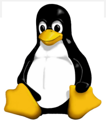
# Computational Science and Scientific Computing Workshop

Elliot S. Menkah, Ph.D
Daniella N. Apeadu

National Institute for Mathematical Sciences, Ghana.
Kwame Nkrumah University of Science and Technology, Ghana.

October 6, 2025 - October 10, 2025

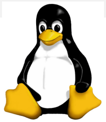
## Linux Command Line



Reading List and Reference Materials?

1. The Linux Command Line - A Complete Introduction by *William Shotts*
2. Linux Command Line and Shell Scripting Bible - *Richard Blum*

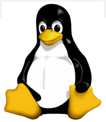
## Linux Command Line



What's Scientific Computing and Why Linux?

- Using computers to analyze and solve problems
  - Eg. Automating daunting and repetitive task such as huge-size matrix vector operations.
- It allows the study of mathematical models of physical phenomena.
- It is used to find optimal system parameters.
- Experimentalists use computers to control experiments and to gather relevant data.

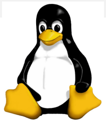
## Linux Command Line



What's Scientific Computing and Why Linux?

- Using computers to analyze and solve problems
  - Eg. Automating daunting and repetitive task such as huge-size matrix vector operations.
- It allows the study of mathematical models of physical phenomena.
- It is used to find optimal system parameters.
- Experimentalists use computers to control experiments and to gather relevant data.

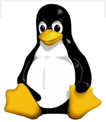
## Linux Command Line



What's Scientific Computing and Why Linux?

- Using computers to analyze and solve problems
  - Eg. Automating daunting and repetitive task such as huge-size matrix vector operations.
- It allows the study of mathematical models of physical phenomena.
- It is used to find optimal system parameters.
- Experimentalists use computers to control experiments and to gather relevant data.

## Linux Command Line



What's Scientific Computing and Why Linux?

- Using computers to analyze and solve problems
  - Eg. Automating daunting and repetitive task such as huge-size matrix vector operations.
- It allows the study of mathematical models of physical phenomena.
- It is used to find optimal system parameters.
- Experimentalists use computers to control experiments and to gather relevant data.

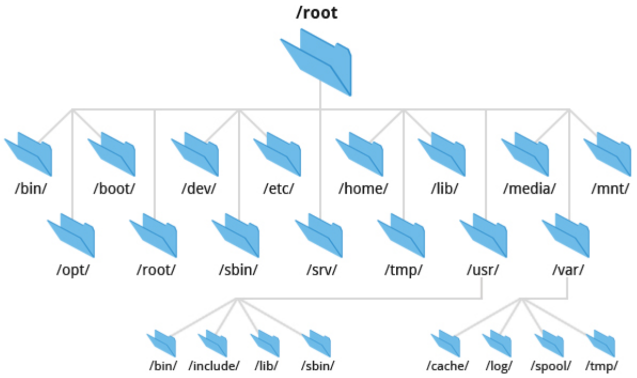
## Linux Command Line - Outline

- File system
  - Linux File system
- Basic Operations
  - Basic Commands
  - File operations
  - User Environment
  - Access Control
  - Process Management
  - Network Management
- Text Editor
  - Vim
- Shell Tools & Programs
  - Shell Program
- Shell Programming
  - Bash Scripting
  - Variables
  - Statements
  - Conditionals
  - Control Sequence / Loops
  - Functions
- Regular Expressions
  - Regular Expression

## Linux Command Line - File system

- Filesystem Types: ext2, ext3, ext4, reiserfs, vfat, xfs, nfs
- Devices: Block devices, Loop devices
- Block Devices:
- inodes
- FHS: Filesystem Hierarchy Standard
- NFS: Network File System

## Linux Command Line - Linux Directory Structure



## Linux Command Line - Basic Operations

- Basic Operations
  - Basic Commands
  - File operations
  - User Environment
  - Access Control
  - Process Management
  - Network Management

## Linux Command Line - Basic Commands

**ls**: list files

**pwd**: present working directory

**cd**: change directory

**cat**: list file content

**alias**: remap a command

**date**: check or set date/time

**uname**: OS info. version and architecture

## Linux Command Line - Basic Commands

- **pwd:** present working directory
- **ls:** list files/directories
- **cd:** change directory

**ls:** list files

```
~ $ ls
```

ls

```
Desktop Pictures
Documents Downloads
```

**pwd:** present working directory

```
~ $ pwd
```

pwd

```
/home/elliott
```

**cd:** change directory
cd *directory-name*

```
~ $ cd Desktop
```

pwd

```
/home/elliott/Desktop
```

Navigate one step(directory) back cd ..

```
~ $ cd ..
```

pwd

```
/home/elliott
```

## Linux Command Line - Basic Commands

**touch**: create new files

**mkdir**: create new directory

**cp**: cp files & directories

**mv**: relocate/move file & directories

**rm**: delete files & directories

## Linux Command Line - Basic Commands ...

- **touch:** create a new file
- **mkdir:** create a new directory
- **cp:** copy files & directories

Create a new file using **touch 'filename'**
Eg.

```
~ $ touch file1
~ $ touch file2

```

Create a new directory using **mkdir 'directoryname'**
Eg. To create two directories called **dir1** and **dir2**

```
~ $ mkdir dir1
~ $ mkdir dir2 dir4 dir5

```

## Linux Command Line - Basic Commands ...

Copy a file from a particular location to another using
**cp 'file-to-be-copied'** *'new-destination'*
Eg. Copy *file1* to directory called **dir1**

```
~ $ cp file1 dir1

```

Copy a directory from a particular location to another. Copying is done **recursively**, [-r]
**cp -r 'directory-to-be-copied' 'new-destination'**
Eg. Copy *file1* to directory called **dir1**

```
~ $ cp -r dir4 dir5

```

More info on commands: https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners/

## Linux Command Line - Basic Commands ...

- **mv:** move/relocate/rename or a file/directory

Move/Relocate a file/directory to a new location using
**mv 'file-to-be-moved' 'new-destination'**
Eg. To place *file2* into **dir2**

```
~ $ mv file2 dir2

```

*'file-to-be-moved'* and *'new-destination'* are actually **paths**.
Linux allows relative paths
Technically, the syntax below shows how it absolute paths works with mv

```
~ $ mv /home/elliott/file2 /home/elliott/dir2

```

Rename a **file** using **mv 'file-to-be-renamed' 'new-name'**
Eg. To rename *file1* into **file3**

```
~ $ mv file1 file3

```

Rename a **directory** using **mv 'directory-to-be-renamed' 'new-name'**
Eg. To rename *dir1* into **dir3**

```
~ $ mv dir1 dir3

```

## Linux Command Line - Process Management

**ps:**: list processes

**kill**: kill processes

**top**: monitor processes

**fuser**: find process owner

## Linux Command Line - User Environment

**env**: command to view user environment

**export**: command to add to user environment

**bashrc**: file to store user environment settings

**profile**: file for global user environment settings. /etc/profile

**tilde**: reference to current user

## Linux Command Line - Basic Commands ...

End of Basic Commands, thank you ...

## Linux Command Line basic tools & File Operations

CLI tools and File operations

## Linux Command Line - File Operations

**echo**: display lines of text or string

**grep**: match string pattern in text

**paste**: join content of files(horizontally)

**cut**: cut out sections of a line of text

**file**: file information

**find**: find files matching

**xargs**: parse as argument

**tar**: de(archive) and (un)compress files

## Linux - Shell Tools: echo

echo

ECHO is a command-line tool used for displaying lines of text or string which are passed as arguments on the command line.

Mostly Used to output status text to the screen or a file

## Linux - Echo Practice

**Structure**

echo [options] string

Eg. : Dump 'Hello Bash' to screen

**Example**

```
$ echo 'Hello World'
```

**Outcome**

```
'Hello World'
```

## Linux - Echo Practice

### Options

1. Options
   - **e :** Allows you to change format of text
   - **n :** Removes preceding newline
2. Escape

   \**a :** For audible alert
   \**b :** backspaces character just before the slash
   \**c :** truncates everything after the slash.
   \**n :** Adds a new-line character
   \**t :** adds a tab character to the output

## Linux - Echo Practice

```
un@mn:~$ echo -e "Hello World!"

```

```
Hello World!

```

```
un@mn:~$ echo -n "Hello World!"


```

```
Hello World!un@mn:~$


```

```
un@mn:~$ echo -e "It is\b red"

```

```
It i  red

```

```
un@mn:~$ echo -e "It is red\n"

```

```
It is  red

```

## Linux - Shell Tools: Echo Practice, Redirect to file

$>$ : Output Redirect to new file

$>>$ : Output Redirect and append to file

Redirect the output of an echo command
**echo [options] 'string' > nameOfFile**
Eg.

```
~ $ echo "Logfile for Today 27/10/2022" > log.txt

```

```
~ $ ls

```

log.txt should be found with other files that may be present in pwd.

```
~ $ log.txt

```

## Linux - Shell Tools: Echo Practice, Redirect to file

To add some more data to **log.txt**
**echo [options] 'string'** $>>$ **log.txt**

```
echo -e "#By Captain Jack Sparrow\n" >> log.txt

```

To verify the content of file **log.txt**
cat *'file-name'*

```
~ $ cat log.txt

```

File should contain:

```
#Logfile for Today 27/10/2022
#By Captain Jack Sparrow

```

## Linux - Shell Tools: grep

**grep**

GREP is a command-line utility for searching plain-text data sets for lines matching a regular expression.

- Line matching and extraction
- Supports Regular Expressions
- Support inverse matching (-v)
- Supports piping

## Linux - Grep Practice

Structure

grep [options] pattern-being-sort [files]

Eg. : Find lines containing text 'Williams' in the file addresses.txt

Example

```
$ grep Williams addresses.txt
```

Outcome

```
Steve Williams
Elizabeth Williams
John Williams
John Williamson
```

## Linux - Grep Practice

### Structure

grep [options] pattern-being-sort [files]

### Options

- **w :** match exact words
- **n :** provide lines of occurrence
- **i :** case-insensitive pattern
- **r :** recursive search and match
- **c :** count
- **A :** Lines After context
- **B :** Lines Before context
- **C :** Lines Before & After context

## Linux - Shell Tools: CUT

cut

CUT is a command-line utility for cutting out sections of string of text.

Cuts out certain section of line from files

cut out byte positions, characters or fields.

Structure

cut [options]... [FILES] ...

- **b :** Extract by bytes
- **c :** Extract by Character
- **f :** Extract by fields

## Linux - Shell Tools: CUT Example -b

williamsfam.txt

```
Steve Williams
Elizabeth Williams
John Williams
John Williamson
```

Example

```
$ cut -b 1,2,3 williamsfam.txt
```

Outcome

```
Ste
Eli
Joh
Joh
```

## Linux - Shell Tools: CUT Example -c

williamsfam.txt

```
Steve Williams
Elizabeth Williams
John Williams
John Williamson
```

Example

```
$ cut -c 2,4 williamsfam.txt
```

Outcome

```
tv
lz
on
on
```

NB: -b and -c can give the same results when dealing with characters.

## Linux - Shell Tools: CUT Example -f

cut [options]... [FILES] ...

**-f option**

-f option uses a tab space as the default delimiter.
The delimiter is denoted by -d and can be changed

**Example**

```
$ cut -d " " -f 1 williamsfam.txt
```

**Outcome**

```
Steve
Elizabeth
John
John
```

## Linux - Shell Tools: paste

**paste**

PASTE is a command-line utility joining files horizontally (parallel merging) by outputting lines consisting of lines from each file specified.

Merges files using tab as delimiter

**Structure**

paste [options]... [FILES] ...

- **-d :** Delimiter
- **-s :** sequential merging

## Linux - Shell Tools: PASTE Example

Checking content of firstnames.txt

```
~ $ cat firstnames.txt
```

```
Jack
Alice
Fred
Kwame
```

Checking content of lastnames.txt

```
~ $ cat lastnames.txt
```

```
Ford
Reynolds
Russo
Mensah
```

**Example**

$ paste firstnames.txt lastnames.txt > fullnames.txt

**Outcome into file fullnames.txt**

Jack Ford
Alice Reynolds
Fred Russo
Kwame Mensah

## Linux Command Line - Practical tools

### Finding files

```
$ find . -type f — xargs grep elliot
```

## Linux Command Line - File Operations ...

End of File Operations, thank you ...

## Linux Command Line - Advanced Shell tools

Advanced Shell tools: AWK, SED, etc ..

## Linux - Shell Tools: AWK

### awk

AWK is a command-line utility that is designed for text processing and typically used as a data extraction and reporting tool.

- it is a tool for manipulating data and generating reports
- it is a filter and cn scan files line by line
- Splits each input line into fields
- Compares input line/fields to pattern and perform action on matches

### Syntax & Structure

awk [options] 'selection_criteria action' input-file > output

## Linux - Shell Tools: Awk Practice

1. With Ref

   > : Output Redirect to new file    >> : Output Redirect and append to file

Extracting the first columns of the data file the

```
~ $ awk '{ print $1}' data_output.dat > log0.txt

```

Outcome of awk command

```
#
#
#Step
0.000000
1.000000
2.000000
3.000000
4.000000
...
```

You can redirect the output of one command as the input of awk

```
~ $ cat  data_output.dat | awk '{ print $1}' > log1.txt

```

Both files, log0.txt and log1.txt should contain the same output.

## Linux - Shell Tools: Awk Practice

Awk assumes a space as the field separator or delimeter.
The field separator or delimeter can be changed by using the flag **-F**
awk -F 'selection_criteria {action}' input-file > output
Eg. To get the users or username in a given linux system, we can extract it form /etc/passwd using ":"

```
~ $ cat /etc/passwd | awk -F ":" '{ print $1}' > users.txt

```

Outcome of awk command

systlog
_apt
tss
uuidd
tcpdump
...

You can also decide to print multiple columns

```
~ $ cat /etc/passwd | awk -F ":" '{ print $1 " " $3}' >
    users.txt

```

## Linux - Shell Tools: Awk Practice

Field separators for both **delimeter field** and **Output Field** can be predefined
awk 'BEGIN{FS=":"; OFS="-"} selection_criteria {action}' input-file > output
Eg. To separate the output by tab spaces:
/etc/passwd using ":"

```
~ $ cat /etc/passwd | awk 'BEGIN{FS=":"; OFS="\t"} { print
    $1, $3}' > users.txt

```

### Outcome of awk command

systlog 104
_apt 105
tss 106
uuidd 107
tcpdump 108
...

## Linux - Shell Tools: Awk Practice

AWK can accept regular expressions to aid filtering
awk 'BEGINFS=":"; OFS="-" selection_criteria {action}' input-file > output
Eg. To get names starting with "ic" from /etc/passwd :

```
~ $ cat /etc/passwd | awk 'BEGIN{FS=":"; OFS="\t"} /^ic/ {
    print $1, $3}' > users.txt

```

**Outcome of awk command**

```
ictptutor 1000
ictpuser 1001
```

## Linux - Shell Tools: Awk Practice

AWK can take let you do some arithmetic
awk 'BEGINFS=":"; OFS="-" selection_criteria {action}' input-file ¿ output
Eg. To divide all values of column 1 by 2.0 :

```
~ $ awk '{ print $1/2.0 }' data_output.dat

```

### Outcome of awk command

2
2
2
2
...
...

## Linux - Shell Tools: Awk Practice

AWK can accept logicals and conditional statements
awk 'BEGINFS=":"; OFS="-" selection_criteria {action}' input-file > output
Eg. To extract running processes with bash names starting with "ic" from /etc/passwd :

```
~ $ ps -ef | awk '{ if($NF == "bash") print $0 }'

```

**Outcome of awk command**

ictpuser 9331 9323 0 Oct25 pts/0 00:00:01 bash

## Linux - Shell Tools: SED

### SED

SED (Stream Editor) is a compact programming language for parsing and transforming text.

- Line Stream matching and extraction
- input is file
- Supports regular expressions
- Supports piping

### Syntax & Structure

sed [options] [SCRIPT] input-file > output

## Linux - Shell Tools: SED Practice

Eg. To replace the string 'Kinetic' in data_output.dat to 'Total'

```
~ $ sed s/Kinetic/Total/ data_output.dat

```

Outcome of awk command

```
# This file was created Tue Oct 11 15:42:37 2022
# Created by:
#Step "#Potential" "#Total"
0.000000 1635.648926 331729.281250
1.000000 -10321.562500 347803.593750
2.000000 -18997.654297 370155.781250
3.000000 -24159.796875 398618.187500
......
```

## Linux - Advance Shell Tools & Programs

End of Advanced Shell Tools & Programs

## Linux - Introduction to Text editing and Shell Scripting

### Text Editing and Shell Scripting

Introduction to Shell Scripting

1. Editing with Linux text editors
   1. Nano
   2. Vi or Vim
   3. Emacs
2. Bash Shell Scripting

# Linux - Introduction to Text editing and Shell Scripting

## Nano Syntax & Structure

To starting text editing with Nano:

```
~ $ nano <file-name>

```

After adding text content to the file

## Editing operations of Nano

ctrl + O - Write to file(save changes made)
ctrl + X - Close the opened file
ctrl + G - Get help with Nano
ctrl + W - Search or find a string in text
...
...

## Linux - Introduction to Text editing and Shell Scripting

### Vim Syntax & Structure

To starting text editing with Vi or Vim:

```
~ $ vim <file-name>

```

Def: Escape mode

### Modes of Vi/Vim

- Escape mode - esc key
- INSERT mode - i key
- VISUAL Block mode - ctrl + v

## Linux - Introduction to Text editing and Shell Scripting

### Vim Syntax & Structure

To starting text editing with Vi or Vim:

```
~ $ vim <file-name>

```

After adding text content to the file, get into ESC mode

### Editing operations of Vim

w - Write to file(save changes made)
q - quit vim of close the opened file

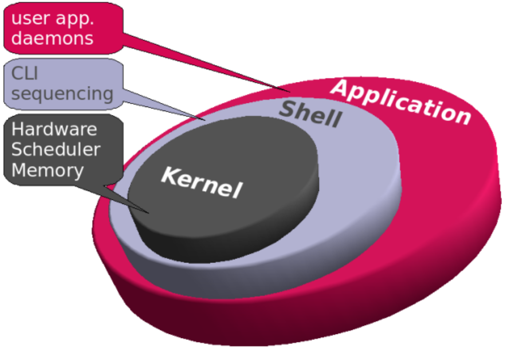
## Shell Scripting

Introduction to Shell Scripting

1. Shell Scripting

## Linux - Introduction to Text editing and Shell Scripting

Computer Structure



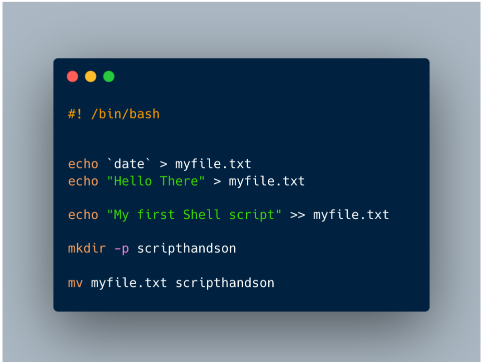
Ref to image: Kernel & Shell.

- Shells :
  - Borne Shell
  - Borne-Again Shell(Bash)
  - korn shell
  - C shell
  - ...
  - ...

# Linux - Introduction to Text editing and Shell Scripting

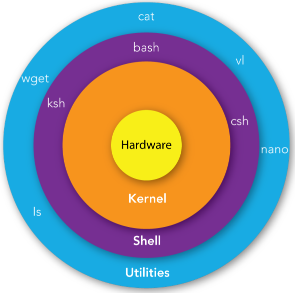
Shell scripts & The Computer Structure.

Why shell scripts look like.



```
#! /bin/bash

echo `date` > myfile.txt
echo "Hello There" > myfile.txt

echo "My first Shell script" >> myfile.txt

mkdir -p scripthandson

mv myfile.txt scripthandson
```

Computer Structure.



Ref to image: Kernel & Shell.

## Why is shell scripting even necessary ?

- Importance:
  - Writing a series of commands
  - Combine lengthy and repetitive commands
  - Execute Routine task

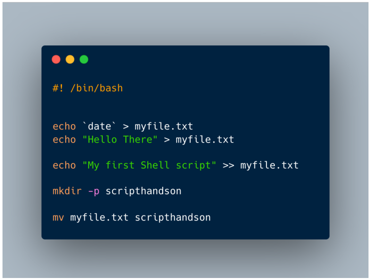
    ...

    ...

# Linux - Introduction to Text editing and Shell Scripting

How to create a shell script

- Steps:
  1. Create a file(with your preferred text editor) and name it with a .sh extension.
  2. Start the content of the script with #!(shebang) /path/to/shell/.
  3. Add some code/text/content to the file/script and save.
  4. Modify file permissions of script to make it executable .



```
#! /bin/bash

echo `date` > myfile.txt
echo "Hello There" > myfile.txt

echo "My first Shell script" >> myfile.txt

mkdir -p scripthandson

mv myfile.txt scripthandson
```

## Linux Command Line - Shell Scripting & Access Control

**chown:**: Change ownership of files

**chmod**: Change permission on files

**setuid**: Share ownership on files

**sticky bit**: Share write access on a directory

# Linux - Introduction to Text editing and Shell Scripting

## Making file executable

To change the permission to make file executable by user:

```
~ $ chmod u+a <script-name.sh>

```

## Running executable script

To run or execute script:

```
~ $ ./<script-name.sh>

```

or

```
~ $ bash <script-name.sh>
```

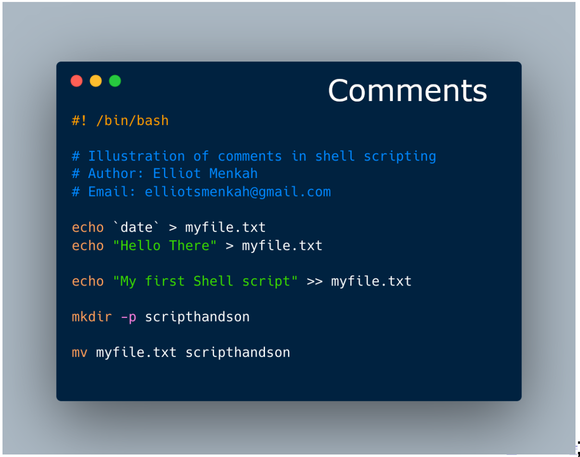
## Linux Command Line - Shell Scripting & Access Control

- Comments
- Variables
- Statements
- Conditionals
- Controls sequence/ Loops
- Functions

## Linux - Introduction to Text editing and Shell Scripting

### Comments in Scripting

Comments in shell scripting are denoted with a preceding # symbol.

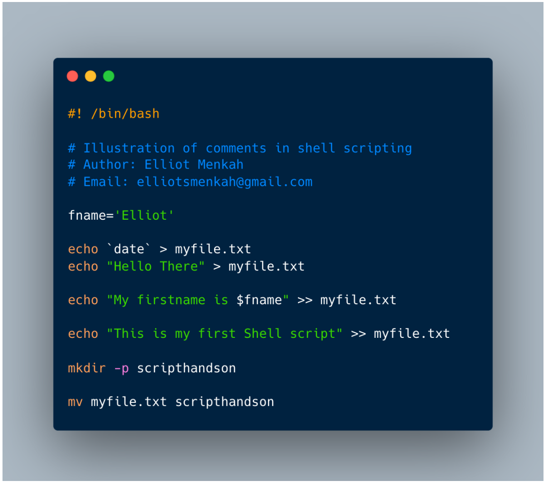


```
#! /bin/bash

# Illustration of comments in shell scripting
# Author: Elliot Menkah
# Email: elliotsmenkah@gmail.com

echo `date` > myfile.txt
echo "Hello There" > myfile.txt

echo "My first Shell script" >> myfile.txt

mkdir -p scripthandson

mv myfile.txt scripthandson
```

## Linux - Introduction to Text editing and Shell Scripting

### Shell Variables

Shell Variables store data.



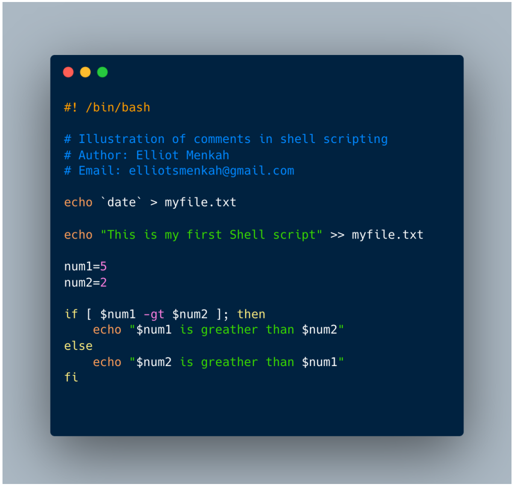
```
#! /bin/bash

# Illustration of comments in shell scripting
# Author: Elliot Menkah
# Email: elliotsmenkah@gmail.com

fname='Elliot'

echo `date` > myfile.txt
echo "Hello There" > myfile.txt

echo "My firstname is $fname" >> myfile.txt

echo "This is my first Shell script" >> myfile.txt

mkdir -p scripthandson

mv myfile.txt scripthandson
```

;

# Linux - Introduction to Text editing and Shell Scripting

## Conditionals

Conditionals are tools for decision making.



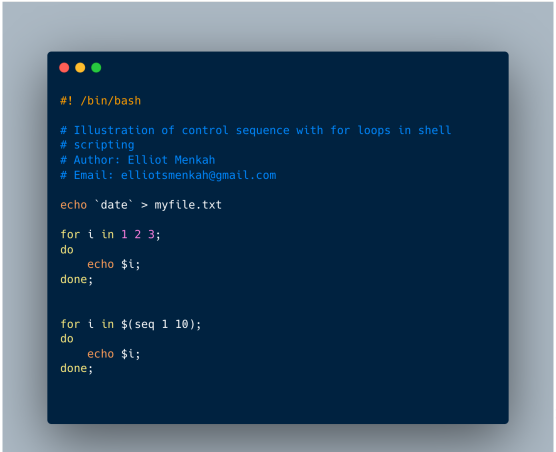
```
#! /bin/bash

# Illustration of comments in shell scripting
# Author: Elliot Menkah
# Email: elliotsmenkah@gmail.com

echo `date` > myfile.txt

echo "This is my first Shell script" >> myfile.txt

num1=5
num2=2

if [ $num1 -gt $num2 ]; then
    echo "$num1 is greather than $num2"
else
    echo "$num2 is greather than $num1"
fi
```

;

# Linux - Introduction to Text editing and Shell Scripting

## Control Sequence/ Loops

Control Sequence or loops are used to iteratively parse instructions to be executed.

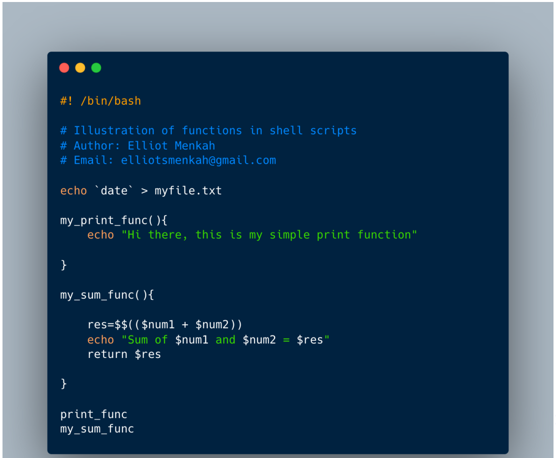


```
#! /bin/bash

# Illustration of control sequence with for loops in shell
# scripting
# Author: Elliot Menkah
# Email: elliotsmenkah@gmail.com

echo `date` > myfile.txt

for i in 1 2 3;
do
    echo $i;
done;


for i in $(seq 1 10);
do
    echo $i;
done;
```

## Linux - Introduction to Text editing and Shell Scripting

### Functions

A functions is a way or technique for grouping reusable bits of code under one name for later use.



```
#! /bin/bash

# Illustration of functions in shell scripts
# Author: Elliot Menkah
# Email: elliotsmenkah@gmail.com

echo `date` > myfile.txt

my_print_func(){
    echo "Hi there, this is my simple print function"

}

my_sum_func(){

    res=$$(($num1 + $num2))
    echo "Sum of $num1 and $num2 = $res"
    return $res

}

print_func
my_sum_func
```

## Linux - Introduction to Plotting with GNUPLOT

Plotting with GNUPLOT

Introduction to Plotting with GNUPLOT