notice this time?

FURTHER STUDY

I hope this series of tutorials have been helpful in teaching you basic Linux OS skills.

In order to get efficient in working in the Linux environment requires **practice** and **applying** what you have learned to perform Linux operating system administration including: **user management**, **installing and removing applications**, **network services** and **network security**.

Although you are **NOT** required to perform **Linux administration** for this course, there are useful **course notes** and **TUTORIALS** for advanced Linux server administration that have been created for the Networking / Computer Support Specialist stream:

- OPS245: Basic Linux Server Administration (<https://wiki.cdote.senecacollege.ca/wiki/OPS245>)
- OPS335: Advanced Linux Server Administration (<https://wiki.cdote.senecacollege.ca/wiki/OPS335>)

Take care and good luck in your future endeavours :)

Murray Saul

LINUX PRACTICE QUESTIONS

The purpose of this section is to obtain **extra practice** to help with **quizzes**, your **midterm**, and your **final exam**.

Here is a link to the MS Word Document of ALL of the questions displayed below but with extra room to answer on the document to simulate a quiz:

https://wiki.cdote.senecacollege.ca/uli101/files/uli101_week12_practice.docx

Your instructor may take-up these questions during class. It is up to the student to attend classes in order to obtain the answers to the following questions. Your instructor will NOT provide these answers in any other form (eg. e-mail, etc).

Review Questions:

1. Write code for a Bash shell script that clears the screen, and then prompts the user for their age. If the age entered is less than 65, then display a message that the person is NOT eligible to retire. If the age is equal to 65, then display a message that the person just turned 65 and can retire. If the age is greater than 65, then display the message that the user is over 65 and why have they not have already retired already?
2. Add code to the script created in the previous question to force the user to enter only an **integer** to provide error-checking for this shell script.
3. Write code for a Bash shell script that will prompt the user for a **valid POSTAL CODE**.
A valid postal code consists of the following format: **x#x #x#**
where **x** represents an upper or lowercase letter
and **#** represents a number from 0-9

Also VALID postal codes can consist of no spaces or one or more spaces in the format shown above.

If the user enters an **INVALID postal code**, indicate an error and allow the user to enter the VALID postal code. When the user enters a VALID postal code, then clear the screen and display the VALID postal code.

4. Write code that works similar to the previous question, but have it read an input file called **unchecked-postalcodes.txt** and only save VALID postal codes to a file called:
valid-postalcodes.txt

Design your Bash Shell script to only run if the user enters TWO ARGUMENTS:
unchecked-postalcodes.txt and **valid-postalcodes.txt**

Otherwise, display an error message and immediately exit your Bash Shell script with a false exit value.

5. What is the purpose of the **/etc/profile** startup file?
6. What is the purpose of the **/etc/bashrc** startup file?
7. What is the purpose of the **~/.bashrc** startup file?
8. What is the purpose of the **~/.bash_profile** file?
9. What is the purpose of the **~/.bash_logout** file?
10. Write code for the **~/.bash_profile** file below to clear the screen, welcome the user by their username, and display a list of all users currently logged into your Matrix server. Insert blank lines between each of those elements.
11. Write a command to run the recently created **~/.bash_profile** startup file from the previous question without exiting and re-logging into your Matrix account.

- This page was last edited on 4 September 2023, at 19:37.