PH434 Autumn 2025 – Class # 2

Working on a terminal

(Dated: 5 August 2025)

I. WORKING WITH BASH II – SCRIPTS

In the previous section, we learned how bash commmands can be used to execute certain tasks via the terminal. An important point here is that each command was typed in sequentially and executed from one to the next. However, it is also possible to bunch a set of commands in a script (written in some text editor) and executing the script file on the terminal.

In this section we look at some simple examples of bash scripts that can be run or executed from the terminal. In the last notes, we created a file called "rhapsody.txt" in a folder called "Newfolder". For example:

```
username@comp:~$ pwd
/home/username
username@comp:~$ ls Newfolder/
rhapsody.txt
```

Suppose we want to now create a new folder called "Check" inside Newfolder/ and copy the "rhapsody.txt" to the folder, and display its contents. Instead of writing all this down as individual commands, we can write a script.

```
username@comp:~$ cat > trial.sh
#! /bin/bash
mkdir Newfolder/Check
cp  Newfolder/rhapsody.txt   Newfolder/Check/
cd  Newfolder/Check
cat rhapsody.txt (Ctrl+D) (green lines are input by the user)
```

Now we need to execute the script file "trial.sh":

```
username@comp:~$ chmod +x trial.sh  (make the script file executable)
username@comp:~$ ./trial.sh  (execute the script)
Is this the real life?
Is this just fantasy?
Caught in a landslide,
No escape from reality.
username@comp:~$ ls  Newfolder/Check/
rhapsody.txt
```

Let us create a script file called “out.sh” to understand variables:

```
#!  /bin/bash
# Initializing a variable  (anything after # is a comment)
vartext="Physics"  (no space between variable and assignment)
varnum=2022
echo "This is the " ${vartext} "batch of" ${varnum}
username@comp:~$ chmod +x out.sh
username@comp:~$ ./out.sh
This is the Physics batch of 2022
```

Alternatively, one can have:

```
#!  /bin/bash
# Initializing a variable  (anything after # is a comment)
vartext="Physics"
varnum=2022
greet="This is the $vartext batch of $varnum"
echo $greet
```

Let us create another script file called “info.sh” that takes user input:

```
username@comp:~$ cat > info.sh
#!  /bin/bash
echo "What is your name?"
read name
echo "Which department are you in?"
read dept
echo "*********************************"
echo "Welcome to the $dept department, $name"
```

```
username@comp:~$ chmod +x info.sh
username@comp:~$ ./info.sh
What is your name?
Saina
Which department are you in?
Physics
***********************************
Welcome to the Physics department, Saina
```

Let us create a script file called “mult.sh” to multiply two numbers:

```
username@comp:~$ cat > mult.sh
#! /bin/bash
num0=2
num1=3
prod=$((num0 * num1))
echo "Multiplying $num0 with $num1 is equal to $prod"
username@comp:~$ chmod +x mult.sh
username@comp:~$ ./mult.sh
Multiplying 2 with 3 is equal to 6
```

Let us create a script file called “file.sh” to copy data into a file:

```
username@comp:~$ cat > file.sh
#! /bin/bash
name="Uma"
add="SH 34"
data = "$name, $add"
echo $data >> file.txt
username@comp:~$ chmod +x file.sh
username@comp:~$ ./file.sh
username@comp:~$ cat file.txt
Uma, SH 34
```

II. WRITING SIMPLE PYTHON SCRIPTS

Similar to bash scripts, we can now write simple python scripts and execute them on the terminal. Let us consider a code to add two numbers using Python (note other alternatives include using IPython or notebook)

Let us create a python script file called "add.py" to add two numbers:

```
username@comp:~$ cat > add.py  (create a python file)
a = 2
b = 3
print (a+b) (# Ctrl + D to save the file)
username@comp:~$ python3 add.py  (command to execute the python file)
5
```

With user input we have:

```
username@comp:~$ cat > add.py
a = input("enter a number:\n") (# by default all inputs are string)
b = 3
a = int(a) (# convert string to integer)
print (a+b)
username@comp:~$ python3 add.py
enter a number:
57 (User input)
60
```

Creating a simple text file:

```
username@comp:~$ cat > text.py
a = "Thierry"
b = "Henry"
data = str(a + " "+ b )
file = open("pytext.txt","a") (# option "a" is for append, "w" write or overwrite)
file.write(data+ "\n")
file.close()
username@comp:~$ python3 text.py
```

```
username@comp:~$ cat pytext.txt
Thierry Henry
username@comp:~$ python3 text.py  (# If you run it again, it appends the text)
username@comp:~$ cat pytext.txt
Thierry Henry
Thierry Henry
```

III. EDITING A SCRIPT – WORKING WITH VIM, NANO AND GEDIT

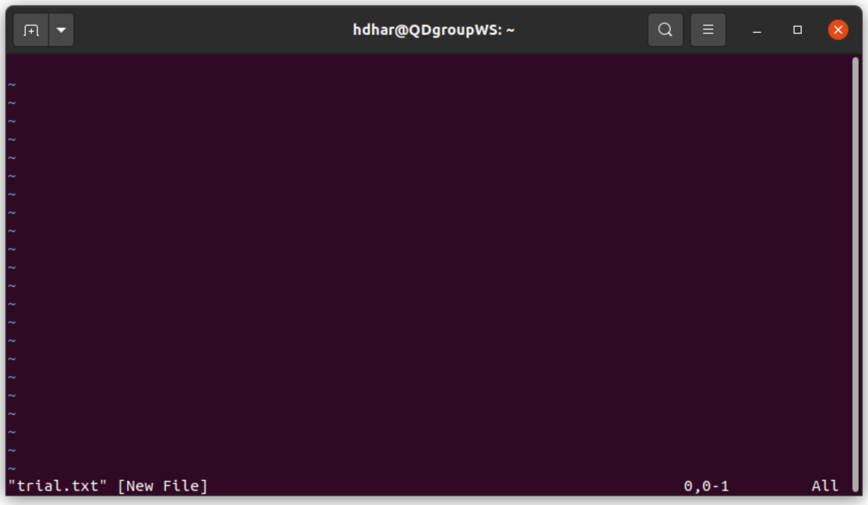
An important aspect of working with bash and python scripts is editing. Once you have created a file using the touch or cat command (and added some text), you can edit the text using one of the above text editors.

- **vim** – *vim* is a screen based text editor and is designed for use as both a command line interface, as well as a standalone application in a GUI. It is one of the most popular text editors among programmers.

  Creating a file on vim:

  ```
  username@comp:~$ vim trial.txt
  ```

  will create a file called 'trial.txt' and open the vim interface (see the figure below).



```
"trial.txt" [New File]                                  0,0-1         All
```

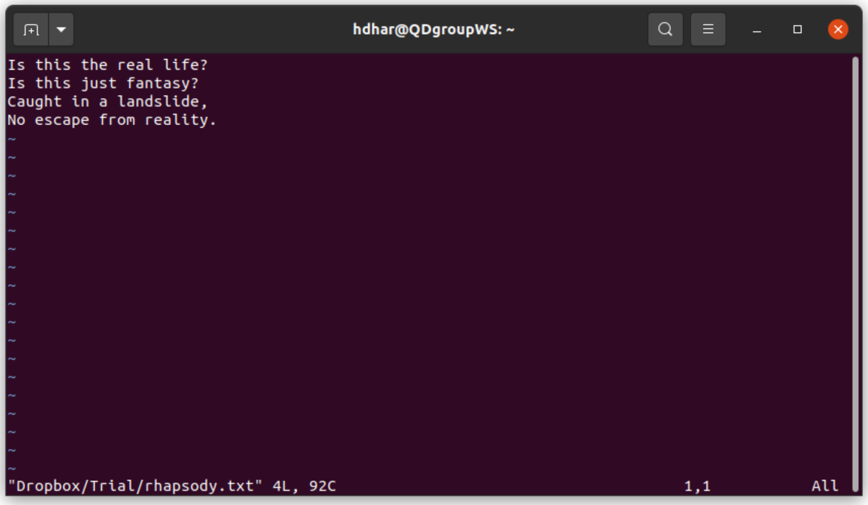
FIG. 1. A snapshot of the vim interface showing a new unsaved file.

The file can be closed with saving by pressing **Esc**, followed by **q** (which implies quit). The interface closes without saving the file 'trial.txt.'

One can also open an existing file, say 'rhapsody.txt', using the command:

username@comp:∼$ vim Dropbox/Trial/rhapsody.txt

which opens the following interface.



```
Is this the real life?
Is this just fantasy?
Caught in a landslide,
No escape from reality.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"Dropbox/Trial/rhapsody.txt" 4L, 92C                        1,1           All
```

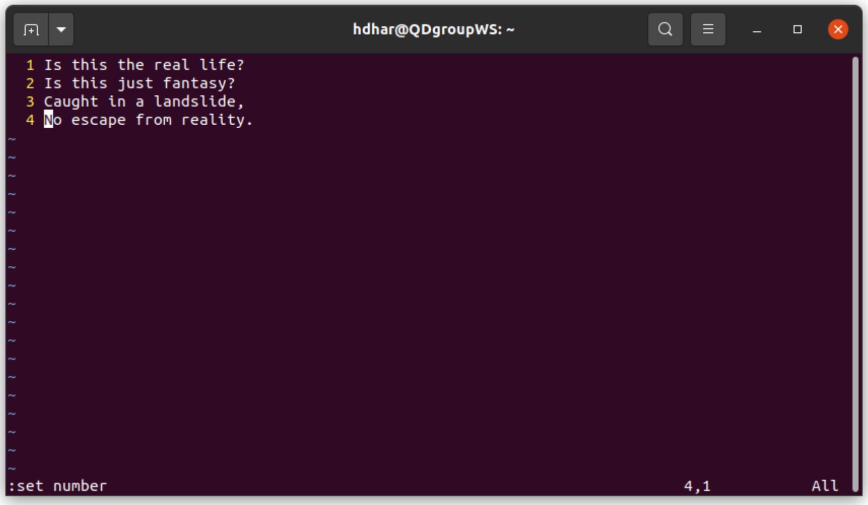
FIG. 2. A snapshot of the vim interface showing an existing text file.

To edit the text, press i (which takes you to insert mode), and move the cursor to edit the relevant parts of the text. Once it is done, press **Esc**, followed by **wq** (implies write and quit) to save the changes. The above command will also save a new file. If you want to quit without saving the changes, press **Esc**, followed by **q!** (force quit).

Note that **Esc** key takes you from insert mode to the normal mode and you will see a colon (' : ', call it prompt) at the bottom line before you type **wq** or **q!**.

Now pressing **Esc**, followed by **3** at prompt, takes you to line 3. Pressing **$** takes you to the last line. In the normal mode, simply pressing **$** takes you to the end of a line, and pressing **0** to the beginning of a line.

In fact, you can set the line number by pressing **Esc**, followed by **set number**. See Figure 3.



```
1 Is this the real life?
2 Is this just fantasy?
3 Caught in a landslide,
4 No escape from reality.
:set number                                                   4,1           All
```

FIG. 3. A snapshot of the vim interface showing line numbers on a text file.

You can also delete, undo, copy, paste and perform other interesting things while editing a file on Vim. See the following webpage [1].

- **nano** – GNU *nano* is another text editor for Unix-like operating systems that uses a command line interface.

  Creating a file in nano:

  username@comp:∼$ nano trial.txt

  will create a file called 'trial.txt' and open the nano interface (see Fig. 4). Pressing **Ctrl + O**, gives you the option to save the file with the above (or different) filename. Most editing options are given in the bottom menu of the interface.

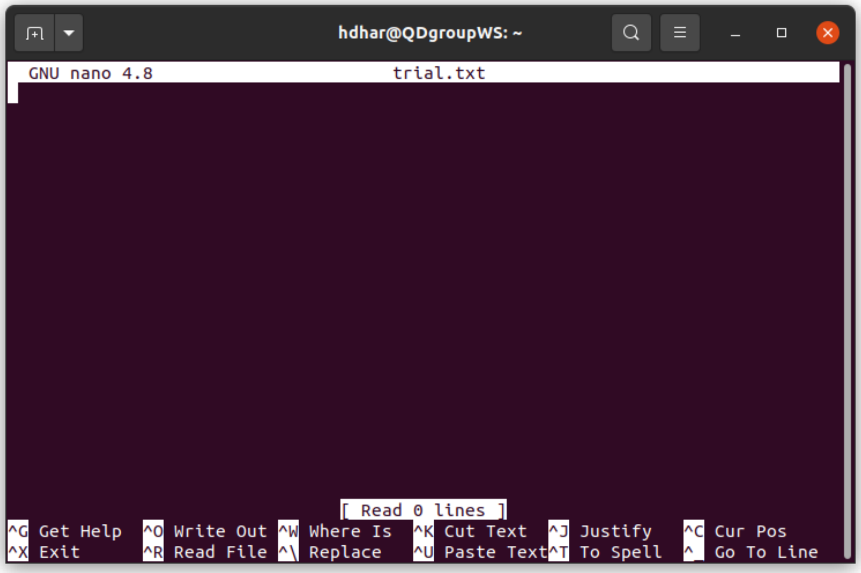
  Similar to vim, one can open an existing file and edit it using nano.

  username@comp:∼$ nano Dropbox/Trial/rhapsody.txt

  This opens the interface below:

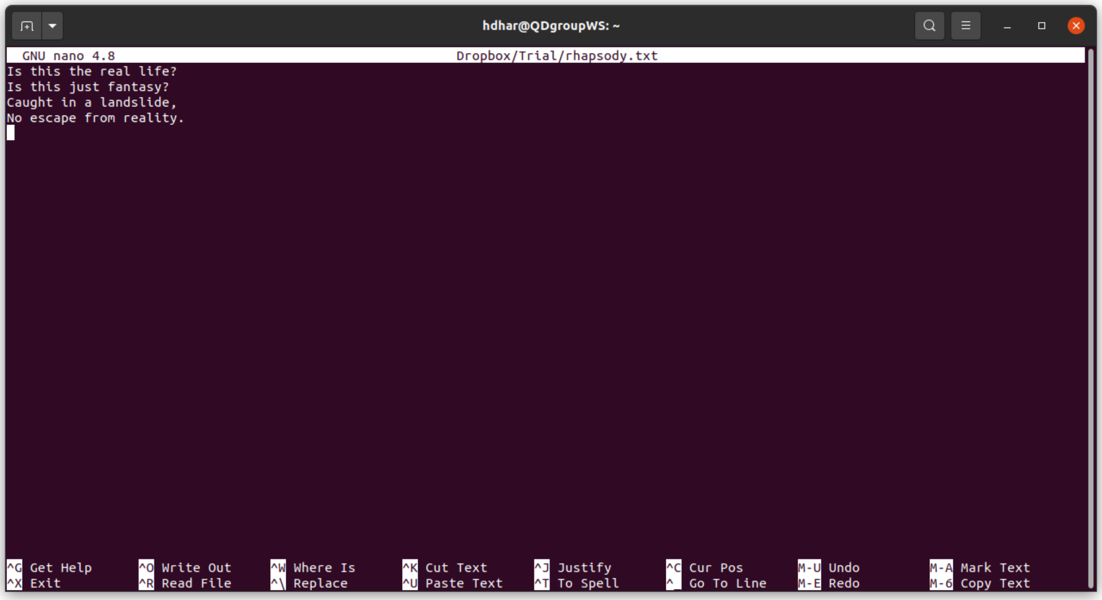
  The bottom menu shows that **Ctrl + X** is to exit nano, **Ctrl + K** is to cut text, and **Ctrl + U** is to paste text. Other options include **Alt + U** for undo and **Alt + E** for redo.

  Nano is a much simpler and user-friendly interface with no modes as in vim, and is more of a WYSIWYG(What You See Is What You Get) text editor. This allows users to directly interact with the text much more easily.



```
hdhar@QDgroupWS: ~
  GNU nano 4.8                          trial.txt




                                   [ Read 0 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line
```

FIG. 4. A snapshot of the nano interface along with editing menu at bottom.

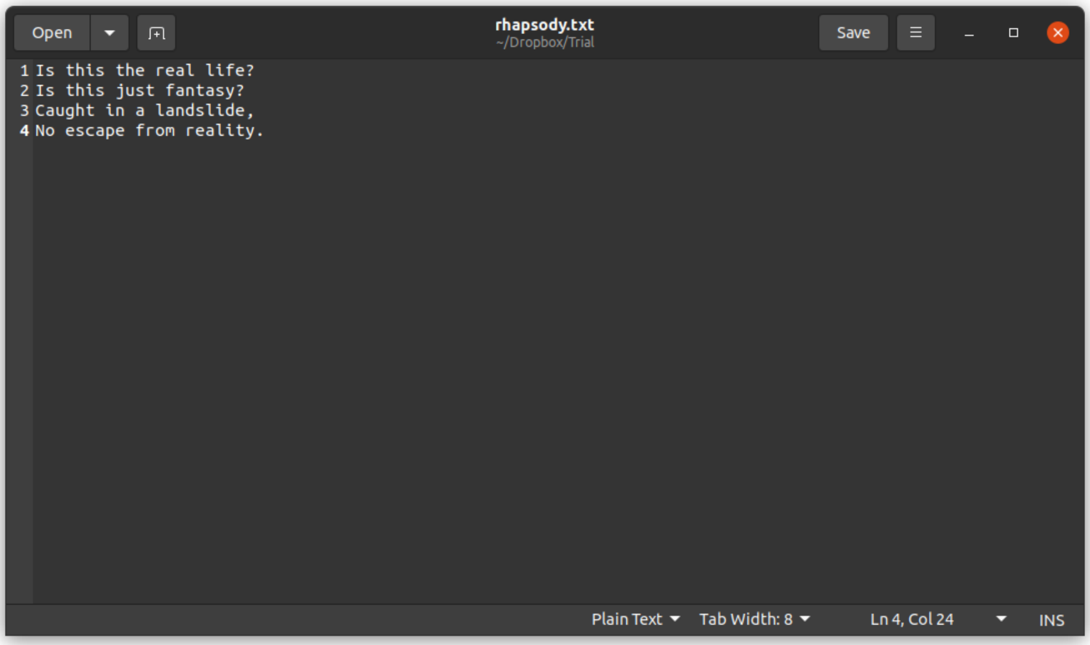


```
hdhar@QDgroupWS: ~
  GNU nano 4.8                          Dropbox/Trial/rhapsody.txt
Is this the real life?
Is this just fantasy?
Caught in a landslide,
No escape from reality.



^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text
```

FIG. 5. A snapshot of the nano interface opening an existing file.

To learn more about nano, see the webpage [2].

- **gedit** – (From Wikipedia) *gedit* is the default text editor of the GNOME desktop environment and part of the GNOME Core Applications. Designed as a general-purpose text editor, gedit emphasizes simplicity and ease of use, with a clean and simple GUI, according to the philosophy of the GNOME project. It includes tools for editing source code and structured text such as markup languages. The text editor works very close to a basic word processor.



```
Is this the real life?
Is this just fantasy?
Caught in a landslide,
No escape from reality.
```

FIG. 6. A snapshot of the gedit interface opening an existing file.

So one can simply open gedit, from the terminal or by clicking a text file in file manager, and directly edit the file (similar to how one would use Word).

IV. TASKS FOR THE DAY

1. Work out the commands and scripts in bash and script discussed in the notes till you are comfortable with each one. Go online and read about more such commands. For the time being, try to avoid commands that are complex and do not follow simple logic.

2. Write a code in bash script and python that takes two values from the user say x and y and solves the equation: $f(x,y) = x^2 + 5y$. Save the output in a text file in the form $\{“x = ”, x, “y = ”, y, “f(x,y) = ”, f(x,y)\}$ for about 10 values of x and y.

3. Write a code in a) bash script and b) python that creates a file to *input* the names of

people and their birthdays (in ddmmyyyy format) and save it in a text file. Run the code 10 times such that the names and birthday of 10 of your friends is saved in the folder.

3. This is for those who are up for a more challenging task. There is gameshow called *Khulja Sim Sim* with three doors, one of which contains a brand new car and the other two doors have grass. It is unknown to the player what is behind any particular door. The player has to choose a door. Depending on your choice, the host opens a different door which is empty (leaving only two doors closed). He now asks you whether you want to stick to your original choice, or do you want to change to the other.

   a) Write a python code in the terminal to simulate the game using simple commands, functions and loops.

   b) Calculate the probability, which tells you what is the best option for the player to win the car – should they stay with their original choice or change to the other door?

   The commands below allow you to create random integers between 0 and 2 (for the three doors).

   ```
   import random
   num = random.randint(0,2)
   ```

---

[1] Please see the link for more details on vim.

[2] Please see the link for more details on nano.