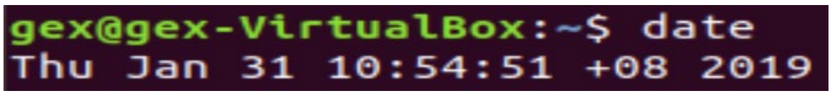
Ashlyn Goh 1002840 CI02

2.1.1 Commands Without Options/Arguments

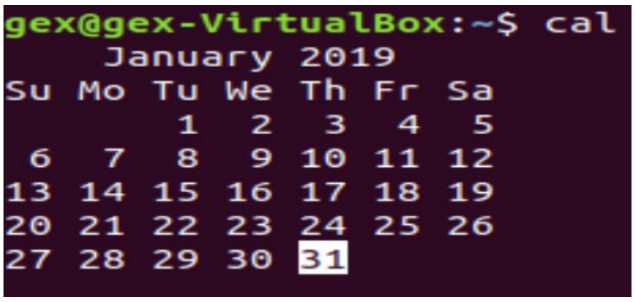
1. [1m] Type date and press enter. Record your answer.



```
gex@gex-VirtualBox:~$ date
Thu Jan 31 10:54:51 +08 2019
```

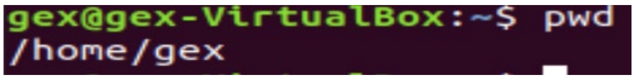
2. **[4m]** Do the same thing with **cal**, **pwd**, **ls**, **who**. Record your answer.

cal:



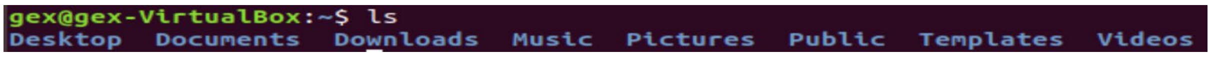
```
gex@gex-VirtualBox:~$ cal
    January 2019
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

pwd:



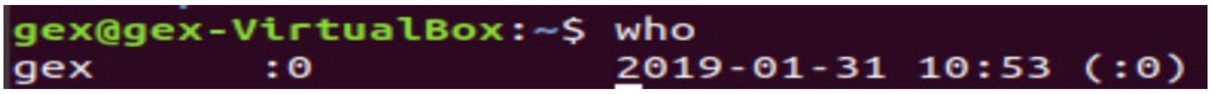
```
gex@gex-VirtualBox:~$ pwd
/home/gex
```

ls:



```
gex@gex-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

who:



```
gex@gex-VirtualBox:~$ who
gex      :0           2019-01-31 10:53 (:0)
```

3. **[5m]** Based on your observation, what's the purpose of each of the five commands above?

date – Print out current date

cal – Print calendar

pwd – Print name of current/working directory

ls – Lists files in current directory

who – Lists who is logged on your machine

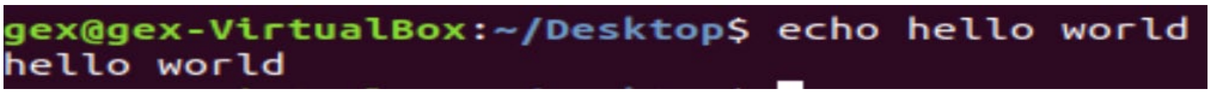
2.1.2 Commands With Options / Arguments

1. **[1m]** The command **cd** changes your current working directory. Type **cd /path** where **/path** is the path to your desktop, and press enter. Has your current directory change? *Hint: you can confirm this by typing* ls + enter *and you will see that the output is the lists of files in your desktop*.

Yes, the directory changed.

Ashlyn Goh   1002840   CI02

2. **[1m]** Type **echo hello world**. Record the output. What does this command do?



```
gex@gex-VirtualBox:~/Desktop$ echo hello world
hello world
```

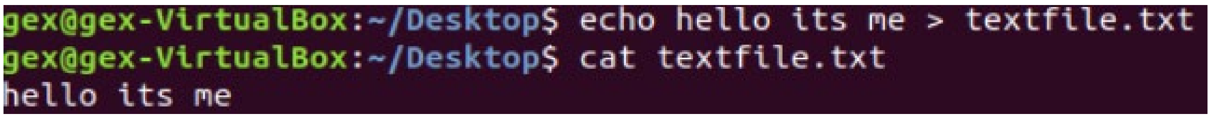
The command outputs (prints) out the given argument, in this case “hello world”, to standard output.

2.2.4 I/O Redirection

3. [1m] What new file do you see?

textfile.txt is seen.

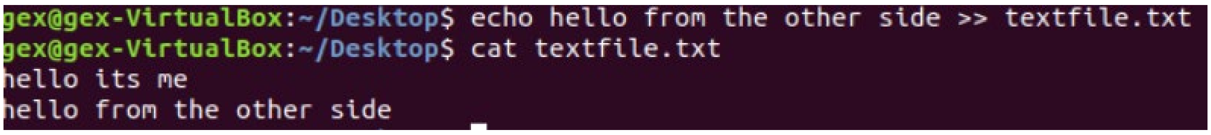
5. [1m] cat textfile.txt. What is the content of textfile.txt?



```
gex@gex-VirtualBox:~/Desktop$ echo hello its me > textfile.txt
gex@gex-VirtualBox:~/Desktop$ cat textfile.txt
hello its me
```

Content of textfile.txt is “hello its me”

8. **[1m] cat textfile.txt**. What is the content of textfile.txt?



```
gex@gex-VirtualBox:~/Desktop$ echo hello from the other side >> textfile.txt
gex@gex-VirtualBox:~/Desktop$ cat textfile.txt
hello its me
hello from the other side
```

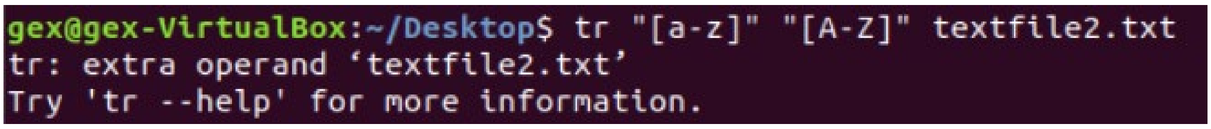
Content of textfile.txt is:
hello its me
hello from the other side

9. [1m] What does >> do differently than >?

> redirects output to a file, overwriting the file.

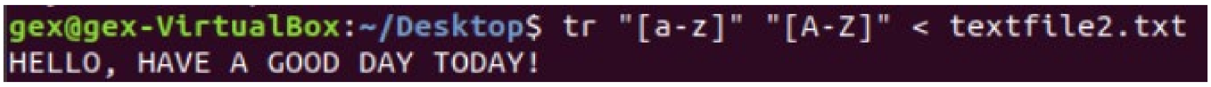
>> redirects output to a file appending the redirected output at the end.

3. **[1m]** What is the output?



```
gex@gex-VirtualBox:~/Desktop$ tr "[a-z]" "[A-Z]" textfile2.txt
tr: extra operand ‘textfile2.txt’
Try 'tr --help' for more information.
```

4. **[1m]** Now try tr "[a-z]" "[A-Z]" < yourfilename.txt. What is the output?



```
gex@gex-VirtualBox:~/Desktop$ tr "[a-z]" "[A-Z]" < textfile2.txt
HELLO, HAVE A GOOD DAY TODAY!
```

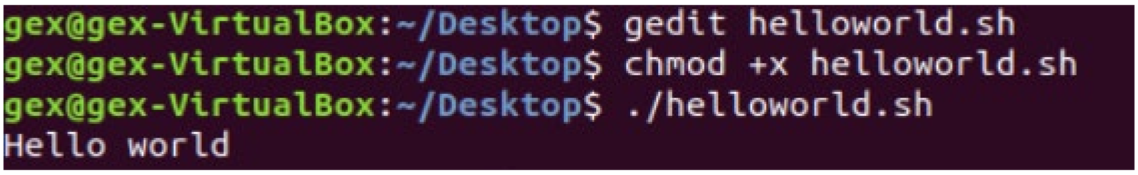
Ashlyn Goh 1002840 CI02

5. **[1m]** What is the difference between tr "[a-z]" "[A-Z]" < yourfilename.txt and tr "[a-z]" "[A-Z]" yourfilename.txt?

The one with "<" works while the other does not.
It is missing the "<" operator which uses the content of the file as an input to the tr command (the stdin is redirected).

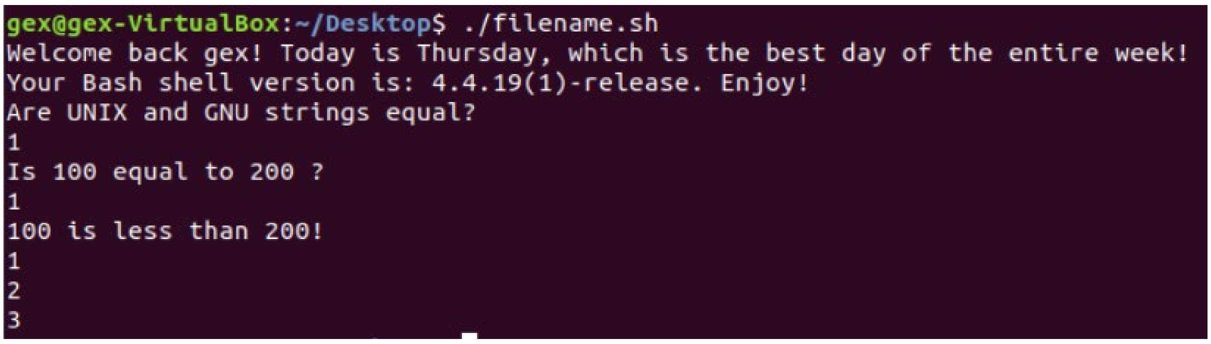
3 Bash Scripting [2m]

5. [1m] Record your answer



```
gex@gex-VirtualBox:~/Desktop$ gedit helloworld.sh
gex@gex-VirtualBox:~/Desktop$ chmod +x helloworld.sh
gex@gex-VirtualBox:~/Desktop$ ./helloworld.sh
Hello world
```

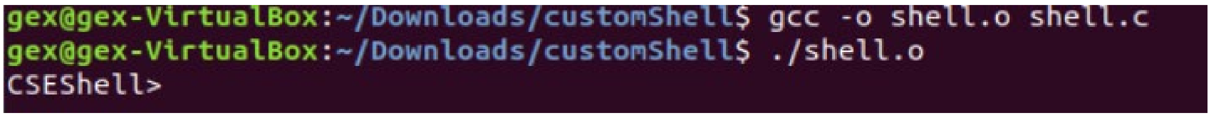
3. **[1m]** Write down the output that you see.



```
gex@gex-VirtualBox:~/Desktop$ ./filename.sh
Welcome back gex! Today is Thursday, which is the best day of the entire week!
Your Bash shell version is: 4.4.19(1)-release. Enjoy!
Are UNIX and GNU strings equal?
1
Is 100 equal to 200 ?
1
100 is less than 200!
1
2
3
```

4 Making your own Shell

3. **[1m]** What do you see on your screen? Copy and paste it to your answer document.



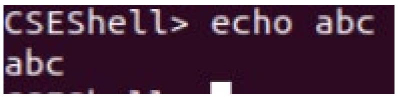
```
gex@gex-VirtualBox:~/Downloads/customShell$ gcc -o shell.o shell.c
gex@gex-VirtualBox:~/Downloads/customShell$ ./shell.o
CSEShell>
```

1. **[1m]** What is the next function called after main?

The next function called is shellLoop().

Ashlyn Goh 1002840 CI02

2. **[2m]** Type echo abc. What do you see? Where do you think the argument abc is stored? args[0] or args[1]?



```
CSEShell> echo abc
abc
```

args[1]

3. **[1m]** Trace the function calls. What is the appropriate return value to stop the program from running? 1 or 0?

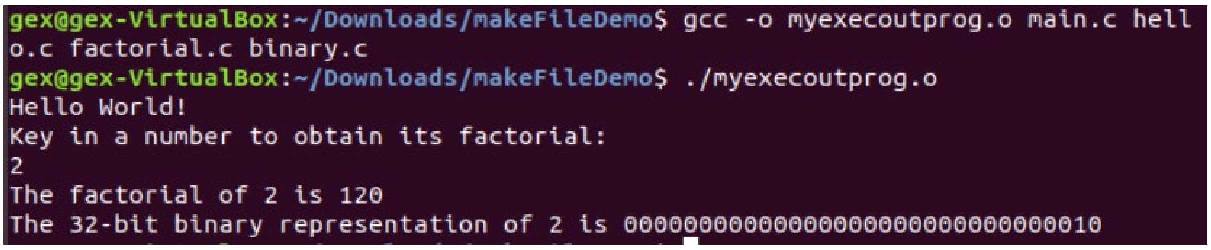
To stop the program from running, return 0.

4. **[1m]** Look at the method shellExecWithExecvp. You notice that there's a function called fork(). Go online and find out what it does. Write your answer in not more than 2 sentences.

System call fork() creates a new process, which becomes the child process of the caller.

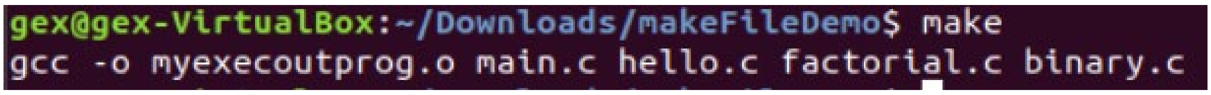
5 Unix Makefile [9m]

3. **[1m]** What is the output?



```
gex@gex-VirtualBox:~/Downloads/makeFileDemo$ gcc -o myexecoutprog.o main.c hell
o.c factorial.c binary.c
gex@gex-VirtualBox:~/Downloads/makeFileDemo$ ./myexecoutprog.o
Hello World!
Key in a number to obtain its factorial:
2
The factorial of 2 is 120
The 32-bit binary representation of 2 is 00000000000000000000000000000010
```

2. **[1m]** What is the output?



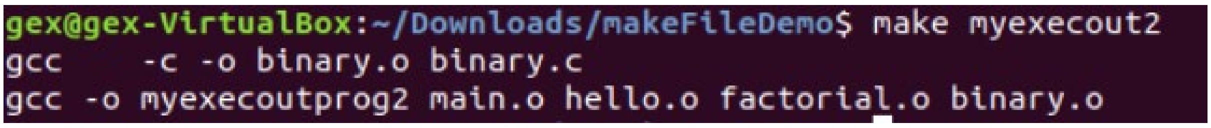
```
gex@gex-VirtualBox:~/Downloads/makeFileDemo$ make
gcc -o myexecoutprog.o main.c hello.c factorial.c binary.c
```

Ashlyn Goh 1002840 CI02

5.1 Recompiling

Q. Now open binary.c in text editor and add another function in it. You can write any function you want, the point is just to modify the file. Save your changes, and run make myexecout2. [1m] What is the output? [1m] Is it the same as above? [1m] Why?

The output is:

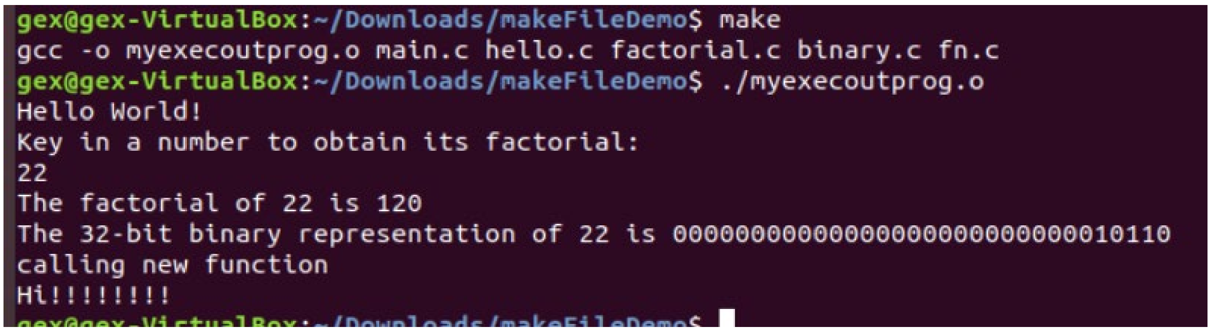


```
gex@gex-VirtualBox:~/Downloads/makeFileDemo$ make myexecout2
gcc    -c -o binary.o binary.c
gcc -o myexecoutprog2 main.o hello.o factorial.o binary.o
```

It is not the same as the given output from before.
This is because since binary.c was edited, only files that are dependent on binary.c are recompiled. Hence, the output is different from before.

Q. [1m] Which file you should change in order to force gcc to recompile everything?
The functions.h file. (Since it is where every other files are dependent on)

Now, create another .c file containing the implementation of any function that you want that can print something on the terminal. Add the function to **functions.h**, and call it from **main.c**. **[3m]** Modify the makefile to include this new .c file during compilation. Zip all the .c, .h, and the makefile.



```
gex@gex-VirtualBox:~/Downloads/makeFileDemo$ make
gcc -o myexecoutprog.o main.c hello.c factorial.c binary.c fn.c
gex@gex-VirtualBox:~/Downloads/makeFileDemo$ ./myexecoutprog.o
Hello World!
Key in a number to obtain its factorial:
22
The factorial of 22 is 120
The 32-bit binary representation of 22 is 00000000000000000000000000010110
calling new function
Hi!!!!!!!!
gex@gex-VirtualBox:~/Downloads/makeFileDemo$
```

The modified files are attached.