جامعة القاهرة الجديدة
التكنولوجية

Fall 2023

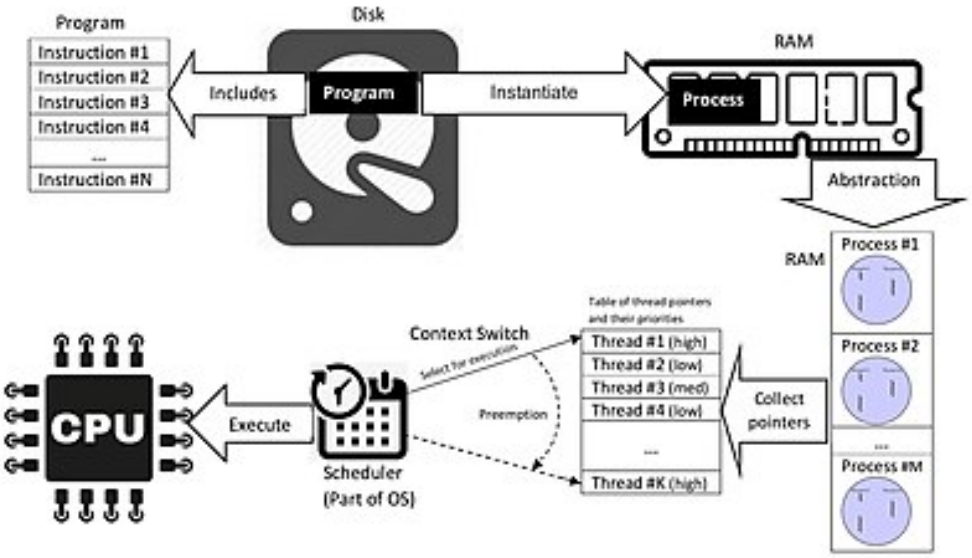
(4)

Linux Essentials

Dr. Hatem Yousry

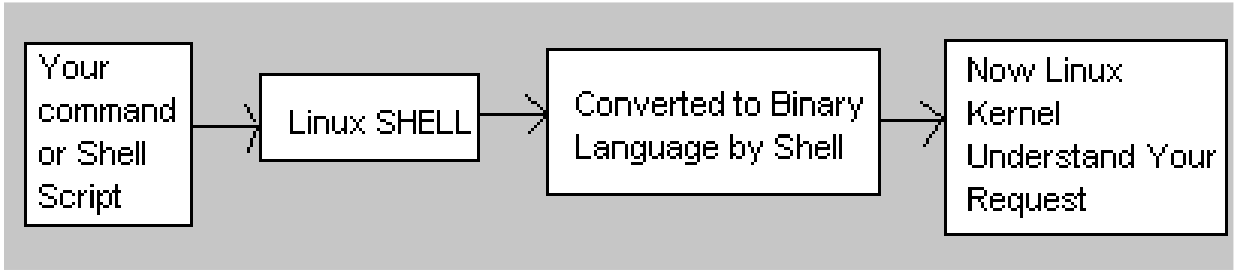
Agenda

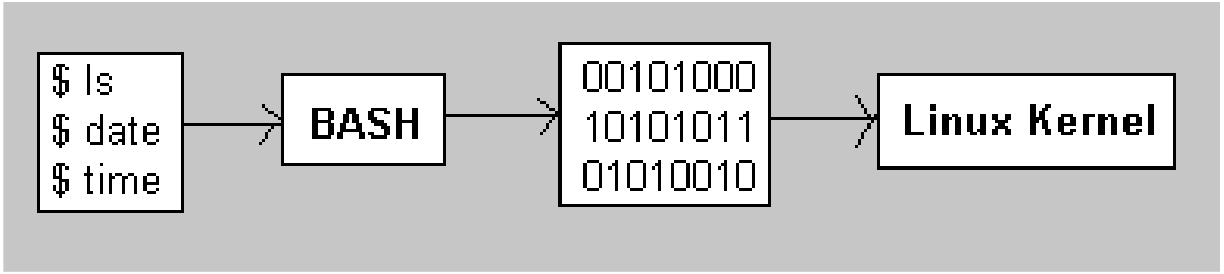
- **Pipe and Filter Architecture.**
- **What is Processes.**
- **Redirection.**
- **Shell Variables.**



What's Linux Shell

- Computer understand the language of 0's and 1's called binary language, In early days of computing, **instruction are provided using binary language,** which is difficult for all of us, to read and write. So in O/s there is special program called Shell. **Shell accepts your instruction or commands** in English and **translate it into computers native binary language.**





Shell Types

- It's environment provided for user interaction. Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file. Linux may use one of the following most **popular shells**

Shell Name	Developed by	Where	Remark
BASH (Bourne-Again SHell)	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's syntax and usage are very similar to the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	

- (In MS-DOS, Shell name is COMMAND.COM which is also used for same purpose, but it's not as powerful as our Linux Shells are!)

Shell Shortcuts for bash

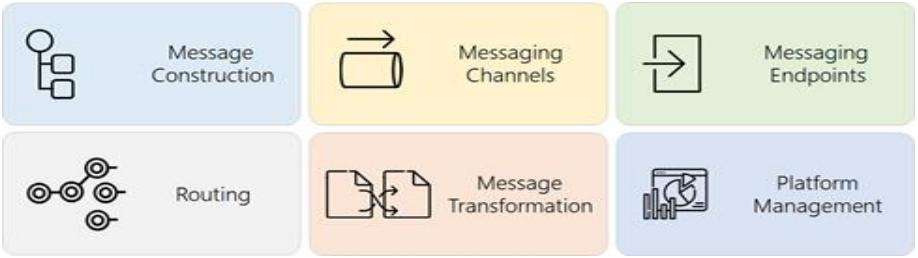
- Ctrl-A (jump to start of line)
- Ctrl-E (jump to end of line)
- Ctrl-K (delete (kill) everything from the cursor onwards
- Ctrl-W (delete the previous word only)
- Ctrl-Y (paste whatever was just deleted)
- Ctrl-C (kill/exit a running process)
- Ctrl-L (clear the screen)
- Ctrl-R (search for previously executed commands)
- Tab (auto-complete command or file/directory name)
- ↑ / ↓ (scroll back / forwards through previously entered commands)

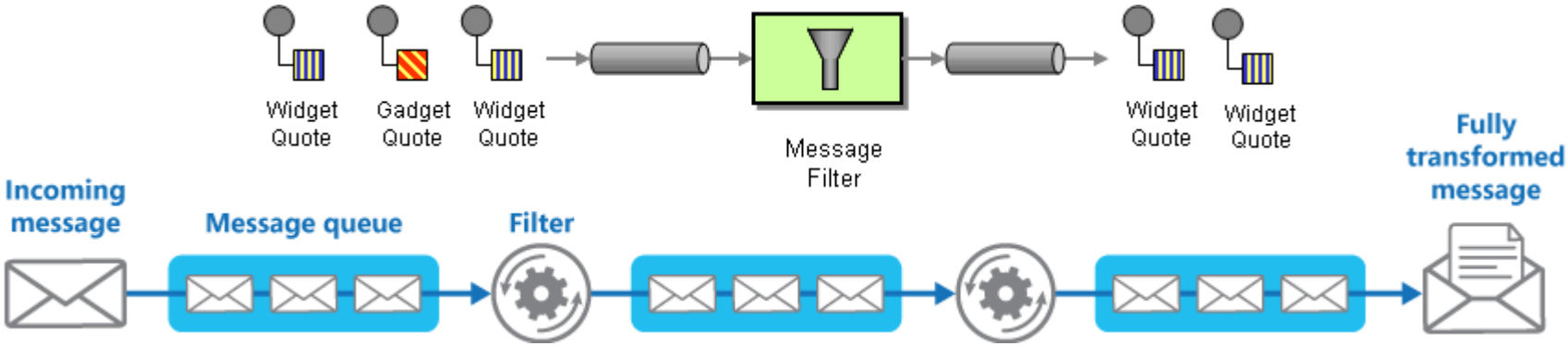
Shell Prompt Example

- Code segments and script output will be displayed as monospaced text. Command-line entries will be preceded by the **Dollar sign ($).** If your prompt is different, enter the command:
- **PS1="$ " ; export PS1**
- Then your interactions should match the examples given (such as ./my-script.sh below).
- Script output (such as **"Hello World"** below) is displayed at the start of the line.
- **$ echo '#!/bin/sh' > my-script.sh**
- **$ echo 'echo Hello World' >> my-script.sh**
- **$ chmod 755 my-script.sh**
- **$ ./my-script.sh**
- **Hello World**
- **$**

Pipe and Filter Architecture

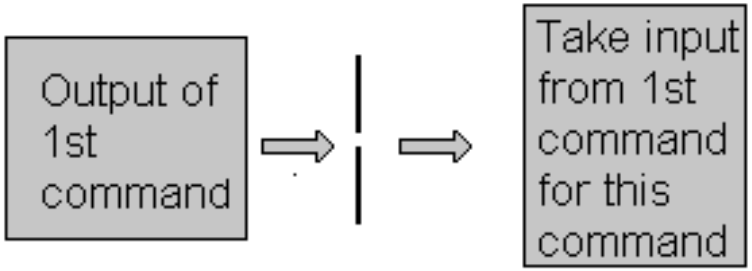
- **Pipe and Filter** is an **architectural** pattern, which has independent entities called **filters** (**components**) which perform transformations on data and process the input they receive, and **pipes**, which serve as connectors for the stream of data being transformed (**Links**), each connected to the next component in the **pipeline**.





Pipes

- A pipe is a way to **connect the output of one program to the input of another program without any temporary file.**
- A pipe is nothing but a temporary storage place where the output of one command is **stored and then passed** as the input for second command.
- Pipes are used to run more than two commands (Multiple commands) from same command line.
- Syntax: **command1 | command2**



Pipes

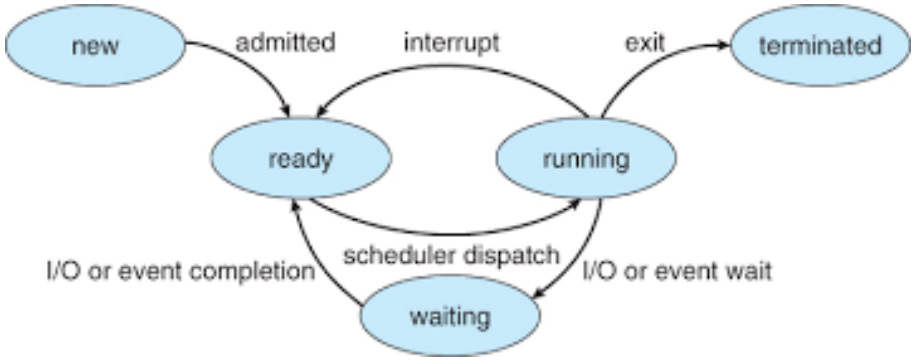
Command using Pips	Meaning or Use of Pipes
$ ls | more	Here the output of ls command is given as input to more command So that output is printed one screen full page at a time
$ who | sort	Here output of who command is given as input to sort command So that it will print sorted list of users
$ who | wc -l	Here output of who command is given as input to wc command So that it will number of user who logon to system
$ ls -l | wc -l	Here output of ls command is given as input to wc command So that it will print number of files in current directory.
$ who | grep raju	Here output of who command is given as input to grep command So that it will print if particular user name if he is logon or nothing is printed (To see for particular user logon)

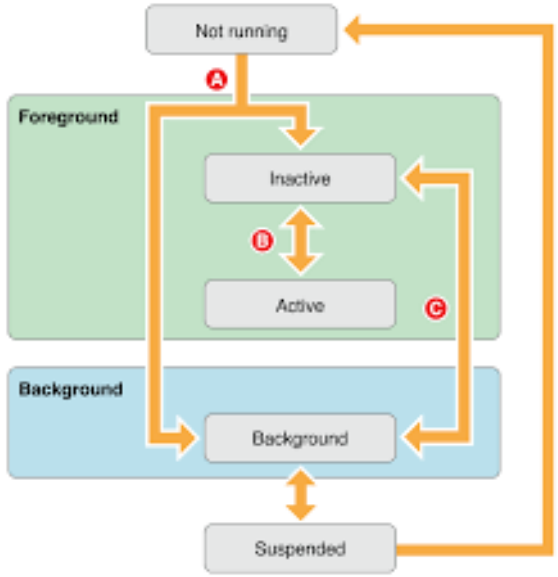
Filter

- If a Linux command accepts its input from the standard input and produces its output on standard output is know as a filter.
- **A filter performs some kind of process on the input and gives output.**
- For e.g.. Suppose we have file called 'hotel.txt' with 100 lines data, And from 'hotel.txt' we would like to print contains from line number 20 to line number 30 and store this result to file called 'hlist' then give command
- **$ tail +20 < hotel.txt | head -n30 >hlist**
- Here head is filter which takes its input from **tail command** (tail command start selecting from line number 20 of given file i.e. hotel.txt) and passes this lines to input to head, whose output is redirected to 'hlist' file.

What is Processes

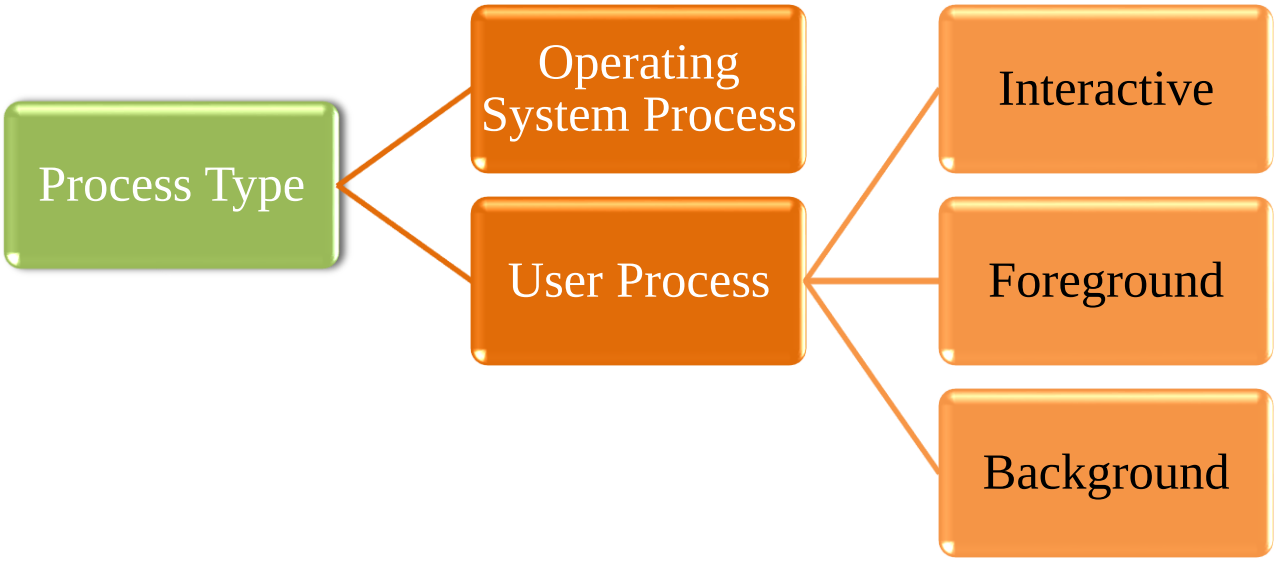
- **Process is any kind of program or task carried out by your PC.**
- For e.g. **$ ls -lR ,** is command or a request to list files in a directory and all subdirectory in your current directory.
- It is a process. A process is program (command given by user) to perform some Job. In Linux when you start process, it gives a number **(called PID or process-id),** PID starts from 0 to 65535.





Foreground and Background Processes

- Processes that require a user to **start them or to interact with them are called foreground processes. Processes that are run independently of a user are referred to as background processes.**
- A background process is a computer process that **runs behind** the scenes (i.e., in the background) and without user intervention.
- Programs and commands run as foreground processes by default.



Why Process required?

- **Linux is multi-user, multitasking O/S.**
- It means you can run more than two process simultaneously if you wish. For e.g.. To find how many files do you have on your system you may give command like
- **$ ls / -R | wc –l**
- **LS** read the entire directory into virtual memory for sorting and The **wc** ("word count") command counts the lines, words, and characters in a file**; ls | wc.** This pipes the output of ls through wc.
- This command will take lot of time to search all files on your system. So you can run such command in Background or simultaneously by giving command like
- **$ ls / -R | wc -l &**
- The **ampersand (&)** at the end of command tells shells start command (ls / -R | wc -l) and run it in background takes next command immediately.
- An instance of running command is called process and the number printed by shell is called **process-id (PID),** this PID can be use to refer specific running process.

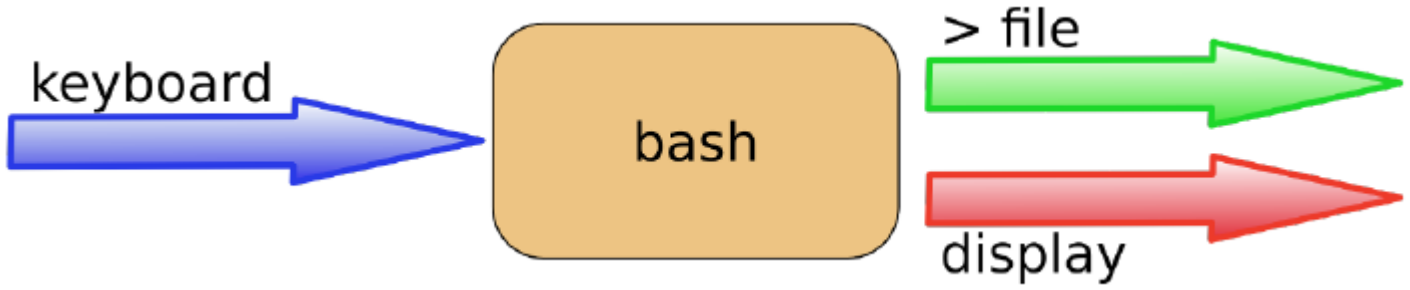
Linux Command Related with Process

- NOTE that you can only kill process which are created by yourself. A Administrator can almost kill 95-98% process. But some process can not be killed, such as **VDU Process**., A VDU is a machine with a screen which is used to display information from a computer. VDU is an abbreviation for '**visual display unit**'.

For this purpose	Use this Command	Example
To see currently running process	ps	**$ ps**
To stop any process i.e. to kill process	kill {PID}	**$ kill 1012**
To get information about all running process	ps -ag	**$ ps -ag**
To stop all process except your shell	kill 0	**$ kill 0**
For background processing (With &, use to put particular command and program in background)	linux-command &	**$ ls / -R | wc -l &**

Redirection

- **Redirection of Standard output/input or Input – Output redirection.**
- Mostly all command gives output on screen or take input from keyboard, but in Linux it's possible to **send output to file or to read input from file.**
- For e.g. **$ ls** command gives output to screen; to send output to file of ls give command , **$ ls > filename**. It means put output of ls command to filename.
- **There are three main redirection symbols** >,>>,<

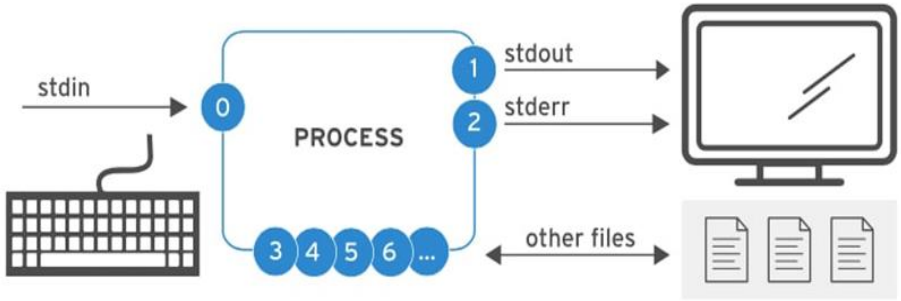


> Redirector Symbol

- Syntax: **Linux-command > filename**
- To output Linux-commands result to file.
- Note that If file already exist, it will be **overwritten** else new file is created.
- For e.g. To send output of ls command give
- **$ ls > myfiles**
- Now if '**myfiles**' file exist in your current directory it will be overwritten without any type of warning.
- (What if I want to send output to file, which is already exist and want to keep information of that file without loosing previous information/data?, For this Read next redirector)

>> Redirector Symbol

- Syntax: **Linux-command >> filename**
- To output Linux-commands result to END of file.
- Note that **If file exist** , it will be **opened and new information / data will be written to END of file**, without losing previous information/data, **And if file is not exist, then new file is created.**
- For e.g. To send output of date command to already exist file give
- **$ date >> myfiles**



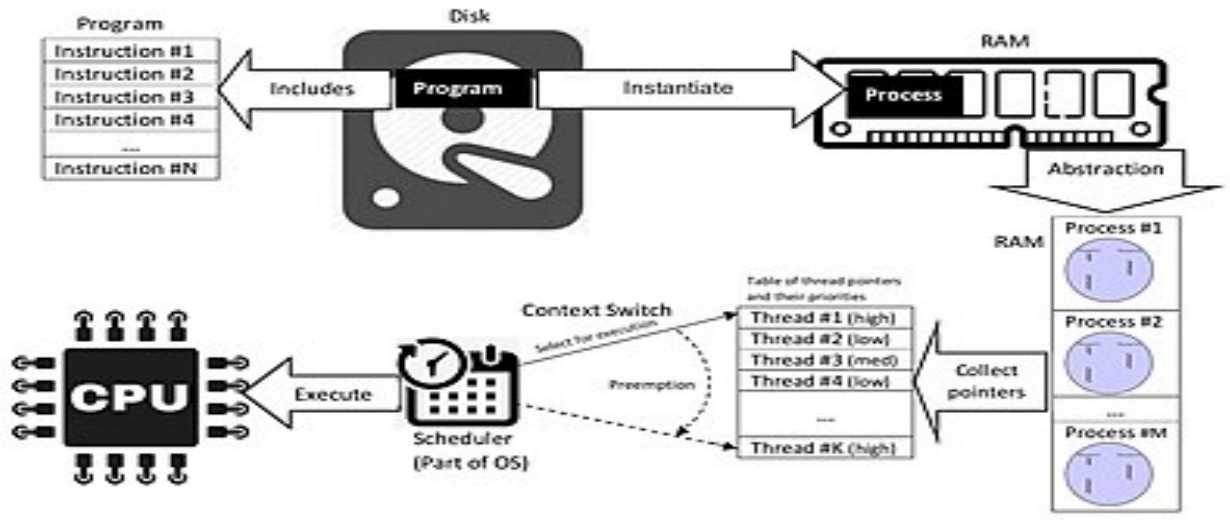
< Redirector Symbol

- Syntax: **Linux-command < filename**
- To **take input** to Linux-command from file instead of key-board.
- For e.g. To take input for cat command give
- **$ cat < myfiles**

Character	*Action*
>	Redirect standard output
>&	Redirect standard output and standard error
<	Redirect standard input
|	Redirect standard output to another command (pipe)
>>	Append standard output
>>&	Append standard output and standard error

Shell Variables

- **Shell program is series of Linux commands.** Shell script is just like batch file is MS-DOS but have more power than the MS-DOS batch file. Shell script can take input from user, file and output them on screen. Useful to create our own commands that can save our lots of time and to automate some task of day today life.
- Variables in Linux Sometimes to process our data/information, it must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold our data.



Shell Variables

- Programmer can give a unique name to this memory location/address called memory variable or variable (Its a named storage location that may take different values, but only one at a time). **In Linux, there are two types of variable**
  1. **System variables -** Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
  2. **User defined variables (UDV) -** Created and maintained by user. This type of variable defined in lower LETTERS.

Some System variables

- You can see system variables by giving command like $ set, Some of the important System variables are

System Variable	Meaning
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/vivek	Our home directory
LINES=25	No. of columns for our screen
LOGNAME=students	Our logging name
OSTYPE=Linux	Our o/s type : -)
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

NOTE that Some of the above settings can be different in your PC. You can print any of the above variables contain as follows

$ echo $USERNAME

$ echo $HOME

Caution: Do not modify System variable this can some time create problems.

How to define User Defined Variables (UDV)

- To define UDV use following syntax
- Syntax: **variablename=value**
- NOTE: Here 'value' is assigned to given 'variablename' and Value must be on right side = sign; For e.g.
- **$ no=10 # this is ok**
- **$ 10=no # Error, NOT Ok, Value must be on right side of = sign.**
- To define variable called 'vech' having value Bus
- **$ vech=Bus**
- To define variable called n having value 10
- **$ n=10**

Rules for Naming variable name (Both UDV and System Variable)

- **(1) Variable name** must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character. For e.g. Valid shell variable are as follows
  - **HOME**
  - **SYSTEM_VERSION**
  - **vech**
  - **no**
- **(2) Don't put spaces** on either side of the equal sign when assigning value to variable. For e.g.. In following variable declaration there will be no error
  - **$ no=10**
- But here there will **be problem** for following
  - **$ no =10**
  - **$ no= 10**
  - **$ no = 10**

Rules for Naming variable name (Both UDV and System Variable)

- **(3) Variables are case-sensitive**, just like filename in Linux. For e.g.
  - **$ no=10**
  - **$ No=11**
  - **$ NO=20**
  - **$ nO=2**
- Above all are different variable name, so to print value 20 we have to use $ echo $NO and Not any
- of the following
  - **$ echo $no # will print 10 but not 20**
  - **$ echo $No # will print 11 but not 20**
  - **$ echo $nO # will print 2 but not 20**
- **(4) You can define NULL variable** as follows (NULL variable is variable which has no value at the time of definition) For e.g.
  - **$ vech=**
  - **$ vech=""**
- Try to print it's value $ echo $vech , Here nothing will be shown because variable has no value i.e. NULL variable.
- **(5) Do not use ?,*** etc, to name your variable names.

How to print or access value of UDV (User Defined Variables)

- To print or access UDV use following syntax
- Syntax: **$variablename**
- For eg. To print contains of variable 'vech'
- **$ echo $vech**
- It will print 'Bus' (if previously defined as vech=Bus) ,To print contains of variable 'n' **$ echo $n**
- It will print '10' (if previously defined as n=10)
- **Caution:** Do not try $ echo vech It will print vech instead its value 'Bus' and $ echo n, It will print n instead its value '10', **You must use $ followed by variable name.**

Question for Variables

- **Q.1.How to Define variable x with value 10 and print it on screen**
- **$ x=10**
- **$ echo $x**

- **Q.2.How to Define variable xn with value Rani and print it on screen**
- **$ xn=Rani**
- **$ echo $xn**

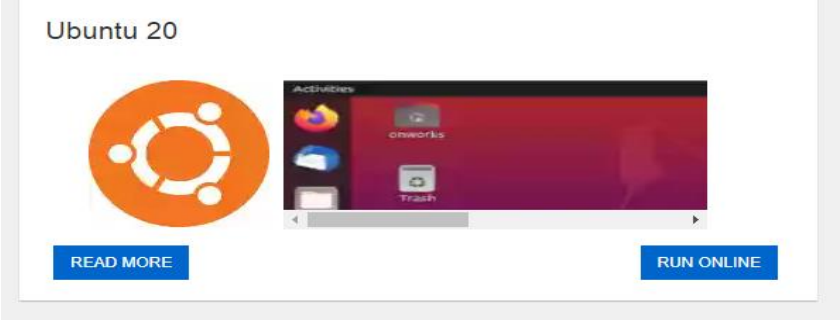
Question for Variables

- **Q.3.How to print sum of two numbers, let's say 6 and 3**
- **$ echo 6 + 3**
- This will print 6 + 3, not the sum 9, To do sum or math operations in shell use expr, syntax is as follows **Syntax: expr op1 operator op2**
- Where, op1 and op2 are any Integer Number (Number without decimal point) and operator can be
- **+ Addition**
- **- Subtraction**
- **/ Division**
- **% Modular**, to find remainder For e.g. **20 / 3 = 6** , to find remainder 20 % 3 = 2, (Remember its integer calculation)
- *** Multiplication**
- **$ expr 6 + 3**
- Now It will print sum as 9 , But
- $ expr 6+3
- will not work because space is required between number and operator

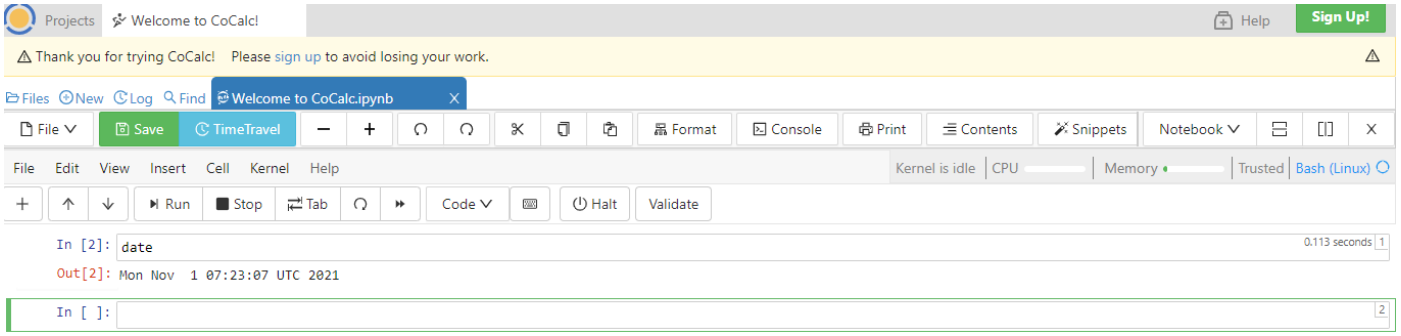
Question for Variables

- **Q.4.How to define two variable x=20, y=5 and then to print division of x and y (i.e. x/y)**
- **$x=20**
- **$ y=5**
- **$ expr x / y**
- **Q.5.Modify above and store division of x and y to variable called z**
- **$ x=20**
- **$ y=5**
- **$ z=`expr x / y`**
- **$ echo $z**

Linux Ubuntu Online Terminal



- https://cocalc.com/projects/3bbb1fa0-33ed-45b1-8f56-5c41b8ac7da0/files/Welcome%20to%20CoCalc.ipynb?session=default
- https://chrome.google.com/webstore/detail/ubuntu-free-online-linux/pmaonbjcobmgkemldgcedmpbmmncpbgi?hl=en
- https://www.onworks.net/os-distributions/ubuntu-based



Thank You

Linux

Dr. Hatem Yousry
E-mail: Hyousry@nctu.edu.eg