OSL
2021-22

Rachna R.Chhajed

Course Objective

- To introduce and learn Linux commands required for administration.
- To learn shell programming concepts and applications.
- To demonstrate the functioning of OS basic building blocks like processes, threads under the LINUX.
- To demonstrate the functioning of OS concepts in user space like concurrency control (process synchronization, mutual exclusion), CPU Scheduling, Memory Management and Disk Scheduling in LINUX.
- To demonstrate the functioning of Inter Process Communication under LINUX.
- To study the functioning of OS concepts in kernel space like embedding the system call in any LINUX kernel.

Course Outcome

- CO1:Apply the basics of Linux commands.
- CO2: Build shellscripts for various applications.
- CO3: Implement basic building blocks like processes, threads under the Linux.
- CO4: Develop various system programs for the functioning of OS concepts in user space like concurrency control, CPU Scheduling, Memory Management and Disk Scheduling in Linux.
- CO5: Develop system programs for Inter Process Communication in Linux

List of Assignments

Assignment No. 1 :

A. Study of Basic Linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep, sed etc.

B. Write a program to implement an address book with options given below: a) Create address book. b) View address book. c) Insert a record. d) Delete a record. e) Modify a record. f) Exit

Assignment No. 2:

Process control system calls: The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states.

A. Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

B. Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

Assignment No. 3:

Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time.

Assignment No. 4:

A. Thread synchronization using counting semaphores. Application to demonstrate: producer-consumer problem with counting semaphores and mutex.

B. Thread synchronization and mutual exclusion using mutex. Application to demonstrate: Reader-Writer problem with reader priority.

Assignment No. 5:

Implement the C program for Deadlock Avoidance Algorithm: Bankers Algorithm.

Assignment No. 6:

Implement the C program for Page Replacement Algorithms: FCFS, LRU, and Optimal for frame size as minimum three.

List of Assignments

Assignment No. 7:

Inter process communication in Linux using following.

A. FIFOS: Full duplex communication between two independent processes. First process accepts sentences and writes on one pipe to be read by second process and second process counts number of characters, number of words and number of lines in accepted sentences, writes this output in a text file and writes the contents of the file on second pipe to be read by first process and displays onstandard output.

B. Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.

Assignment No. 8: Implement the C program for Disk Scheduling Algorithms: SSTF, SCAN, C-Look considering the initial head position moving away from the spindle.

Study Assignment: Implement a new system call in the kernel space, add this new system call in theLinux kernel by the compilation of this kernel (any kernel source, any architecture and any Linux kernel distribution) and demonstrate the use of this embedded system call using C program in user space.

Assignments::01

1. Shell Programming

A. Study of Basic Linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep,sed etc.

B. Write a program to implement an address book with options given below:

a) Create address book.

b) View address book.

c) Insert a record.

d) Delete a record.

e) Modify a record.

f) Exit

What is kernel??

Kernel:

It is the backbone of Linux OS, which is used to manage resources of Linux OS like memory, I/O, software, hardware management processes.

User → Shell Script → Kernel → PC h/w

- User writes script.
- Script contains instructions.
- Kernel interprets the instruction in machine language.
- As per the instruction kernel controls the PC hardware.

Basis for Comparison	Microkernel	Monolithic Kernel
Size	Microkernel is smaller in size	It is larger than microkernel
Execution	Slow Execution	Fast Execution
Extendible	It is easily extendible	It is hard to extend
Security	If a service crashes, it does effects on working on the microkernel	If a service crashes, the whole system crashes in monolithic kernel.
Code	To write a microkernel more code is required	To write a monolithic kernel less code is required
Example	QNX, Symbian, L4Linux etc.	Linux,BSDs(FreeBSD,OpenBSD,NetBSD)etc.

What is UNIX shell?

UNIX shell is a command line like DOS in Windows.

It's a user interface for UNIX operating system.

Mainly shells are used for inputting user OS commands.

It is called "Shell" because it hides all the information behind the shell interface

Types of Shells:

- **Bourne Shell (sh)**
- **C Shell (csh)**
- **Korn Shell (ksh)**
- **Bash Shell (bash)**

Bourne Shell (sh): It's the default UNIX shell. Most of the scripts to configure OS is written using this shell. It was developed by Stephen Bourne.

C Shell (csh): It is called C shell because the syntax used here is similar to c language. It adds many features compare to bourne shell. This shell is not widely used now. It was developed by Bill Joy.

Shell (ksh): This shell is backward compatible with bourne shell & inherits many features of C shell. This was developed by David Korn.

Bash Shell (bash): It stands for Bourne again Shell i.e. It is superset of bourne shell. It was built by Stephen Bourne.

Finding shell?

How to find which shell we are working at?

There is a simple command to check which shell we are working at…

echo $SHELL

(or)

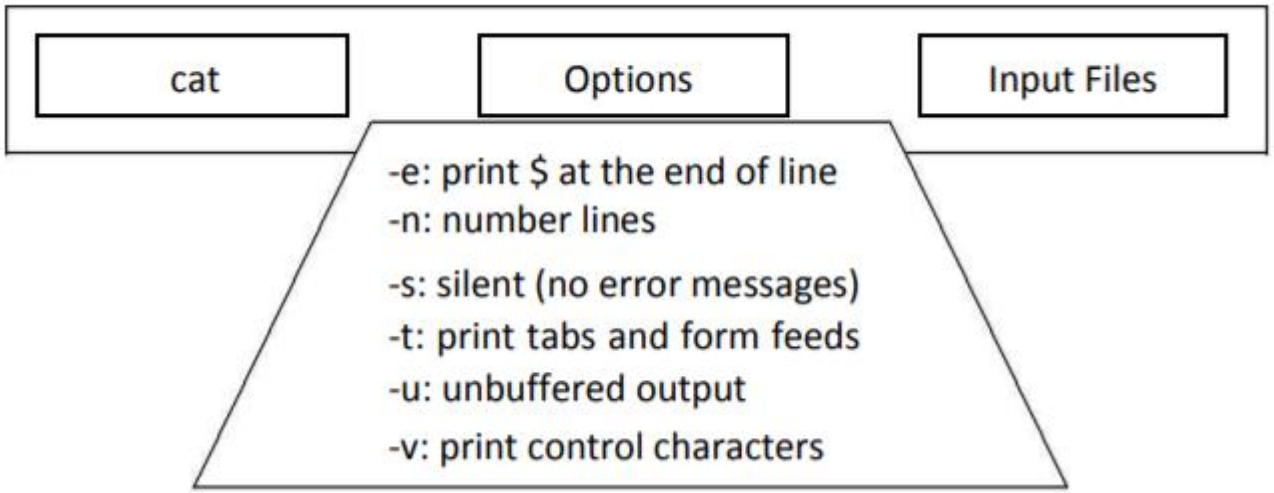
echo $

Filter Commands

FILTER	ACTION
Cat	Passes all data from input to output.
Cmp	Compares two files.
Comm	Identifies common lines in two files.
Cut	Passes only specified columns.
Diff	Identifies differences between two files or between common files in two directories.
Head	Passes the number of specified lines at the beginning of the data.
Paste	Combines columns.
Sort	Arranges the data in sequence.
Tail	Passes the number of specified lines at the end of the data.
Tr	Translates one or more characters as specified.
Uniq	Deletes duplicate (repeated) lines.
Wc	Counts characters, words, or lines.
Grep	Passes only specified lines.
Sed	Passes edited lines.
Awk	Passes edited lines – parses lines.

CONCATENATING FILES:

UNIX provides a powerful utility to concatenate commands.
It is known as the **catenate** command, or **cat** for short.
It combines one or more files by appending them in the order they are listed in the command. The input can come from the keyboard; the output goes to the monitor.



SHELL SCRIPT/PROGRAM

- A Shell Program runs in Interpretive Mode.
- It is not compiled to a separate executable file as a C program is.
- Each statement is loaded into memory when it is to be executed.
- When a group of command have to be executed regularly, they should be stored in a file, and the file itself is executed as a shell script or a shell program.
- The basic concept of a shell script is a list of commands, which are listed in the order of execution.
- A good shell script will have comments, preceded by # sign, describing the steps.

Practical examples where shell scripting actively used:

- Monitoring your Linux system.
- Data backup and creating snapshots.
- Find out what processes are eating up your system resources.
- Find out available and free memory.
-
- Find out all logged in users and what they are doing.

Rule to write Shell script:

Write Script

Give Execute permission to user

Run Script

Debug (optional)

```
#
  #!/bin/sh
```


To start shell script.. you need to alert the system that a shell script is being started. This is done using the **shebang** construct.

#!/bin/sh

This tells system that the commands that follow are to be executed by the Bourne or any shell.

It's called a shebang because the # symbol is called a hash, and the ! symbol is called a bang.

To create a script containing these commands, you put the shebang line first and then add the commands −

Ex: #!/bin/sh
echo”Wel-come to PICT”
ls

Steps to write and execute a script

- Open the terminal. Go to the directory where you want to create your script.
- Create a file with **.sh** extension.
- Write the script in the file using an editor.
- Make the script executable with command **chmod +x <fileName>**.
- Run the script using ./<**fileName**>.

1.To find all available shells in your system type

$ cat /etc/shells

2. There are three types of quotes

"Double Quotes '➔" "

"Double Quotes" - Anything enclose in double quotes removed meaning of that characters (except \ and $).

Single quotes ` ➔' '

'Single quotes' - Enclosed in single quotes remains unchanged.

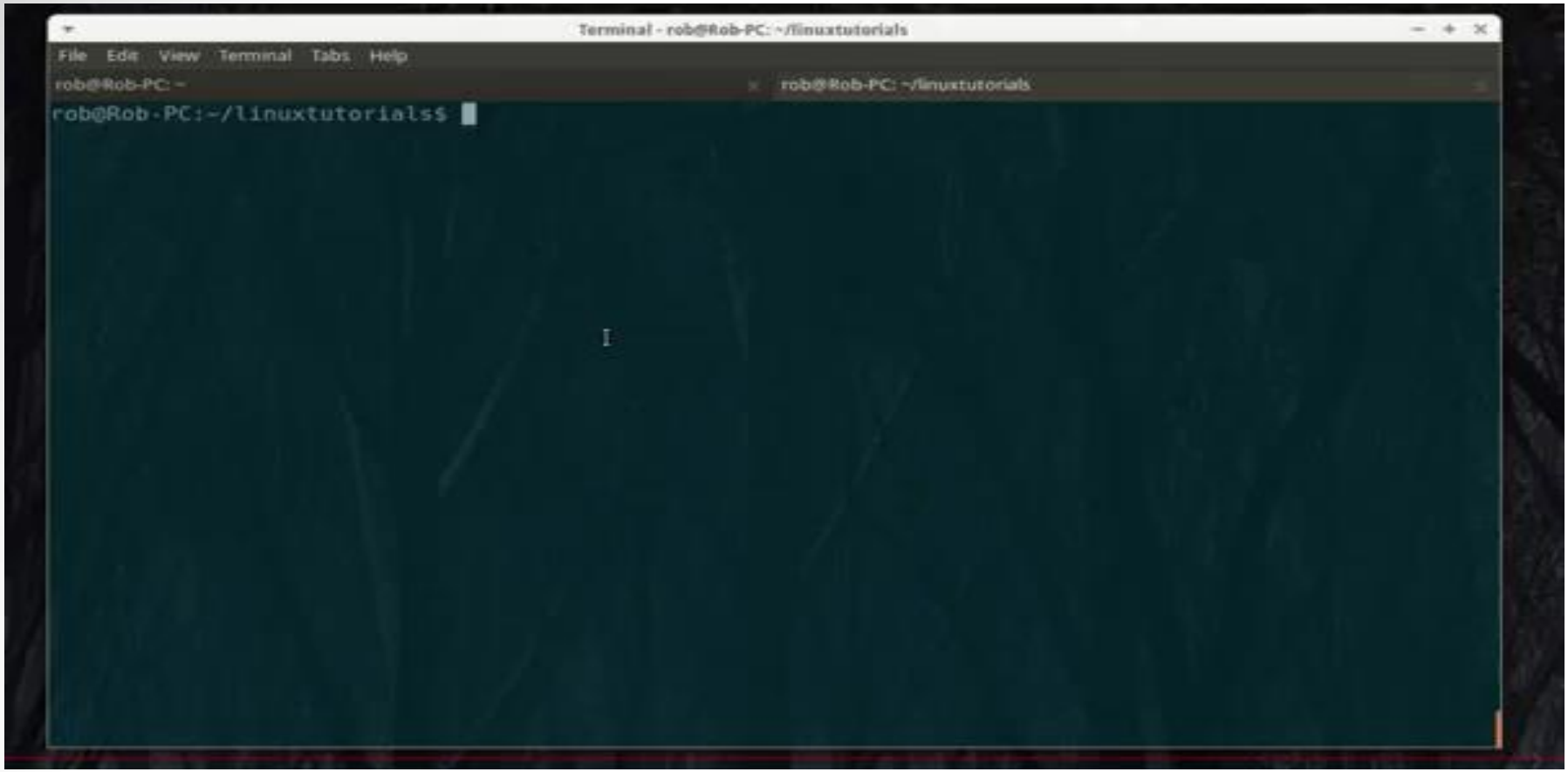
Back quote ➔` `

`Back quote` - To execute command.

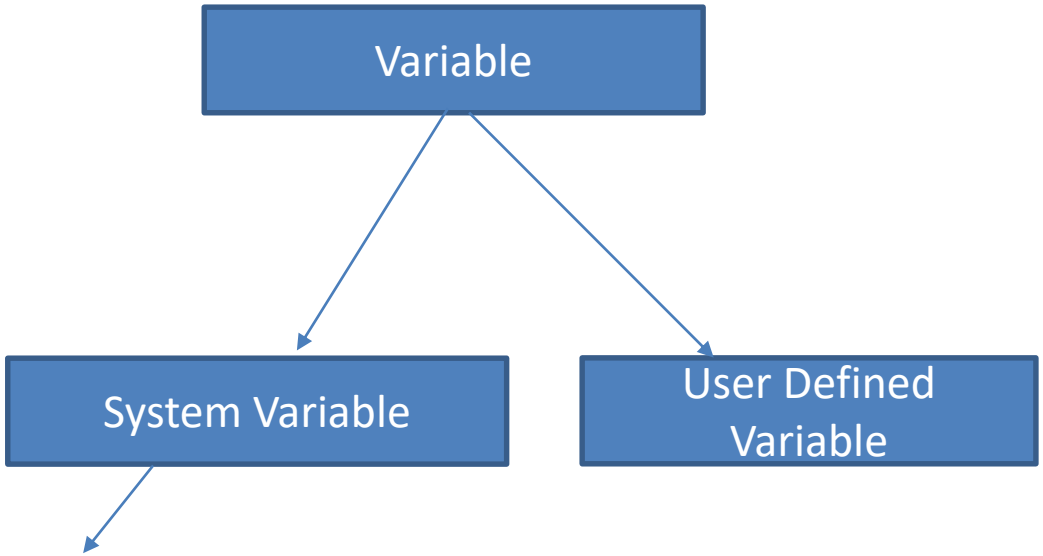
For eg. $ echo "Today is date" Can't print message with today's date.

$ echo "Today is `date`".

How to run the shell script..



Variable



Created and maintained by **Operating System(Linux)**
Generally these variables are defined in **CAPITAL LETTERS**

System Defined Variables	Meaning
BASH=/bin/bash	Shell Name
BASH_VERSION=4.1.2(1)	Bash Version
COLUMNS=80	No. of columns for our screen
HOME=/home/linuxtechi	Home Directory of the User
LINES=25	No. of columns for our screen
LOGNAME=LinuxTechi	LinuxTechi Our logging name
OSTYPE=Linux	OS type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Path Settings
PS1=[\u@\h \W]\$	Prompt Settings
PWD=/home/linuxtechi	Current Working Directory
SHELL=/bin/bash	Shell Name
USERNAME=linuxtechi	User name who is currently login to system

User defined variable

- Values are assigned to user variables using an **equal sign.**
- No spaces can appear between the variable, the equal sign, and the value (another trouble spot for novices).
- examples of assigning values to user variables:
- var1=10
  var2=-57
  var3=testing
  var4="still more testing"
- The shell script **automatically determines the data type** used for the variable value.

- Variables defined within the shell script maintain their values throughout the life of the shell script but are deleted when the shell script completes.

read: Making Scripts Interactive

- The read statement is the shell's internal tool for taking inputs from the user, i.e., making scripts interactive.
- It is used with one or more variable. When we use a statement like

```
read name
```

The script pauses at that point to take i/p from the keyboard.

Since this is the form of assignment, no $ is used before name.

example

#!/bin/bash

1.

```
echo "Enter name :  "
read name
echo "Entered name : $name“
```

2. # multiple variable

```
echo"Enter name :  "
read fname  sname
echo "Entered names : $fname, $sname ”
```

3.#input in same line with enter name??

```
read -p 'user name : ' user_val
echo "user name : $user_val“
```

How to make silent value like password..???

How to read array

```
#!/bin/bash
echo "entered name : "
read -a names
#print names
echo "Names : ${names[0]} ,names[1]} ,names[2]} "
```

```
1  #!/bin/bash
2
3  echo "entered name : "
4  read -a names
5  #print names
6  echo "Names are : ${names[0]},${names[1]}
       ,${names[2]} "
```

```
$bash -f main.sh

entered name :
Names are : Rachna,amish,karnavat
```

Variable

Shell script

#!/bin/bash

var1=PICT

var2=IT

echo" $var1 $var2"

```
1  #!/bin.bash
2
3  var1=PICT
4  var2=IT
5
6  echo "$var1 $var2"
7
```

output

- $ chmod =x test.sh
- $ PICT IT

Script

```
#!/bin/bash
# testing variables
days=5
guest=”Rachna”
echo “$guest checked in $days days ago”
days=2
guest=”Sonal”
echo “$guest checked in $days days ago”
```

Output

```
$ chmod u+x  test.sh
$ ./test.sh

Rachna  checked in 5days ago

Sonal  checked in 2 days ago
```

Shell Parameter

Parameters	Function
$1-$9	Represent positional parameters for arguments one to nine
${10}-${n}	Represent positional parameters for arguments after nine
$0	Represent name of the script
$*	Represent all the arguments as a single string
$@	Same as $*, but differ when enclosed in (")
$#	Represent total number of arguments
$$	PID of the script
$?	Represent last return code

Example

```
#!/bin/bash
#
echo this script name is $0
#
echo The first argument is $1
echo The second argument is $2
echo And third argument is $3
#
echo \$ $$ PID of the script
echo \# $# Total number of arguments
"script.sh" 14 lines, 264 characters
```

```
sssit@JavaTpoint:~$ chmod +x script.sh
sssit@JavaTpoint:~$ ./script.sh 1 5 90
this script name is ./script.sh
The first argument is 1
The second argument is 5
And third argument is 90
$ 2488 PID of the script
# 3 Total number of arguments
? 0 Last return code
* 1 5 90 All the arguments
sssit@JavaTpoint:~$
```

expr : Computation and String Handling

The shell relies on external **expr** command for computing features.

Functions of expr:

- Perform arithmetic operations on integers
- Manipulate strings

Computation

expr can perform four basic arithmetic operation as well as modulus function

Examples

$x=3; y=5
$expr 3 + 5
$expr $x - $y
$expr 3 * 5
$expr 3 / 5
$expr $y % $

```
#!/bin/bash

x=3; y=5
expr 3 + 5
expr $x - $y
expr 3 \* 5
expr 3 / 5
expr $y % $x
```

```
$bash -f main.sh

8
-2
15
0
2
```

Command su

z =`expr $

```
#!/bin/bash

x=3; y=5
z=`expr $x + $y`
echo $z
#Incrementing value of a variable
x=`expr $x + 1`
echo $x
```

```
$bash -f main.sh

8
4
```

Incrementing

x =`expr $x + 1`

Small script which you can do…

1. Define variable num with value 10 and print it on screen.

2. Define variable name with value PICT and print it on screen.

3. Print sum of two numbers, let's say 12 and 13 .

THE if CONDITIONAL

Form 1	Form 2	Form 3
if command is successful then execute command else execute command fi	if command is successful then execute command fi	if command is successful then execute command elif command is successful then …. else ….. fi

if command is successful ; then

if...else…fi -Example

```
script name: isnump_n
#!/bin/sh
#Script to see whether argument is
positive or negative
#
if [ $# -eq 0 ]
then
   echo "$0 : You must give/supply
   one integers"
   exit 1
fi
if test $1 -gt 0
then
   echo "$1 number is positive"
   else echo "$1 number is negative"
fi
```

Try it as follows:

$ chmod 755 isnump_n

$ isnump_n 5

5 number is positive

$ isnump_n -45

-45 number is negative

$ isnump_n

./ispos_n : You must give/supply one integers

$ isnump_n 0

0 number is negative

Mathematical operator..

Mathematical Operator in Shell Script	Meaning	Normal Arithmetical/ Mathematical Statements	But in Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

for and while loop

#code

```
#!/bin/sh
for i in 1 2 3 4 5
do
echo “Printing looping
... number $i“
 done
```

- Output

```
#!/bin/sh
for i in 1 2 3 4 5
do
echo “Printing looping  ... number
    $i“
 done
```

```
$bash -f main.sh

“Printing looping ... number 1“
“Printing looping ... number 2“
“Printing looping ... number 3“
“Printing looping ... number 4“
“Printing looping ... number 5“
```

```
#!/bin/bash
# for loops

for VARIABLE in 1 2 3 4 5 .. N
do
    command1
    command2
    commandN
done
#OR--------------------------------

for VARIABLE in file1 file2 file3
do
    command1 on $VARIABLE
    command2
    commandN
done
#OR--------------------------------------------

for OUTPUT in $(Linux-Or-Unix-Command-Here)
do
    command1 on $OUTPUT
    command2 on $OUTPUT
    commandN
done
```

```
#!/bin/bash
# for loops
echo ${BASH_VERSION}
for (( i=0; i<5; i++ ))
do
   echo $i
done
```

```
#!/bin/bash
# for loops

for i in {1..10}
do
   echo $i
done
```

```
#!/bin/bash
# for loops

for i in {1..10..2}
do
   echo $i
done
```

For loop with unix commands

```
#!/bin/bash
# for loops
for command in ls pwd date
do
  echo "---------------$command-----------------"
  $command
done
```

while loop

```
# while loops

while [ condition ]
do
    command1
    command2
    command3
done
```

```
#!/bin/bash
# while loops
n=1

while [ $n -le 10 ]
do
    echo "$n"
    n=$(( n+1 ))
done
```

```
1
2
3
4
5
6
7
8
9
10
test@test:~/Desktop$
```

```
#!/bin/bash
# while loops
n=1

while (( $n <= 10 ))
do
    echo "$n"
    (( ++n ))
done
```

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

Write a script for above output using while loop.

While loop

```
#!/bin/sh
a=0
while [ "$a" -lt 10 ]
do
      b ="$a“
      while [ "$b" -ge 0 ]
      do
            echo -n "$b " b=
            b=`expr $b -
      done
       echo a=`expr $a + 1`
done
```

```
1  #!/bin/sh
2
3  a=0
4  while [ "$a" -lt 10 ]    # this
       is loop1
5  do
6     b="$a"
7     while [ "$b" -ge 0 ]  # this
          is loop2
8     do
9        echo -n "$b "
10       b=`expr $b - 1`
11    done
12    echo
13    a=`expr $a + 1`
14 done
```

```
$bash -f main.sh

0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

Few example of while script..

```
1   #!/bin/bash
2   # while loops
3   n=1
4
5   while [ $n -le 3 ]
6   do
7       echo "$n"
8       (( n++ ))
9       gnome-terminal &
10  done
11
```

```
#!/bin/bash
# while loops
n=1

while [ $n -le 3 ]
do
    echo "$n"
    (( n++ ))
    xterm &
done
```

Read file using while loop

```
#!/bin/bash
# while loops


while read p
do
    echo $p
done < hello.sh
```

```
#!/bin/bash
# while loops


cat hello.sh | while read p
do
    echo $p
done
```

Until loop

```
until [ condition ]
do
   command1
   command2
   ...
   ....
   commandN
done
```

```
n=1

until [ $n -ge 10 ]
do
   echo $n
   n=$(( n+1 ))
done
```

Case statement

```
#! /bin/bash

case expression in
    pattern1 )
        statements ;;
    pattern2 )
        statements ;;
    ...
esac
```

```
vehicle=$1

case $vehicle in
    "car" )
        echo "Rent of $vehicle is 100 dollar" ;;
    "van" )
        echo "Rent of $vehicle is 80 dollar" ;;
    "bicycle" )
        echo "Rent of $vehicle is 5 dollar" ;;
    "truck" )
        echo "Rent of $vehicle is 150 dollar" ;;
    * )
        echo "Unknown vehicle" ;;
esac
```

```
#! /bin/bash

echo -e "Enter some character : \c"
read value


case $value in
    [a-z] )
        echo "User entered $value a to z" ;;
    [A-Z] )
        echo "User entered $value A to Z" ;;
    [0-9] )
        echo "User entered $value 0 to 9" ;;
    ? )
        echo "User entered $value special character" ;;
    * )
        echo "Unknown input" ;;
esac
```

Select …in comaand

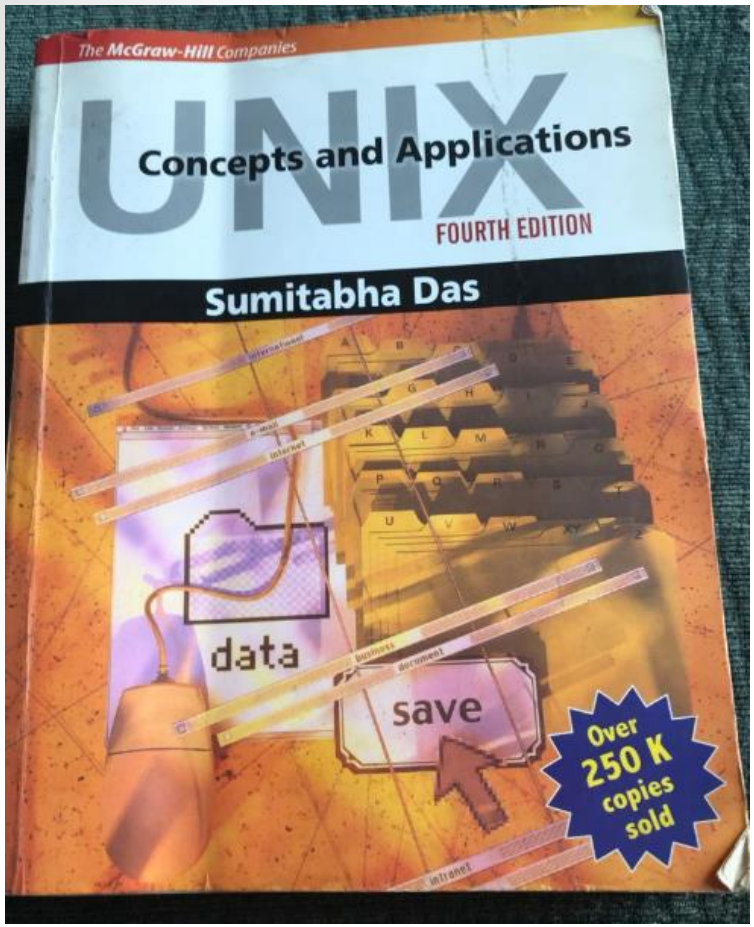
```
1  #!/bin/bash
2   select name in Rachna Vishal Sonal Tushar
3   do
4     echo "$name selected "
5
6  done
```

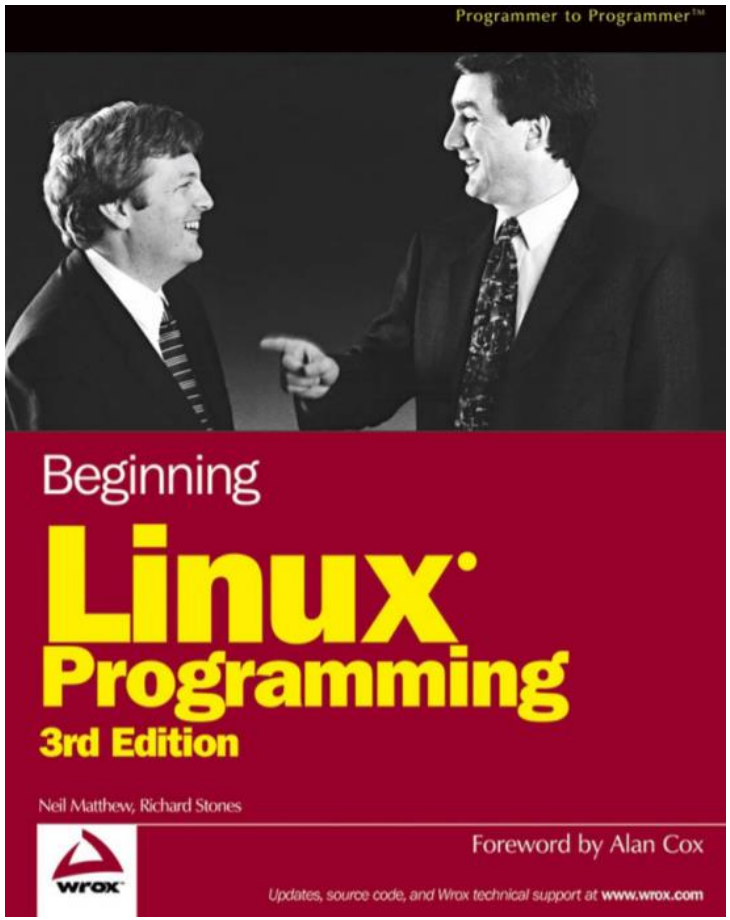
Practice Assignments

- Ramesh's basic salary is input through the keyboard. His dearness allowance is 40% of basic salary, and rent allowance is 20% of basic salary. Write a program to calculate his Gross salary??

- If a five digit number is input through the keyboard, write a program to calculate the sum of its digits

Reference books available in library





Thank You !

Connect me at
rrchhajed@pict.edu