United International University
QUEST FOR EXCELLENCE

Lab Manual

OPERATING SYSTEM CONCEPTS LABORATORY

CSI 310

Contents

Lab 01: Basic Linux Shell Commands

Topics to cover	Shell commands on manual, user, editor, view text, directory and file permissions, directory and file operations, file view commands, standard I/O/E, redirection, piping
Online	N/A
Assignment	N/A

COMMAND DETAILS

Introduction

- Many people says that Linux is a command based operating system.
- So many of us thinks that Linux is not so user friendly OS.
- But it is not true. Linux is a GUI based OS with a Shell which is more powerful than its counterpart in Windows OS.
- We will be familiar with some shell commands.

Identity

- Type ***uname*** and Linux will tell his name to you
- If you want to know your name type ***whoami***

Manual

- For each command Linux contains manual. To view the manual : ***man*** name
    - ***man uname***

Editors

- To view files a large number of editors are available. They are:
    - kwrite
    - emacs
    - gedit
    - vi
- To view : ***editorname filename***
    - ***kwrite file.txt***

User

- In Linux, root is the most powerful user. But other users can be created easily. Each linux user must be under certain group.
  - To add a group : ***groupadd group1***
  - To delete a group : ***groupdel group1***
  - To add a user : ***useradd –g groupname username***
  - To delete a user : ***userdel username***
  - To change a user : ***su user1***
  - To update the passwd : ***passwd user1***

View Text

- To view a line of text in the shell: ***echo***
  - ***echo 'welcome to linux'***
- To clear the shell : ***clear***

Directory and File Permissions

- Each file or directory has 3 security groups.
  - Owner
  - Group
  - All Others
- Each security group has 3 flags that control the access status : read, write, execute
- They are listed as 'rwx' or a "-" if the access is turned off.
  - ***rwxrwxrwx*** [read, write and executable for owner, group and all others]
  - ***rw-r--r--*** [read and write by owner, read only for group and all others]
- To change the permissions type ***chmod***
  - u, g, o or all [whose permission you are changing]
  - + or - [type of change: add or subtract permission]
  - combination of r , w or x [which permission you are changing: read, write or execute]

- file or directory [name of file or directory to change]
  - ✓ ***chmod go+rw file1 file2*** add read and write access for group and others for files 'file1' and 'file2'
  - ✓ ***chmod a+rwx file1*** add read, write and execute for everyone for 'file1'.
  - ✓ ***chmod 555 file1***

- ➢ To change the owner of a file or directory type ***chown***.

  chown username <file or directory>
  - ***chown user1 file***

- ➢ To change the group of a file or directory type ***chgrp***.
- ➢ ***chgrp*** groupname <file or directory>
  - ***chgrp group1 file1 file2***

Directory and File Listings

- ➢ To list information about directory or files : ***ls***
- ➢ This command contains some options.
  - -***a*** [do not hide entries starting with .]
  - -***A*** [do not list implied . and ..]
  - -***h*** [print sizes in human readable format]
  - -***l*** [use a long listing format]
  - -***S*** [sort by file size]

Directory Operations

- ➢ To print the current directory : ***pwd***
- ➢ To change the current directory : ***cd dirname***
  - The variable HOME is the default directory.
- ➢ To make a new directory : ***mkdir***
  - -m [set permission mode (as in chmod)]

  - -v [print a message for each created directory
- To delete an empty directory : ***rmdir***

- To move to a directory pushing the current directory to stack : ***pushd dirname***
- Effect:
  - adds a directory to the top of the directory stack
  - or rotates the stack making the new top of the stack the current working directory
- To moves to the directory at the top of the stack as well as to remove the topmost entry : ***popd***
- Effect:
  - removes the top directory from the stack
  - performs a ***cd*** to the new top directory.

- To display the list of currently remembered directories : ***dirs***
- The default display is on a single line with directory names separated by spaces.
- How to add to the list : ***pushd***
- How to remove from the list : ***popd***

File Operations

- To copy a file : ***cp***
- Copy source to destination or multiple sources to directory
  - -***i*** [prompt before overwrite]
  - -***r*** [copy directories recursively]
  - -***u*** [copy only when the src file is newer than the dest file or when the dest file is missing

- To remove a file or directory : ***rm***
  - -***f*** [ignore nonexistent files, never prompt]
  - -***i*** [prompt before any removal]
  - -***r*** [remove the contents of directories recursively]
  - -***v*** [explain what is being done]

- To move or rename a file : ***mv***
  - rename src to dest or move src(s) to directory
  - -***i*** [prompt before overwrite]
  - -***u*** [move only when the src file is newer than the dest file or when the dest file is missing
  - -***v*** [explain what is being done]

- To determine file type : ***file filename***
- File tests each argument in an attempt to classify it. This causes the file type to be printed
  - - ***i*** [show the mime type].
  - -***v*** [Print the version of the file]
    1. ***file a.txt*** : a.txt: very short file
    2. ***file a.xls*** : a.xls: Microsoft Office Document
    3. ***file -i a.xls*** : a.xls: \012- application/msword

- To concat files and print on the standard output : ***cat file1 file2 file3 …***
  - -***n*** [number all output lines]
  - -***s*** [never more than one single blank line]

File Viewing

- To view files in shell use: ***more*** or ***less***.
  - ***more filename***
  - ***less filename***
- The main difference between more and less is that
  - less allows backward and forward movement using the arrow keys.
  - more only uses the [Spacebar] and the [B] key for forward and backward navigation.

- To output the first lines of files : ***head file1 file2 file3 …***
- Print the first 10 lines of each file to standard output
- With more than one file , precede each with a header giving the file name
  - -***n*** [output the last n lines, instead of the last 10]

- To output the last lines of files : ***tail file1 file2 file3 …***
- Print the last 10 lines of each file to standard output
- With more than one file, precede each with a header giving the file name
  - -***n*** [output the last n lines, instead of the last 10]

- To sort lines of a text files : ***sort file1 file2 file3…***
- Write sorted concatenation of all file(s) to standard output.

- To print the number of lines, words and bytes in files : ***wc file1 file2 file3 …***
- print byte, word, and newline counts for each file and a total line if more than one file is specified.
  - -***l*** [print the newline counts]
  - -***w*** [print the word counts]

Standard I/O/E

By default, three default files known as standard files are automatically opened when a command is executed.

They are standard input (***stdin***) ,standard output (***stdout***) and standard error (***stderr***).

For example, the command ***ls -a*** scans the current directory and collects a list of all the files,

produces a human readable list, and outputs the result to the terminal window.

Redirection

- Linux redirection features can be used to detach the default files from ***stdin***, ***stdout*** and
- ***stderr*** and attach other files to them.
- **Input redirection**:
  - < - get input from file instead of the keyboard
- **Output redirection**:
  - > - send output to file instead of the terminal window
- **Append output**:
  - >> - command is used to append to a file if it already exists

Piping

- The input of a command may come from the output of another command.
- This is accomplished with the ' | ' pipe operator.

- **<u>How to view the lines 15-20 of a file named 'a.txt' ?</u>**

- The input of a command may come from the output of another command.
- This is accomplished with the ' | ' pipe operator.

- **<u>How to view the lines 15-20 of a file named 'a.txt' ?</u>**

  - ***head -20 a.txt | tail -5***

Lab 02: Linux Shell Commands on Find and Grep

Topics to Cover	find and grep Command with regular expression

Online	See sample online 1
Assignment	N/A
External Link	Linux grep command https://www.computerhope.com/unix/ugrep.htm Regular expression https://www.computerhope.com/unix/regex-quickref.htm

SAMPLE ONLINE: 01

Consider two task files provided you on elms. Perform following operations using those files where required:

1. create a file (myfile) which is inside the folder (myfolder1) that is inside another folder (myfolder2) in C drive.
2. combine taskfile1 and taskfile2 contexts in myfile and then show myfile content.
3. count the total number of lines, words, characters in taskfile1 and taskfile2.
4. count the total number of lines, words, characters in taskfile1 and taskfile2, and print the info in myoutput.txt
5. sort the contexts of taskfile1 and taskfile2 in reverse alphabetic order
6. sort the contexts of taskfile1 and taskfile2, and then copy the 3-10 lines to myfile, then show the content of myfile.
7. print your name in command prompt, and copy that name and today's date in myoutput2.txt
8. print your name in command prompt but copy only today's date in myoutput3.txt
9. delete output.txt
10. delete myfolder1

Task file 01

```
August 25, 1991 Dear friend,
 I am writing to you because she said you listen and understand
and didn’t try to sleep with that
person at that party even though you could have. Please don’t try
to figure out who she is because
then you might figure out who I am, and I really don’t want you to
do that. I will call people by
different names or generic names because I don’t want you to find
me. I didn’t enclose a return address
for the same reason. I mean nothing bad by this. Honest.
 I just need to know that someone out there listens and
understands and doesn’t try to sleep with
people even if they could have. I need to know that these people
exist.
 I think you of all people would understand that because I think
you of all people are alive and
appreciate what that means. At least I hope you do because other
people look to you for strength
```

```
and friendship and it’s that simple. At least that’s what I’ve
heard.
 So, this is my life. And I want you to know that I am both happy
and sad and I’m still trying to
figure out how that could be.
 I try to think of my family as a reason for me being this way,
especially after my friend Michael
stopped going to school one day last spring and we heard Mr.
Vaughn’s voice on the loudspeaker.
 “Boys and girls, I regret to inform you that one of our students
has passed on. We will hold a
memorial service for Michael Dobson during assembly this Friday.”
 I don’t know how news travels around school and why it is very
often right. Maybe it was in the
lunchroom. It’s hard to remember. But Dave with the awkward
glasses told us that Michael killed
himself. His mom played bridge with one of Michael’s neighbors and
they heard the gunshot.
```

Task file 02

```
Fat Kid Rules the World by K. L. Going
Nailed by Patrick Jones
How to Say Goodbye in Robot by Natalie Standiford
Rats Saw God by Rob Thomas
Nick and Norah's Infinite Playlist by David Levithan and Rachel
Cohn
Story of a Gril by Sara Zarr
I am the messenger  by Zusak
Born to Rock by Korman
Smack by Burgess
Such a Pretty Girl by Weiss
Getting the Girl by Zusak
Hard Love by Wittlinger
Repossessed by Jenkins
You Don’t Know Me by Klass
Wasteland by Block
It’s Kind of a Funny Story by Vizzini
```

COMMAND DETAILS

Grep

- grep matches a pattern in a given a list of files or standard input and outputs only the matching lines.
    - ***grep*** pattern filename
        - ✓ ***grep abc file.txt***
- grep patterns are case sensitive by default.
- Some options
    - ***-i*** [case insensitive search]
    - ***-c*** [count of total matches]
    - ***-E*** [regular expressions can be provided as patterns]
    - ***-n*** [display the line numbers of the matched lines]

Find

- search for files in a directory hierarchy.
- By default, find returns all files below the current working directory.
    - ***find***
- To search a pattern : ***find -name '*txt*'***
- To search for a file type :
    - ***find -type d*** [find all directories]
    - ***fine -type f*** [find all regular files]
- Find executes the '***-print***' action by default. To change it to style such as '***ls***' : ***find -type f –ls***

- To search all the directories
    - not recommended
    - ***find / -name "myfile" -type f***
- To search a specific directory
    - ***find /home/dir1 -name "myfile" -type f***
- To search multiple directories
    - ***find dir1 dir2 -name "myfile" -type f***
- To Search for all files owned by a user
    - ***find -user userid***
- To take an action
    - ***find -type f -name '*ch*' -exec chmod a+rwx {} \;***
    - ***{}*** is replaced with the name of the file
    - The ***;*** indiates the end of the command.

Lab 03: Shell Command

Topics to cover	N/A
Online	See sample online 02
Assignment	See sample assignment 01

Resource Link	N/A

SAMPLE ONLINE: 02

Obtain the file Grepdata.txt.

Once you have the file, write a series of grep statements that do the following:

1. Print all lines that contain a phone number with an extension (the letter x or X followed by four digits).
2. Print all lines that begin with three digits followed by a blank. Your answer must use the \{ and \} repetition specifier.
3. Print all lines that contain a date. Hint: this is a very simple pattern. It does not have to work for any year before 2000.
4. Print all lines containing a vowel (a, e, i, o, or u) followed by a single character followed by the same vowel again. Thus, it will find "eve" or "adam" but not "vera". Hint: \(and \)
5. Print all lines that do not begin with a capital S.
6. Write grep statements that use command-line options along with the pattern to do the following:
7. Print all lines that contain CA in either uppercase or lowercase.
8. Print all lines that contain an email address (they have an @ in them), preceded by the line number.
9. Print all lines that do not contain the word Sep. (including the period).
10. Print all lines that contain the word de as a whole word.

Save these statements in a file named in the form

lastname_firstname_grep1.sh
and email it to me. So, for example, if your name is Joe Doakes, your file would be named doakes_joe_grep1.sh.

Your patterns should work in any generic file of this sort. They should not be dependent upon the data in this particular file; if I add more lines of the same form to the file, your patterns should still work.

Grepdata.txt file

```
Sep. 17, 2013
Esperanza High School
1830 N. Kellog Dr.
Anaheim, CA 92807-1281
Steve Marshal
714-555-7870 X7310
aztecwrestling@example.com
Brian Fortenbaugh
714-555-7870 x7309
```

```

Sep. 24, 2013
Sonora High School
401 S. Palm St.
La Habra, CA 90631
Carl Hohl (aka Krazy Rabbit)
562-555-9800

Oct. 1, 2013
```

```
Lakewood High School
440 Briercrest Ave.
Lakewood, CA 90713-2013
Andy Miramontes
562-555-1281

Oct. 8, 2013
North Torrance High School
2013 W. 182nd
Torrance, CA 90504
Don Henderson
310-555-4412

Oct. 15, 2013
```

```
El Dorado High School
1651 N. Valencia Ave.
Placentia, CA 90631
Steve Lawson
714-555-5350 x2134
Lawsonhawk@example.com

Nov. 12, 2013
Rosemead High School
9063 E. Sepulveda Dr.
Rosemead, CA 91770
Daren de Heras
daren103@example.com
```

SAMPLE ASSIGNMENT: 01

Similar to online 1 and 2

Lab 04: Linux Shell Script

Topics to Cover	Shell scripts basics like file extension, executing etc. Variables, read, if-else
Online	N/A
Assignment	N/A
Resource link	https://drive.google.com/file/d/1VdCYb2_B9gXPvnTtvmuIZkVb43efbzze/view?usp=sharing

WHAT IS SHELL SCRIPT?

- We have seen some basic shell commands, it's time to move on to scripts.
- There are two ways of writing shell programs.
    - You can type a sequence of commands and allow the shell to execute them interactively.
    - You can store those commands in a file that you can then invoke as a program. This is known as Shell Script.
- We will use bash shell assuming that the shell has been installed as /bin/sh and that it is the default shell for your login.

WHY SHELL SCRIPT?

- Shell script can take input from user, file and output them on screen.
- Useful to create own commands.
- Save lots of time.
- To automate some task of day today life.
- System administration part can be also automated.

HOW TO WRITE AND EXECUTE?

- Use any editor to write shell script.
- The extension is ***.sh***.
- After writing shell script set execute permission for your script.

  chmod +x script_name
- Execute your script

  ./script_name

SHELL SCRIPT FORMAT

- Every script starts with the line

  #!/bin/bash
- This indicates that the script should be run in the bash shell regardless of which interactive shell the user has chosen.
- This is very important, since the syntax of different shells can vary greatly.
- # is used as the comment character.
- A word beginning with # causes that word and all remaining characters on that line to be ignored.

A sample shell script

```
#!/bin/bash
   echo "Hello User"
   echo "See the files in current directory"
ls
```

VARIABLES

- In Linux (Shell), there are two types of variable:
- System variables - created and maintained by Linux itself.
    - ***echo $USER***
    - ***echo $PATH***

User defined variables - created and maintained by user.

- All variables are considered and stored as strings, even when they are assigned numeric values.
- Variables are case sensitive.

- When assigning a value to a variable, just use the name.
- No spaces on either side of the equals sign.

  var_name=value
- Within the shell we can access the contents of a variable by preceding its name with a **$**.

myname=A [use quotes if the value contains spaces]

myos=Linux

text = **1+2**

echo Your name:$myname [*A*]

echo Your os:$myos [*Linux*]

echo $text [*1+2*]

- If you enclose a $variable expression in double quotes, it’s replaced with its value when the line is executed.
- If you enclose it in single quotes, no substitution takes place.

You can also remove the special meaning of the $ symbol by prefacing it with a \.

myvar=”Hello”

echo $myvar [Hello]

echo “$myvar” [Hello]

echo ‘$myvar’ [$myvar]

echo \$myvar [$myvar]

READ

To read user input from keyboard and store it into a variable use ***read var1,var2,.....varn***

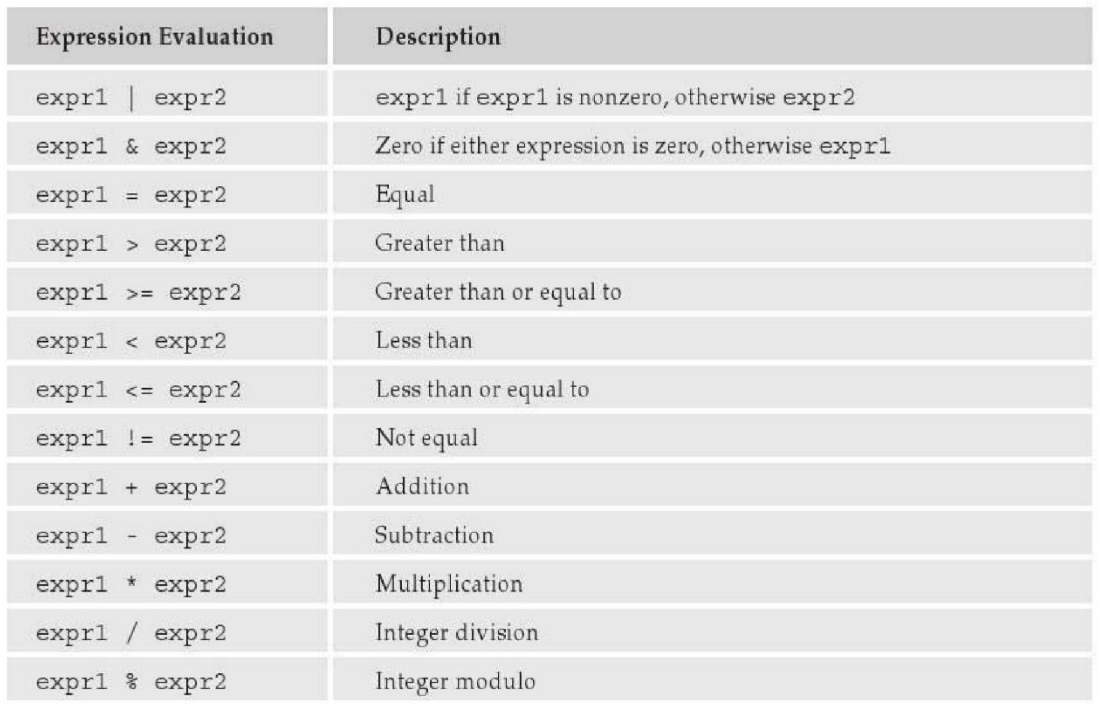
```
#!/bin/bash
echo -n "Enter your
name:”
read name
echo -n "Enter your student no:”
read stdno
echo "Your Name:$name”
echo "Your Age:$stdno”
```

SHELL ARITHMETIC

- The ***expr*** command evaluates its arguments as an expression.
- It is commonly used for simple arithmetic operations.

```
#!/bin/bash
expr 1 + 1
expr 1 - 1
expr 1 \* 1
expr 1 / 1
var=`expr 1 + 1`
x=1
x=`expr $x + 1`
```

Expression Evaluation



Expression Evaluation	Description
`expr1 | expr2`	expr1 if expr1 is nonzero, otherwise expr2
`expr1 & expr2`	Zero if either expression is zero, otherwise expr1
`expr1 = expr2`	Equal
`expr1 > expr2`	Greater than
`expr1 >= expr2`	Greater than or equal to
`expr1 < expr2`	Less than
`expr1 <= expr2`	Less than or equal to
`expr1 != expr2`	Not equal
`expr1 + expr2`	Addition
`expr1 - expr2`	Subtraction
`expr1 * expr2`	Multiplication
`expr1 / expr2`	Integer division
`expr1 % expr2`	Integer modulo

IF-ELSE

```
if [ conditiong1 ]; then
        statement1
elif [ condition2 ]; then
        statement2
else
        statement3
fi
```

- It is must to put spaces between the [braces and the condition being checked.
- If you prefer putting then on the same line as ***if***, you must add a semicolon to separate the test from the ***then***.

Comparison Rules

String Comparison	Result
`string1 = string2`	True if the strings are equal.
`string1 != string2`	True if the strings are not equal.
`-n string`	True if the string is not null.
`-z string`	True if the string is null (an empty string).

Arithmetic Comparison	Result
`expression1 -eq expression2`	True if the expressions are equal.
`expression1 -ne expression2`	True if the expressions are not equal.
`expression1 -gt expression2`	True if `expression1` is greater than `expression2`.
`expression1 -ge expression2`	True if `expression1` is greater than or equal to `expression2`.
`expression1 -lt expression2`	True if `expression1` is less than `expression2`.
`expression1 -le expression2`	True if `expression1` is less than or equal to `expression2`.
`! expression`	True if the expression is false, and vice versa.

File Conditional

File Conditional	Result
`-d file`	True if the file is a directory.
`-e file`	True if the file exists. Note that, historically, the -e option has not been portable, so -f is usually used.
`-f file`	True if the file is a regular file.
`-g file`	True if set-group-id is set on `file`.
`-r file`	True if the file is readable.
`-s file`	True if the file has nonzero size.
`-u file`	True if set-user-id is set on `file`.
`-w file`	True if the file is writable.
`-x file`	True if the file is executable.

```
#!/bin/bash
  echo "Enter first number "
  read num1
  echo "Enter second number"
  read num2
  if [ $num1 -gt $num2 ] ; then
        echo "$num1 is greater than $num2"
  elif [ $num1 -lt $num2 ] ; then
        echo "$num1 is less than $num2"
  else
        echo "$num1 and $num2 are equal"
  fi
```

Lab 05: Linux Shell Script

Topics to Cover	Case, For, While, Until loop Functions
Online	N/A
Assignment	Sample Assignment 02
Resource link	https://drive.google.com/file/d/1VdCYb2_B9gXPvnTtvmuIZkVb43efbzze/view?usp=sharing

SAMPLE ASSIGNMENT: 02

Q1. How many vowels are there in the 7th line of file1? Write a shell script to get the answer.
Q2. Write a shell script that takes input names of three text files, sorts lines of first and last file in lexicographic order and the 2nd file in reverse lexicographic order. Your script should print the first word and last word of each line from the combined file and counts total number of words.
Q3. Write a shell script that takes two string in the command prompt, makes a folder with the first argument, creates a text file within the folder and then takes three line as input and appends it to the file.
Q4. Find all words of the file2 that has a repetition of character.
Q5. Write a shell script that asks you for a command name and generates a file in the home

Note: files will be provided in the class

CASE

```
case $var in
        condition1) statement ;;
        condition2) statement ;;
        *) statement3
esac
```

- Notice that each pattern line is terminated with double semicolons ;; .

You can put multiple statements between each pattern and the next, so a double semicolon is needed to mark where one statement ends and the next pattern begins

Example 01 (case)

```
#!/bin/sh
echo "Is it morning? Please answer yes or no" read timeofday
case "$timeofday" in

     yes) echo "Good Morning";;
     no ) echo "Good
     Afternoon";;

          y )   echo "Good Morning";;

          n )   echo "Good Afternoon";;

          * )   echo "Sorry, answer not recognized";;

esac
```

....

Example 02 (case)

```
#!/bin/sh
    echo “Is it morning? Please answer yes or no” read
    timeofday
    case “$timeofday” in
        yes | y | Yes | YES )    echo “Good Morning”;;
        n* | N* )                echo “Good Afternoon”;;
        * )                      echo “Sorry, answer not recognized”;;
    esac
```

COMMAND LINE ARGUMENTS

- Command line arguments can be passed to the shell scripts. There exists a number of built in variables
    - **$*** - command line arguments
    - **$#** - number of arguments
    - **$n** - nth argument in $*
- ***./script_name arg1 arg2 .... argn***

FOR

```
for variable in list
    do
              statement
    done
for (( expr1; expr2; expr3 ))
        do
statement
done
```

Example 01 (for)

```
#!/bin/bash

echo "the number of args is $#"
a=1
for i in $*
do
        echo "The $a No arg is $i" a=`expr $a + 1`
done
```

Example 02(for)

```
#!/bin/bash
for i in `ls`
do
        echo $i
done
```

Example 03(for)

```
for(( i=0;i<=50;i++))
do
          echo $i
done
```

WHILE

```
while condition do
        statements
    done
```

Example 01 (while)

```
#!/bin/bash
password="abc"
```

```
echo "Enter password"
read pass
while [ $pass != $password ]
do
 echo "Wrong Password,Try again"
 read pass
done
echo "Write Password"
```

UNTIL

```
until condition do
    statements
done
```

Example 01 (until)

```
#!/bin/bash
password="abc"
echo "Enter password"
read pass
until [ $pass = $password ]
do
 echo "Wrong Password, Try again"
 read pass
done
echo "Write Password"
```

FUNCTIONS

- Functions can be defined in the shell and it is very useful to structure the code.
- To define a shell function simply write its name followed by empty parentheses and enclose the statements in braces.

```
function_name () {
statements
}
```

- Function must be defined before one can invoke it.

Example 01 (function)

```
 #!/bin/sh
foo() {
        echo “Function foo is executing”
}
echo “script starting”
foo
echo “script ending”
```

Output

```
script starting
Function foo is executing
script ending
```

- When a function is invoked, the parameters to the script ***[$*, $#, $1, $2]*** and so on are replaced by the parameters to the function.
- When the function finishes, they are restored to their previous values.

Example 02 (function)

```
#!/bin/bash
showarg()
{
a=1
for i in $*
do
echo "The $a No arg is $i"
a=`expr $a + 1`
done
}
echo "Listing start"
showarg $*
echo "Total:$#"
echo "Listing End"
```

- Functions can return numeric values using the return command.
- Functions can also return strings by the following ways.

Qee

Example 03 (function)

```
f(){ var="123“; }
f
echo $var
```

Example 04 (function)

```
f(){ echo "123"; }
result="$(f)"
```

Example 05 (function)

```
#!/bin/sh
yes_or_no()
{
echo "Is your name $* ?"
echo "Enter yes or no:"
read x
case “$x” in
y | yes ) return 0;;
n | no ) return 1;;
esac
}
if yes_or_no “$1”
then
echo “Hi $1, nice name”
else
echo “Never mind”
fi
```

....

Be careful. Function calling can be recursive.

```
f()
{
statements
f
}
f
```

The parameter must be passed every time a function is invoked either from main or from any other functions

Lab 06: Linux Shell Script

Topics to Cover	N/A
Online	Similar to Sample Assignment 02
Assignment	N/A
Resource link	N/A

Lab 07: Mid Term Exam

Syllabus	Shell Command and Shell Script
Full Marks	Will be scaled to 15%
Exam type	Coding(Practical)
Resource link	Faculty may provide previous year's questions for practice

Lab 08: Scheduling Assignment

Topics to cover	Several scheduling algorithms of Batch System and Interactive System like FCFS, preemptive and non-preemptive SJF, Round-robin etc.
Assignment	An assignment will be given on implementing a specific scheduling assignment. See Sample Assignment 03.
Presentation group	Students will form presentation group (2/3 members per group) for lab 11.

SAMPLE ASSIGNMENT: 03

Statement	Implement FCFS scheduling algorithm and print performance statistics (average turnaround time) after certain time interval.

Solution to sample assignment 03

FCFS.java

```
package fcfs;

import java.util.Comparator;
import java.util.PriorityQueue;

public class FCFS{

    /**
     * @param args the command line arguments
     */
     static PriorityQueue<Process> processQueue = new PriorityQueue<Process>(10, new
Comparator<Process>() {
        public int compare(Process process1, Process process2) {
            return (int)(process1.getArrivalTime()-process2.getArrivalTime());
        }
    });

      static PriorityQueue<Process> readyQueue = new PriorityQueue<Process>(10, new
Comparator<Process>() {
        public int compare(Process process1, Process process2) {
            return (int)(process1.getArrivalTime()-process2.getArrivalTime());
        }
    });
     static GlobalTimer globalTimer = new GlobalTimer(0);

    public static void main(String[] args) {

         processQueue.add(new Process(1,3,2,globalTimer));

         processQueue.add(new Process(2,6,3,globalTimer));

         processQueue.add(new Process(3,1,4,globalTimer));

         processQueue.add(new Process(4,4,5,globalTimer));

        while(true){

            if(checkIfNewProcessArrived()){

                while(!processQueue.isEmpty() && processQueue.element().getArrivalTime()<
globalTimer.time){

                        readyQueue.add(processQueue.poll());
                }
            }

            if(!readyQueue.isEmpty())
                    runProcessInCpu();
            else {
```

```
            System.out.println("No process is in Ready Queue");
            System.out.println("Global time: "+globalTimer.time);
            globalTimer.time++;

          }
       }
    }

    public static boolean checkIfNewProcessArrived(){

        if(!processQueue.isEmpty()){

           if(processQueue.element().getArrivalTime()<globalTimer.time)
              return true;
        }
        return false;
    }

    public static void runProcessInCpu(){
         Process = readyQueue.poll();
         process.runProcess();
    }
 }
```

Process.java

```
package fcfs;
public class Process {
   int id;
   int arrivalTime;
   int duration;
   GlobalTimer globalTimer;

   public Process(int id,int arrivalTime, int duration, GlobalTimer globalTimer){
      this.id=id;
      this.arrivalTime=arrivalTime;
      this.duration=duration;
      this.globalTimer=globalTimer;
   }

   public void runProcess(){
      for(int i=0;i<duration;i++){
         System.out.println("My process ID: "+id);
         System.out.println("Global time: "+globalTimer.time);
         globalTimer.time++;
       }

      System.out.println("********Process Id: "+id + " completed its job********");
   }
```

```
    public int getId() {
        return id;
    }

    public int getArrivalTime() {
        return arrivalTime;
    }

    public int getDuration() {
        return duration;
    }

    public GlobalTimer getGlobalTimer() {
        return globalTimer;
    }
}
```

GlobalTimer.java

```
package fcfs;
public class GlobalTimer {

    int time;

    public GlobalTimer(int initialTime){

        this.time=initialTime;
    }

}
```

Lab 09: IPC Assignment

Evaluation + viva	Assignment 03(scheduling)
Topics to cover	Semaphore and it's usage in IPC
Assignment	An assignment will be given on inter-process-communication. See Sample Assignment 04.
Presentation topic	Students will submit their chosen presentation topic.

SAMPLE ASSIGNMENT: 04

Statement	Implement producer-consumer ipc problem
Discussion	In computing, the producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened. The problem can also be generalized to have multiple producers and consumers.

Solution to sample assignment: 04

Producer.java

```
package producerconsumer;

import java.util.Queue;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Producer implements Runnable{

    Semaphore mutex = null;
    Semaphore empty = null;
    Semaphore full = null;
    Queue<Integer> buffer=null;
    Random rand = new Random();

    public Producer(Semaphore mutex,Semaphore empty,Semaphore
full,Queue<Integer> buffer){
        this.mutex=mutex;
        this.empty=empty;
        this.full=full;
        this.buffer=buffer;
    }

    @Override
    public void run() {
        int item;
        while(true){
            try {
                item = rand.nextInt(50) + 1;
                empty.down();
                mutex.down();
                buffer.add(item);
                System.out.println(Thread.currentThread().getName()+" produced: " + item);
                mutex.up();
                full.up();

                Thread.sleep(rand.nextInt(30) + 1);
            } catch (InterruptedException ex) {
                Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
```

Consumer.java

```
import java.util.logging.Logger;

public class Consumer implements Runnable{

    Semaphore mutex = null;
    Semaphore empty = null;
    Semaphore full = null;
    Queue<Integer> buffer=null;
    Random rand=new Random();

    public Consumer(Semaphore mutex,Semaphore empty,Semaphore full,Queue<Integer>
buffer){
        this.mutex=mutex;
        this.empty=empty;
        this.full=full;
        this.buffer=buffer;
    }
    @Override
    public void run() {

        while(true){
            int item;
            try {
```

```
            full.down();
            mutex.down();
            item=buffer.remove();
            System.out.println(Thread.currentThread().getName()+" consumed: " + item);
            mutex.up();
            empty.up();
            Thread.sleep(rand.nextInt(10) + 1);
          } catch (InterruptedException ex) {
            Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);
          }

      }
    }

}
```

Semaphore.java

```
package producerconsumer;

public class Semaphore {

  private int initialValue = 0;
  private int bound = 0;
  private String name = null;
  public Semaphore(int initialValue,int upperBound, String name){
    this.bound = upperBound;
    this.initialValue = initialValue;
    this.name = name;
  }

  public synchronized void up() throws InterruptedException{
    while(this.initialValue == bound){
        System.out.println(Thread.currentThread().getName()+" is waiting in up for " + name +":");
        wait();
    }
    this.initialValue++;
    System.out.println(Thread.currentThread().getName()+" performed up for " + name +":");
    this.notify();
  }

  public synchronized void down() throws InterruptedException{
    while(this.initialValue == 0){
         System.out.println(Thread.currentThread().getName()+" is waiting in down " + name +":");
         wait();
    }
    this.initialValue--;
    System.out.println(Thread.currentThread().getName()+" peformed down for " + name +":");
    this.notify();
  }
}
```

ProducerConsumer.java

```
package producerconsumer;

import java.util.LinkedList;
import java.util.Queue;

public class ProducerConsumer {

    /**
     * @param args the command line arguments
     */
       static int N =5;
       public static Semaphore mutex =new Semaphore(1,1, "mutex");
       public static Semaphore empty =new Semaphore(N,N, "empty");
       public static Semaphore full =new Semaphore(0,N, "full");
       public static Queue<Integer> buffer = new LinkedList<Integer>();


    public static void main(String[] args) {
        Producer producer1=new Producer(mutex,empty,full,buffer);
        Producer producer2=new Producer(mutex,empty,full,buffer);

        Consumer consumer1=new Consumer(mutex,empty,full,buffer);
        Consumer consumer2=new Consumer(mutex,empty,full,buffer);

        new Thread(producer1,"Producer 1").start();
        new Thread(consumer1,"Consumer 1").start();
        new Thread(producer2,"Producer 2").start();
        new Thread(consumer2,"Consumer 2").start();
    }

}
```

Lab 10: IPC Assignment Evaluation

Evaluation + viva	Assignment 04(IPC)
Presentation	Students will notify faculty member slide update and ask for suggestion if needed.

Lab 11: Presentation

Will be scaled to 10%

Lab 12: Final Exam

Syllabus	All topics covered throughout the course
Full Marks	Will be scaled to 25%
Exam type	Written(quiz)