CENG 322
LAB 1

Introduction to Linux and Shell Commands

Road Map

- Introduction
- Shell Commands
- Exercises - Part 1
- Shell Scripts
- Exercises - Part 2

Unix and Linux

- 1970s
- Ken Thompson
- Dennis Ritchie

IBM hp UX

- 1990s
- Linus Torvalds

debian redhat

Unix and Linux

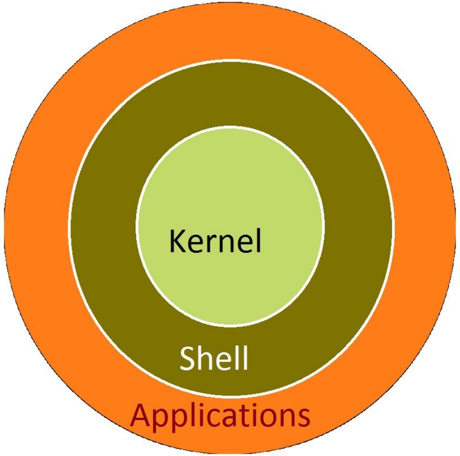
- **Ubuntu** is a member of this family (it is a “**Linux distro**” (distribution)).
- Ubuntu is **the most popular** Linux distribution.
- It is based on **Debian**.
- You can and should install it on your computer using a **USB stick** (you can use Ubuntu **along with your existing operating system**).
- If you have no experience installing an OS, **do not forget to backup** your data!
- You can try Ubuntu using a “**Live USB**”.

Terminal and Shell

- A **Shell** is a program which **processes** user commands and **returns** output (e.g. **Bash** in Linux).
- It communicates with the internal part of the operating system (called the **kernel**).
- A **Terminal** is a program that runs a **Shell** using a text-based user interface.
- "**Command line**" is the Windows-centric word for the "terminal".

BASH

THE BOURNE-AGAIN SHELL



Some of the most important Linux directories

- **/** is the “root directory” (The parent of every directory).

  (This is distinct from **/root** which is the “home directory” of the “root user” (=“superuser”).)

Some of the most important Linux directories

- **/** is the “root directory” (The parent of every directory).

  (This is distinct from **/root** which is the “home directory” of the “root user” (=“superuser”).)

- **/home** contains “home directories” of the users (e.g. **/home/altug**).

  (A home directory contains user’s data files and user-specific configuration files.)

  (Home directory of the “root user” is an exception: **/root**)

Some of the most important Linux directories

- **/** is the “root directory” (The parent of every directory).

  (This is distinct from **/root** which is the “home directory” of the “root user” (=“superuser”).)
- **/home** contains “home directories” of the users (e.g. **/home/altug**).

  (A home directory contains user’s data files and user-specific configuration files.)

  (Home directory of the “root user” is an exception: **/root**)
- **/etc** contains system-wide configuration files.

  (They can generally be edited by hand in a text editor.)

Some of the most important Linux directories

- **/** is the "root directory" (The parent of every directory).

  (This is distinct from **/root** which is the "home directory" of the "root user" (="superuser").)
- **/home** contains "home directories" of the users (e.g. **/home/altug**).

  (A home directory contains user's data files and user-specific configuration files.)

  (Home directory of the "root user" is an exception: **/root**)
- **/etc** contains system-wide configuration files.

  (They can generally be edited by hand in a text editor.)
- **/bin** contains important system "binaries" (=programs).

  (It contains user binaries, executable files, Linux commands that are used in single user mode, and common commands that are used by all the users, like cat, cp, cd, ls, etc.) (**/sbin** is the same but with root privileges required.)

Some of the most important Linux directories

- **/** is the “root directory” (The parent of every directory).

  (This is distinct from **/root** which is the “home directory” of the “root user” (=“superuser”).)
- **/home** contains “home directories” of the users (e.g. **/home/altug**).

  (A home directory contains user’s data files and user-specific configuration files.)

  (Home directory of the “root user” is an exception: **/root**)
- **/etc** contains system-wide configuration files.

  (They can generally be edited by hand in a text editor.)
- **/bin** contains important system “binaries” (=programs).

  (It contains user binaries, executable files, Linux commands that are used in single user mode, and common commands that are used by all the users, like cat, cp, cd, ls, etc.) (**/sbin** is the same but with root privileges required.)
- **/lib** contains essential shared libraries.

  (These libraries are referenced from binaries in **/bin** and **/sbin**.)

Some of the most important Linux directories

- **/** is the “root directory” (The parent of every directory).

  (This is distinct from **/root** which is the “home directory” of the “root user” (=“superuser”).)
- **/home** contains “home directories” of the users (e.g. **/home/altug**).

  (A home directory contains user’s data files and user-specific configuration files.)

  (Home directory of the “root user” is an exception: **/root**)
- **/etc** contains system-wide configuration files.

  (They can generally be edited by hand in a text editor.)
- **/bin** contains important system “binaries” (=programs).

  (It contains user binaries, executable files, Linux commands that are used in single user mode, and common commands that are used by all the users, like cat, cp, cd, ls, etc.) (**/sbin** is the same but with root privileges required.)
- **/lib** contains essential shared libraries.

  (These libraries are referenced from binaries in **/bin** and **/sbin**.)
- **/usr** contains user binaries and read-only data.

  (**/usr/bin**, **/usr/sbin** are similar to **/bin** and **/sbin**. But these are used by the users and not by the system.)

  (Similar to **/lib** for **/bin** and **/sbin**, there is also **/usr/lib** for **/usr/bin** and **/usr/sbin**.)

Shell Commands

Shell Commands

- Shell commands are **case sensitive**.
- Be careful with **spaces**. (e.g. `chmod u=wx a.txt`. You cannot add spaces in other places.) If a value contains a space in it, surround it with quotation marks. (e.g. `cd "directory name"`)
- Some commands require superuser privileges. Use **sudo** for them. `sudo <yourcommand>`
- Open “Terminal”. (In any directory, you can **right click > “Open in Terminal”** to open Terminal and set the selected directory as the working directory)
- Use the **TAB** button to autofill what you are typing. Use **arrow keys** (e.g. up) to see previous commands.
- For getting help, you can try using **command --help** (e.g. `cd --help`) or **man command**

Navigation

- **directory** (=folder) vs. **file** (Note: a.txt ≠ A.txt in Linux)

- **absolute path** (starts with slash) vs. **relative path**
  (You use these paths not only in a terminal but in programming languages as well.)

- **/** (directory separator) e.g. **abc/def/g** (Inside abc, there is def. Inside def, there is g)
  .. (parent directory) e.g. **../../abc/d** (Go two levels up. There is abc. Inside abc, there is d.)
  ~ (home directory) It is **/home/altug** for me. Thus, e.g., **~/x/y** is **/home/altug/x/y**
  . (current directory)

- **`pwd`** (print "working directory", also known as "current directory")
  `cd path` (change working directory to **path** which is either absolute or relative)

What are the absolute paths of the red files & dirs?

- /
  - home
    - Altug
      - Desktop
        - programs
          - x.out
          - y.out
        - things
          - a.png
        - b.txt
      - Downloads
        - files
          - c.pdf
      - Documents
      - Pictures
  - bin

What are the relative paths for red files & dirs? (Blue indicates the working directory.)

- /
    - home
        - Altug
            - Desktop
                - **programs**
                    - x.out
                    - y.out
                - things
                    - a.png
                - b.txt
            - Downloads
                - files
                    - c.pdf
            - Documents
            - Pictures
    - bin

Files

- **touch** is used to create an empty file:

  `touch a.txt`

- **cat** is used to create a file with content:

  `cat a.txt > b.txt` (overwrite)

  `cat a.txt >> b.txt` (append)

  You can merge contents of multiple files:

  `cat a.txt b.txt > c.txt`

  You can enter custom content:

  `cat > a.txt` (Try it. CTRL+D to finish)

- **echo** can be used to write a single line (echo is used for printing):

  `echo "abc" > a.txt` (overwrite)

  `echo "abc" >> a.txt` (append)

Files

Display content

`cat file` (displays entire file)

`less file` (displays one page at a time)

(**return**: one line forward, **space**: one page forward, **y**: one line back, **b**: one page back, **/**: search, **q**: quit)

`head file` (displays first 10 lines. Alternatively: `head -n 50 a.txt`),

`tail file` (similar to **head** but displays the last lines)

`wc file` (displays number of lines, words, and bytes. -l lines, -w words, -c bytes)

Manage files

`rm file` (remove)

`cp file directory` (copy)

`mv file directory` (move)

`mv file1 file2` (rename)

Move (`mv file target_dir`) or rename (`mv file new_file`) a file

(Blue indicates the working directory.)

- /
  - home
    - Altug
      - **Desktop**
        - programs
          - **x.out**
          - y.out
        - things
          - a.png
        - b.txt
      - Downloads
        - files
          - c.pdf
      - Documents
      - Pictures
  - bin

(1) Rename x.out to x1.out

(2) Move x1.out to Downloads

Directories

list directory contents: `ls`

- **-l** long list (displays lots of info)
- **-t** sort by modification time
- **-S** sort by size
- **-r** reverse the order

long list, order by modification time, reverse the order: `ls -ltr`

Using “glob”s (similar to “RegEx”es) and brace expansion:

```
ls a.* (anything)
ls *a*
ls img_*.png
ls users-[0-9][a-zA-Z0-9][0-9]*
```

```
ls [Aa].* (any of these)
ls *.txt
ls img_?.png (single character)
ls *.{jpg,jpeg} (brace expansion)
```

Directories

create: `mkdir directory` (make directory)

What will this command do? `mkdir directory name`

copy: `cp directory1 directory2`

move or rename: `mv directory1 directory2` (if directory2 exists then move inside of it)

remove: `rm -r directory` (remove recursively)

Grep (search text)

`grep pattern file` (search in file; show those lines that match)

`grep pattern file1 file2 file3` (search in all these files)

`grep -i pattern file` (case-insensitive search)

`grep -v pattern file` (show those lines that do not match)

`grep -R pattern .` (search recursively inside the current directory)

See `grep --help`

Permissions

```
ls -l
```

Read **(r),** write **(w),** execute **(x).** In case of directory, “x” is for listing directory contents.

First: user **(u)** (the owner, which is you). **Second**: group **(g)**. **Third**: “the world” (others) **(o)**.

chmod (change the read, write, and execute permissions of files and directories)

`chmod u=rwx,g=rx,o=r file`

`chmod u=rw file`

`chmod u+x file` (to allow executable permissions)

`chmod u-wx file` (to take out write and executable permissions)

Permissions

We can specify 3 permissions using a single integer in the range [0, 7].

read (**4**), write (**2**), execute (**1**).

e.g.

7 indicates “all permissions” (4+2+1)

6 indicates “read and write” (4+2)

We need to specify 3 integers (for u, g, o).

e.g. 754: user can read, write, and execute; group members can read and execute, others can read.

```
chmod 754 my_script.sh
```

-R recursively (for a directory)

```
chmod -R 755 directory
```

Some operators

> Redirect (into a file):

```
ls > output.txt
cat a.txt > b.txt
cat > sample.txt
```

| Pipe (redirect into a program):

```
ls -l | wc
cat a.txt | wc
ls -al | sort
ls *.txt | cat > txtFile
```

Some operators

; Multiple commands combined (execute all in order):

```
    echo "Contents:"; ls
```

&& Execute the second command if the first command succeeds:

```
    gcc a.c && ./a.out
```

& Execute in the background:

```
    ./a.out &
```

Terminating processes

How to terminate a process? **ctrl+c**

How to terminate a process running in the background? **Close the terminal**

How to terminate a process without an access to the terminal in which the process is run? (Assume we know the pid.)

`kill <PID>`, or `kill -sigkill <PID>` for forcing

`top -b` or `top -b | grep <NAMEOFPROCESS>` to learn PID, then see above

There are many other commands

`history`

which command (e.g. `which ls`)

`whoami`

`date`

`cal`

`cat /proc/cpuinfo`

`cat /proc/meminfo`

`diff a.txt b.txt` (Show difference between contents. **a**:add, **c**:change, **d**:delete)

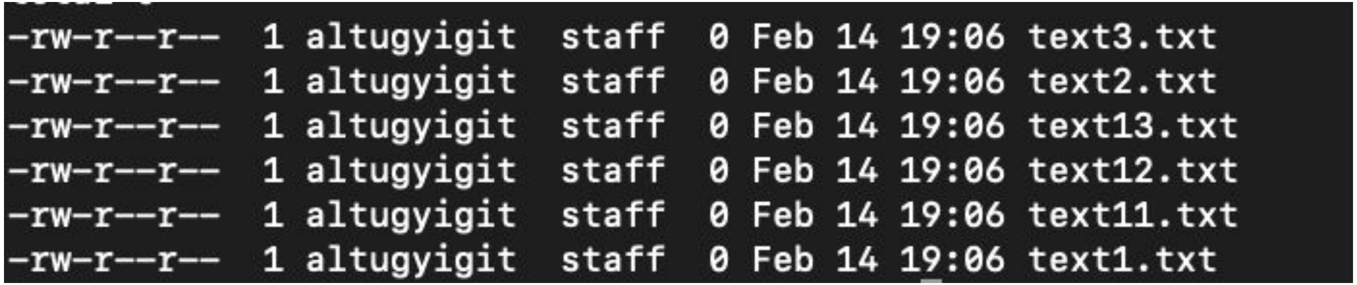
`echo $PATH`

(PATH is an environment variable. If your program is in one of these directories, you can run it no matter what your working directory is.)

Others: wget, tar, ping, ...

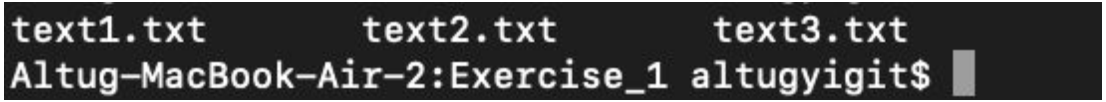
Exercises - Part 1

1. **Create** a directory named Exercise_1, then create files named text1.txt, text2.txt, text3.txt, text11.txt, text12.txt, text13.txt in it. **List all files** with permissions in reverse order by name.



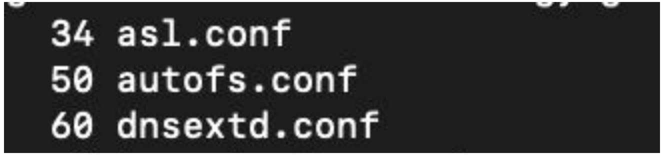
```
-rw-r--r--  1 altugyigit  staff  0 Feb 14 19:06 text3.txt
-rw-r--r--  1 altugyigit  staff  0 Feb 14 19:06 text2.txt
-rw-r--r--  1 altugyigit  staff  0 Feb 14 19:06 text13.txt
-rw-r--r--  1 altugyigit  staff  0 Feb 14 19:06 text12.txt
-rw-r--r--  1 altugyigit  staff  0 Feb 14 19:06 text11.txt
-rw-r--r--  1 altugyigit  staff  0 Feb 14 19:06 text1.txt
```

2. **List** text1.txt to text3.txt specifying the numbers (i.e. 1-3).



```
text1.txt       text2.txt       text3.txt
Altug-MacBook-Air-2:Exercise_1 altugyigit$ 
```

3. **Change** your current directory as “/etc”, then print **number of lines** for each configuration (.conf) file in the directory.



```
  34 asl.conf
  50 autofs.conf
  60 dnsextd.conf
```

4. **Display** count of all **configuration** files (.conf) in etc, the output will be just a number.

5. **Search** for the **cd command** in **history**, then **write** output to the "/home/[user_name]/Desktop/out.txt" (ignore the square brackets, type your username)

6. **Create** a python file (my_code.py) and **copy** the code below into the file. **Ignore** comment lines (lines starting with #) and **write** it in a new file called my_code_new.py.

```
# Python program to determine whether
# the number is Armstrong number or not
# Function to calculate x raised to
# the power y
def power(x, y):
    if y == 0:
        return 1
    if y % 2 == 0:
        return power(x, y // 2) * power(x, y // 2)
    return x * power(x, y // 2) * power(x, y // 2)
```

Shell Scripts

hello.sh

- First line should specify the shell:
  `#!/bin/bash` for GNU Bash, or
  `#!/bin/sh` for POSIX Shell.
  (We will be using **bash**, which is a superset of **sh**.)

- An example **hello.sh**:

  ```
  #!/bin/bash
  echo "Hello, World!" # We can use any shell command.
  ```

- It must be readable and executable:
  e.g. `chmod a+rx hello.sh`

- Now we can execute: `./hello.sh`

variables.sh

$x

Write value: `var=value`

- There is no space around =
- Don’t forget the quotation marks when needed: `course="CENG 322"`

Read value: `$var`

```
#!/bin/bash
code=322
echo "CENG $code"
planet=World
echo "Hello $planet"
```

arguments.sh

$0 The filename of the current script.

$1, $2, $3, ... Command-line arguments

$# The number of command-line arguments

```
#!/bin/bash
echo "$# arguments were given to $0: $1, $2, $3"
```

```
$ ./arguments.sh 1 2 3
3 arguments were given to ./argument.sh: 1, 2, 3
```

parameter-expansion.sh

${x}

${var} is an alternative to $var

```
#!/bin/bash
fruit=apple
echo "I have 3 ${fruit}s."   # $fruits would not work!
# You can do more. For example:
echo "THIS IS AN ${fruit^^}."
# ${parameter^} first character to uppercase
# ${parameter^^} all characters to uppercase
# ${parameter,} first character to lowercase
# ${parameter,,} all characters to lowercase
```

parameter-expansion.sh

${x}

`${var}` is an alternative to `$var`

```
#!/bin/bash
fruit=apple
echo "I have 3 ${fruit}s."   # $fruits would not work!
# You can do more. For example:
echo "THIS IS AN ${fruit^^}."
# ${parameter^} first character to uppercase
# ${parameter^^} all characters to uppercase
# ${parameter,} first character to lowercase
# ${parameter,,} all characters to lowercase
```

arrays.sh

Same syntax can be used with arrays: e.g. `${array[0]}`

```
#!/bin/bash
courses=("CENG322" "CENG312")  # Define an array
courses+=("CENG316")           # Add an element
echo "${courses[1]}"           # Read an element
# ${courses[1]} is CENG312
# Read all elements: ${courses[*]} or ${courses[@]}
# But they are different (@ is usually what you need):
files=(a.txt b.txt)
ls "${files[*]}"  # equivalent to ls "a.txt b.txt"
ls "${files[@]}"  # equivalent to ls "a.txt" "b.txt"
```

arrays.sh

Same syntax can be used with arrays: e.g. `${array[0]}`

```
#!/bin/bash
courses=("CENG322" "CENG312")  # Define an array
courses+=("CENG316")           # Add an element
echo "${courses[1]}"           # Read an element
# ${courses[1]} is CENG312
# Read all elements: ${courses[*]} or ${courses[@]}
# But they are different (@ is usually what you need):
dirs=(a b)
ls "${dirs[*]}"  # equivalent to ls "a b"
ls "${dirs[@]}"  # equivalent to ls "a" "b"
```

user-input.sh

```
#!/bin/bash

echo "Enter your name: "
read name
echo "This is your name: ${name}"

echo "Enter two numbers: "
read num1 num2

read -p "username: " user_var
read -sp "password: " pass_var
```

user-input.sh

```
#!/bin/bash

echo "Enter your name: "
read name
echo "This is your name: ${name}"

echo "Enter two numbers: "
read num1 num2

read -p "username: " user_var
read -sp "password: " pass_var
```

subshell.sh

`(...)` `$(...)`

```
#!/bin/bash

a=1
(a=2; echo "inside: a=$a") # execute in a subshell
echo "outside: a=$a"

echo "The current date is $(date)"
# execute in a subshell and return its output
```

evaluation.sh

((...)) $((...))

```
#!/bin/bash

((a=2+3))
# Perform arithmetic but don't return the
# result (instead change the values of shell variables)

a=$((2+3))
# Perform arithmetic and return the result
```

Exercise

```
$ ./add.sh 8 5
13
```

Test

```
type [
which [
```

```
[ 1 -eq 1 ]  # There are spaces inside brackets

[ abc = abc ] && echo "equal"
```

Strings: = or !=

Numbers: -ne: not equals, -eq: equals, -lt: less than, …

Files and dirs: -f is file, -d is directory, -e exists, -x is executable, …

```
man [
```

Test

```
type [
which [

[ 1 -eq 1 ]  # There are spaces inside brackets

[ abc = abc ] && echo "equal"
```

Strings: = or !=

Numbers: -ne: not equals, -eq: equals, -lt: less than, …

Files and dirs: -f is file, -d is directory, -e exists, -x is executable, …

```
man [
```

Test

```
type [
which [

[ 1 -eq 1 ]  # There are spaces inside brackets

[ abc = abc ] && echo "equal"
```

Strings: = or !=

Numbers: -ne: not equals, -eq: equals, -lt: less than, …

Files and dirs: -f is file, -d is directory, -e exists, -x is executable, …

man [

condition.sh

```
#!/bin/bash
a="x"
echo "enter a file path"
read b
if [ $a = $b ]
then
   echo "a and b are the same"
else
   echo "a and b are different"
fi
# You can use elif [ ... ] then as well.
```

case.sh

```
#!/bin/bash
case $word in
   pattern1)
      Statement(s) to be executed if pattern1 matches
      ;;
   *)
      Default condition to be executed
      ;;
esac
```

loop.sh

```
#!/bin/bash
counter=0
while [ $counter -ne 10 ]
do
   echo "iteration $counter"
   counter=$((counter+1))
done
```

function.sh

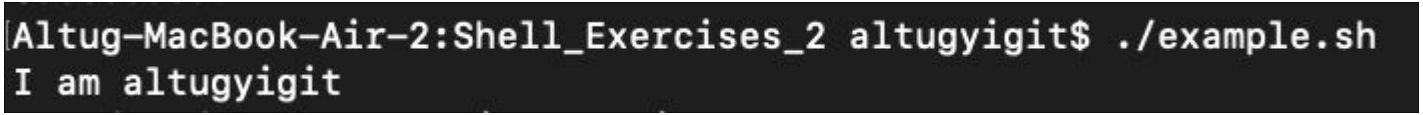
```
#!/bin/bash
add () {
    sum=$(($1 + $2))
    return $sum
}

add 2 2
result=$?
echo "2 + 2 = $result"
```

read_file.sh

```
#!/bin/bash
while read -r line

do

   # SOME COMMAND

done < text.txt
```

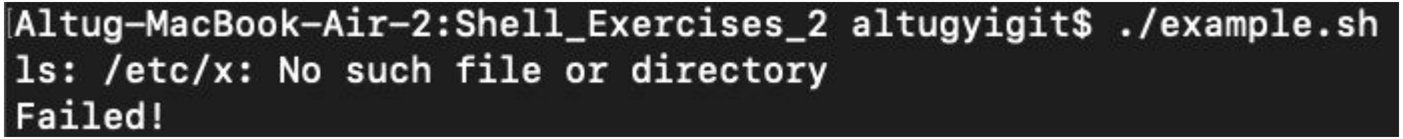
Exercises - Part 2

1. **Store** the output of the command “whoami” in a variable. **Display** “I am _.” where “_” is the output of the “whoami” command.



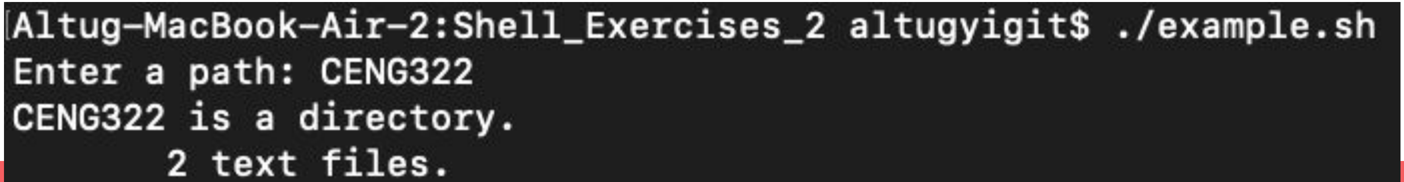
```
[Altug-MacBook-Air-2:Shell_Exercises_2 altugyigit$ ./example.sh
I am altugyigit
```

2. **Write** a script that executes the command “ls /etc/x”. If the command returns a 0 exit status, report “Succeeded” and exit with a 0 exit status. If the command returns a non-zero exit status, report “Failed!” and exit with a 1 exit status.



```
[Altug-MacBook-Air-2:Shell_Exercises_2 altugyigit$ ./example.sh
ls: /etc/x: No such file or directory
Failed!
```

3. **Write** a shell script that prompts the user for a name of **a file or directory** and **displays** messages if it is a file or a directory. If it is a directory, **display count of the text** (i.e. .txt) files in it. (Note: use if, elif with parameters -f and -d)

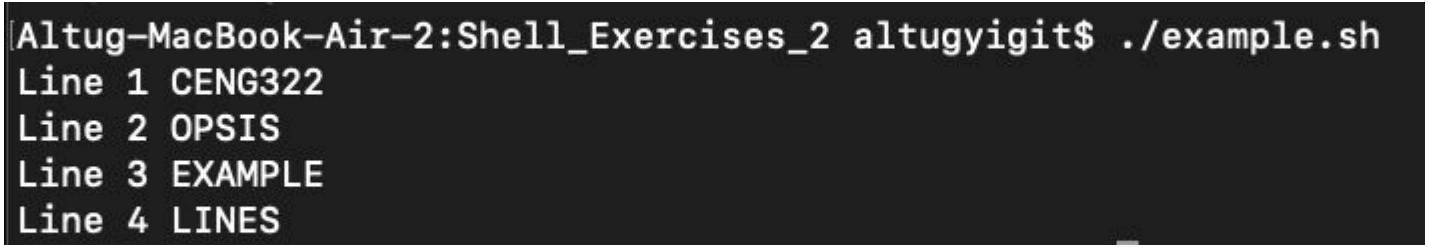


```
[Altug-MacBook-Air-2:Shell_Exercises_2 altugyigit$ ./example.sh
Enter a path: CENG322
CENG322 is a directory.
       2 text files.
```

4. **Write** a shell script that takes input from the user for an **answer to the question** “Are you a student? Y/N”. If the **answer is yes** (or y,Y,yEs etc.), **display** "You got the discount :)", if **it is no** (or n,N,nO etc.), **display** "No discount!". If **anything else** is entered, **display** “Please enter y/yes or n/no". (Note: use the case...esac statement)

5. **Rename** all txt files in the current directory as “new_OLDFILENAME.txt”. (Hint: use ls and for loop)

6. **Read** and **display** a .txt file in the current dir **line by line** with the line numbers. (Hint: use while, read, echo)



```
Altug-MacBook-Air-2:Shell_Exercises_2 altugyigit$ ./example.sh
Line 1 CENG322
Line 2 OPSIS
Line 3 EXAMPLE
Line 4 LINES
```