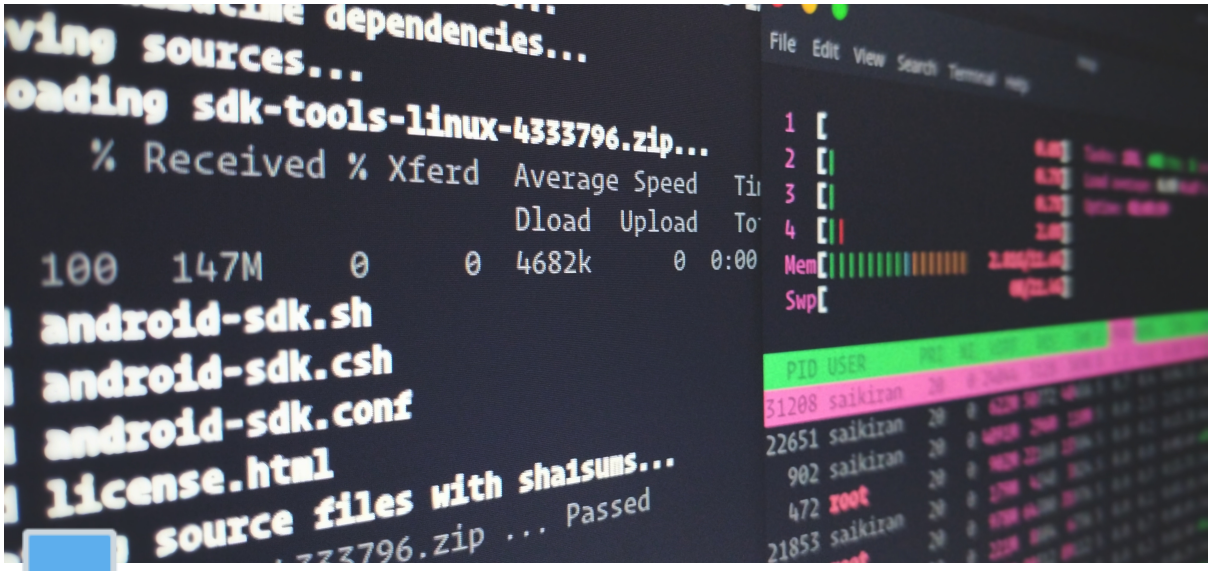


Session-3 @Ibrahim Abou Elenein

Basic Commands

Echo

Display a line of text

```
$ echo <text string to display>
$ echo $<Varaible name>
$ echo Good Morning
$ myVar=5
$ echo $myVar
$ my_name="Ibrahim Abou Elenein"
$ echo My Name is $my_name # The variable will be substitued
$ echo "My Name is $my_name" # The variable will be substitued
$ echo 'My Name is $my_name'
$ man echo
```

Cat

Display text file

```
$ cat file1
$ cat file2
```

```
$ cat file1 file2
$ cat file{1,2} # Don't forget :D
```

Less

displays the contents of a file or a command output (a Pager)

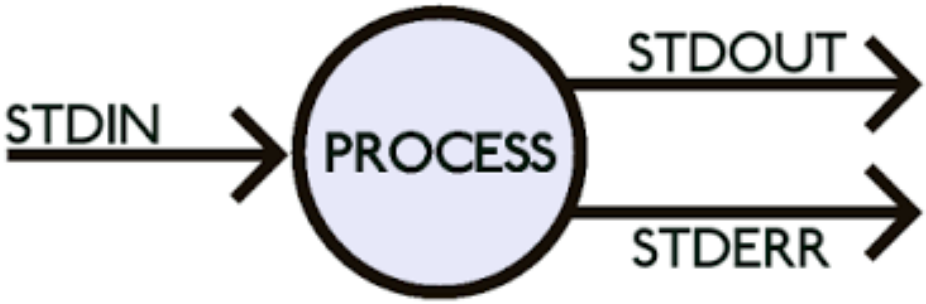
Less is More

```
$ less filename
$ less -N filename # show line numbers
```

Composite Commands

Linux command Philosophy

- Create smalll, portable , specialized programs that perform one task well
- Make this program capable of reciveing input from, and redirect output to, other programs
- This way, Linux commands are like LEGO blocks



I/O Streams

Function	Stream Name	Stream Desriptor	Deafult Device
Input	stdin	o	keyboard
Output	stdout	1	Screen

Function	Stream Name	Stream Desriptor	Deafult Device
Error	stderr	2	Screen

Sequential Commands

we can have mulipule commands in the same line :

```
$ <first Command>;<second Command>;<third one>
$ echo "one"; sleep 10; echo "two"
```

Using Sequential commands is useful when the first command takes long time to excute and we won't to wait until it is complete

```
$ make; sudo make install # compile
```

Note : The Sequential commnds jsut run after each other, they have independent I/O

Conditional Commands

ORing (" || ")

Second command will only be excute if the First Returns Falure (Used for error handling)

```
$ cat <filename> || echo "File Not Found"
```

ANDing (" && ")

Seconed command will only excute if the First Returns Successfully

```
$ mkdir dir1 && cd dir1
```

👽 I / O Redirection ：

Standard Output Redirection

```
$ command > file # redirect output to file; overwrite
$ command >> file # redirect output to file; Appened
$ echo "Hello World!"
$ echo "Hello World!" > greetings.txt
$ ls -lah /usr/bin >>file-listings.txt
$ cat file{1..3} > combinded-file.txt
$ cat *.log > all.log
```

Note :

- Error message still go to the screen
- ">" stands for "1>" which means Redirect Output Stream

Standard Error Redirection

```
$ command 2> file # overwrite
$ command 2>> file #Append
$ make 2> log
$ make 2>> log
```

Both Output & Error Redirection

```
$ command > file1 2>> file2
$ command > file 2>&1 (both to file)
$ command &>file
$ command &>>file
```

Standard Input Redirection

```
$ command < file # input to the command is read from the file
$ wc -l < log-file.log # count the number of lines of the files log-file
$ sort < log-file > sorted-log-file
$ mail me@guc.edu.eg < resume
$ spell < report.txt  > error.log
```

Note : you can redirect to devices

Common Devices for Redirection

- /dev/stdout Standard output device
- /dev/stderr Standard error device
- /dev/stdin Standard input device
- /dev/null This device is useful for being a data sink. This is useful when we want to discard the command output (Such as compiler long output)
- /dev/zero This device is useful as an input device to generate an infinite stream of zeros
- /dev/random This device is useful as an input device to generate random bytes. 😮
- /dev/full This device is used as an output device to simulate a full file. it is used for testing purposes

Using Pipes

What is Pipe ? A pipe is a mechanism in Linux Kernel that is used to enable one process to send info to other process

```
$ command1 | command2 | command3
$ cat *.log | grep "error"
$ man gzip | grep -i "compress"
$ cat *.py | grep "print" | less
$ cat name-list.txt | sort | uniq -c
$ cat * | grep -i "error" | grep -v "severe" | sort > file.log
```

Tee command

it redirects to file & stdout

```
$ make | tee make-out.txt
$ date | tee -a file{1..4} # appned
$ make | tee make-out.txt | grep "error" | sort > error.log
```

Thanks💥