SHAHEED ZULFIQAR ALI BHUTTO UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Computer Science

Introduction to Operating Systems
Laboratory Manual

Course Code: CSCL2205
Semester: Spring 2019

Prepared by:
Department of Computer Science
SZABIST Karachi.

100-C, Block 5 Clifton, Karachi, Pakistan.
Phone: 111-922-478 | (021) 35823433
Website: www.szabist.edu.pk

Subject: Operating Systems (CSCL2205)
Contact Hours: 3

Course Outline

The course outline suggested for Semester Spring of year 2019 is as follows:

Week/Session #	**Topic**
1	Installation and Overview.
2	Commands (basic and file related).
3	More Commands.
4	Users, Groups and Privileges.
5	Introduction to Bash and Scripting.
6	Structures and Programs.
7	Administration tasks in Unix.
8	**Mid-term Examination.**
9	Permissions and Shared Directories.
10	Processes and SJF algorithm
11	Processes and FCFS algorithm
12	Process Management and Scheduling
13	Package in UNIX & CentOS 6 systems.
14	Network in CentOS 6 systems.
15	Project Task I & II
16	**Final Exam**

Note: Each week/session comprises of one (1) credit hour or three (3) credit hours of Lab.

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Lab – 1 & 2
Installation, Overview and Commands

Objective:

Note: On the completion of this lab, student is able to answer the above-mentioned points. If you are unable, ask lab instructor about your queries.

Hardware Requirements:

1. Can run on very minimal hardware. Recommend that computer have minimum of:
2. 1 GB of free disk space
3. 64MB of RAM for Red Hat Enterprise Linux installations: 256 MB of RAM
4. 300 MHZ CPU
5. 800 MB of free disk space
6. Required Items:

Following items are required before starting. Make sure to verify items in the following checklist:

- Computer with Windows 10.
- ISO/Image/Setup of desired version of Cent OS 6 (ask your lab administrator to share network path/common share)
- Free disk space of 40 GB.

Procedure:

Steps for Installation and creating new project are available below as follows:

A. INSTALLATION:

a. Use the latest ISO given by your lab administrator.
b. Run the setup and follow the instructions to install.
c. Select suitable components, define path and click Next/Install.
d. Wait until the setup completes.
e. Install the provided license key from your University.

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

B. CREATING VIRTUAL MACHINE:

a. Make sure you have a local copy ”CentOS-6-Live.iso” image. If you are unable to
b. find in D Drive, Copy from CSBlack\OS\Images\.
c. 2. Start Menu -> VMWare Workstation.
d. 3. File -> New Virtual Machine.
e. 4. Browse -> D Drive (Copy directory).
f. 5. Type your <RegistrationID> as Machine Name.
g. 6. Set up configuration according to Hardware Requirements.
h. 7. Finish & Power On Virtual machine.

Figure 1 -Installation Window - Press next

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

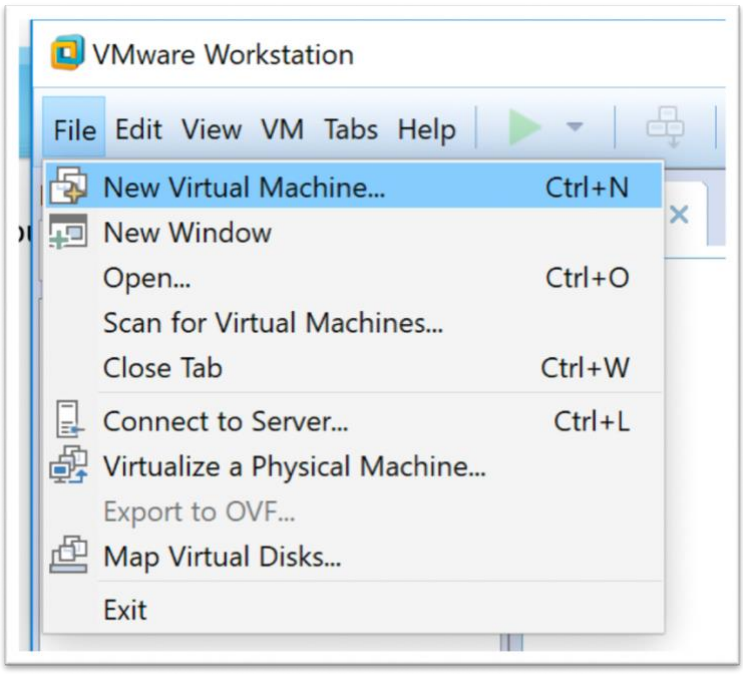


Figure 2 - Create a new Virtual Machine

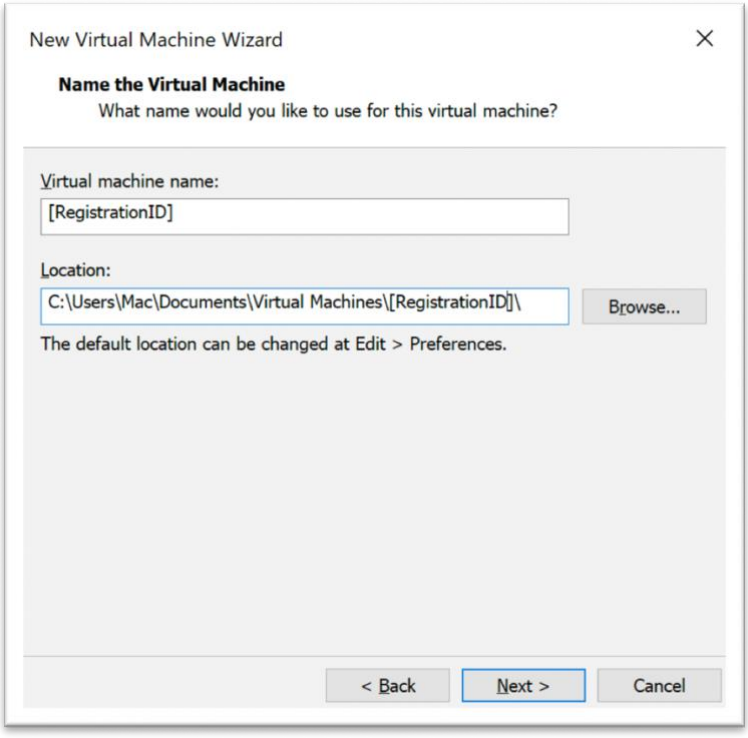


Figure 3 - Write your Registration ID

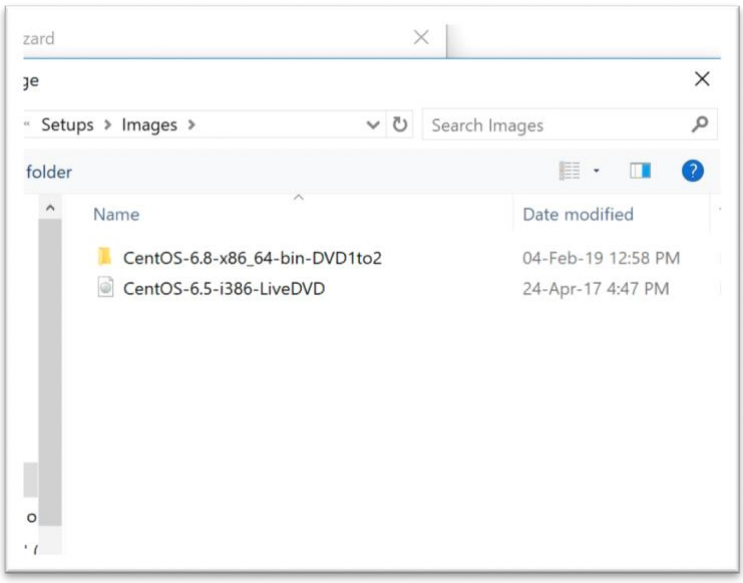


Figure 4 - Folder of Images - Usually present at your desktop

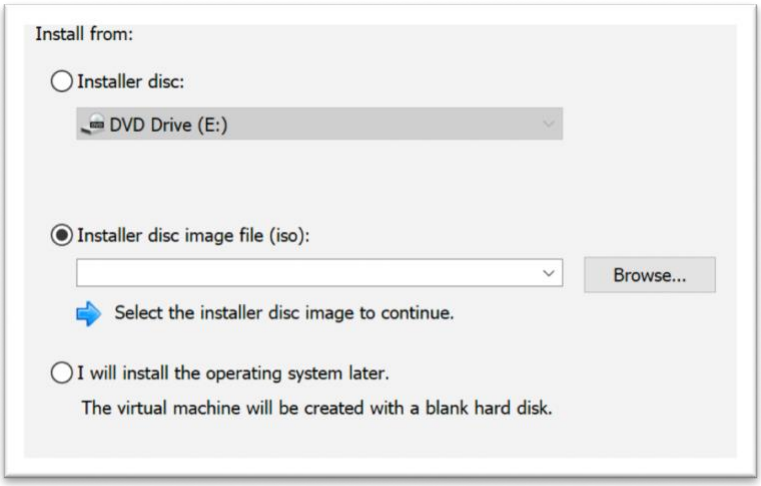


Figure 5 - Select the Images Folder and CentOS Image (ISO) file

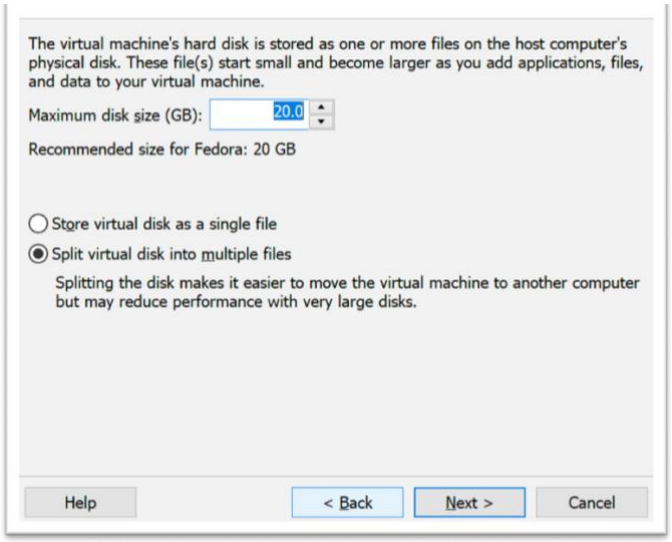


Figure 6 - Select disk type - Click Next

C. Power on Virtual machine

a. Modes:

0. **Graphical mode**, is Window X System, containing all application program that have compatible graphical user interface.
1. **Text mode**, is command line or prompt shell, it can use any process which does not contain any graphical user interface.

b. Run level and `init`:

0. Halt
1. Single-user text mode
2. Not used (user-definable)
3. Full multi-user text mode
4. Not used (user-definable)
5. Full multi-user graphical mode (with an X-based login screen)
6. Reboot

c. User & privileges

- **Standard User**, as Linux-based are multi-user operating system, which means that applications, files and settings are separate for each user and the resources of the physical machine are shared among all users.
- **Super User** (Administrator) – su, Authorized with all privileges, Super user is an administrator for any Linux-based operating system. It can move into any users' personal space, access any applications and add or delete any user.
- **Groups**, users with specific privileges are classified into groups. Permissions are granted over the entire group. Example: LabAdmins(SU), Students(U) and Faculty (U) are groups with specific permissions. These are also known as roles.

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Basics:

1. Runlevel 3 is for CLI, and 5 is for GUI, in order to find out current runlevel of the operating system

   $ runlevel

2. The output for the above command will be "N 3", the letter 'N' indicate that runlevel has not been changed since the last system boot. "3" indicate current runlevel

3. In prompt/terminal/shell/command, it is generally echo $RUNLEVEL and echo $PREVLEVEL.

4. Sudo, allows user to execute commands and run programs with security privileges (usually of superuser). It prompts for your personal password and allows to execute command. The "sudoers" file contain information of users and their respective privileges. This file is maintained by system administrator. Ex: sudo command

Shell:

A system program that allows a user to execute:

- Shell functions (internal commands)
- Other programs (external commands)
- Shell scripts

Linux/UNIX have many of them, includes:

- CSH (C Shell)
- BASH (Bourne Again Shell)
- ZSH (Z Shell)
- KSH (Korn Shell)

Features:

- Shell is a "power-user" interface, so the user interacts with the shell by typing in the commands.
- The shell interprets the commands, that may produce some results, they go back to the user and the control is given back to the user when a command completes (in general).
- In the case of external commands, shell executes actual programs that may call functions of the OS kernel.
- These system commands are often wrapped around a so-called system calls, to ask the kernel to perform operation(s) (usually privileged) on your behalf.

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Command I/O

- Input to shell: Command name and arguments typed by the user
- Input to a command: Keyboard, file, or other commands
- Standard input: keyboard.
- Standard output: screen/display.
- These STDIN and STDOUT are often together referred to as a terminal.
- Both standard input and standard output can be redirected from/to a file or other command.
- File redirection:
    1. ◦ < input
    2. > output
    3. >> [output append]

Command Format:

Format: command name and o or more arguments:

% commandname [arg1] … [argN]

By % mean prompt here and hereafter.

Arguments can be options (switches to the command to indicate a **mode** of **operation**).

Usually prefixed with a hyphen (-) or two (--) in GNU style non-options, or operands, basically the data to work with (actual data, or a file name).

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Commands:

1. **man** - Manual pages
   In Unix, most programs, and many protocols, functions, and file formats, have accompanying manuals. With the man command, you can retrieve the information in the manual and display it as text output on your screen.

To use the man command, at the Unix prompt, enter:

1. **Manual** in Unix-based Operating systems.
2. Provides information regarding commands, tools an
3. **Usage**

% man [topic]
Ex: % man ftp
% man -k mail | more % man man

2. passwd

1. Change your login password.
2. A very good idea after you got a new one.
3. It's usually a awkward program asking your password to have at least 6 chars in the password, at least two alphabetical and one numerical characters. Some other restrictions (e.g. dictionary words or previous password similarity) may apply.
4. Depending on a privilege, one can change user's and group passwords as well as real name, login shell, etc.

$ passwd [username]
$ man passwd

3. date

Displays dates in various formats
$ date
$ date –u (in GMT) $ man date

4. cal

1. Calendar, for month and for entire year
2. Years range: 1 – 9999 (but not 0)

3. Calendar was corrected in 1752 - removed 11 days

$ cal	#Current Month
$ cal 2 2000	# Feb 2000, leap year
$ cal 2 2100	# not a leap year
$ cal 2 2400	# leap year
$ cal 9 1752	# 11 days skipped
$ cal 0	# error
$ cal 2019	# whole year

5. clear

1. Clears the screen

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

2. There's an alias for it: ***Ctrl+L***
3. Example sequence: $ [somecommand]
$ [anothercommand] $ [fewmorecommands] $ [output]
$ clear #Clear the screen
$ [command]

6. sleep

1. "Sleeping" is doing nothing for some time.
2. Usually used for delays in shell scripts (Lab 4).
$ sleep Ns (N is Integer and s is suffix.)
$ sleep 10 (10 seconds pause)
$ sleep 5h 30m 1s (sleep for 5 hours, 30 minutes and 1 second as 'h' denotes hours, 'm' for minutes and 's' for seconds).
$ sleep 1d (for days)

7. time

The **time** command runs the specified program command with the given arguments. When command finishes, time writes a message to standard error giving timing statistics about this program run.

These statistics consist of:
1. The elapsed real time between invocation and termination.
2. The user CPU time.
3. The system CPU time.
4. Example:

$ time [-p] command [arguments...] ('-p' switch indicate printable POSIX format)
$ time cal
$ time date

8. Command grouping & redirection

The use of Semicolon: ";".
Often grouping acts as if it were a single command, so an output of different commands can be redirected to a file:
$ (date; cal; date) (Command Grouping)
$ echo "SP-OS-19" > 1712007.txt (Redirection) $ (date; cal; date) > Cal.txt

9. which

1. Displays a path name of a command.
2. Searches a path environmental variable for the command and displays the absolute path.
3. To find which tcsh and bash are actually in use, type:
$ which tcsh
$ which bash
$ man which # for more details

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

10. `whereis`

1. Display all locations of a command (or some other binary, man page, or a source file).
2. Searchers all directories to find commands that match **`whereis`** argument

```
$ whereis tcsh
$ whereis ftp
$ whereis mv
```

11. `unalias`

1. Removes alias
2. Requires an argument. $ unalias notepad

12. `history and !`

	History	!
1	Display a history of recently used commands	!n # repeat command **n** in the history
2	history (all commands in the history)	!-1 #repeat last command = !!
3	history 10 (last 10)	!-2 # repeat second last command
4	!! (repeat last command)	!ca #repeat last command that begins with 'ca'

13. `whoami`

Displays the current (logged-in) user

```
$ whoami
$ echo $(whoami) (Common in scripting practice)
```

14. `apropos`

Search man pages for a substring like **$ apropos word**

Equivalent:

```
$ man -k word
$ apropos date
$ man -k date
$ apropos passwd
```

15. `exit` / `logout`

1. Exit from your login session.

```
$ exit
```

16. `shutdown`

1. Causes system to shut down or reboot cleanly.
2. May require **superuser** privileges

```
# shutdown -h now          # stop
# shutdown -r now          # reboot
# reboot                   # reboot (Support with CentOS)
```


100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

17. ls

1. List directory contents
2. Has whole bunch of options, # see man ls for details.

```
% ls          # all files except those starting with a "."
% ls-a              # all file including with "." at the start.
%l s-A        # all without "." and ".."
% ls -F       # append "/" to dirs. and "*" to executables
% ls-l        # long format
% ls -al
% ls -lt             # sort by modification time (latest - earliest)
% ls -ltr     # reverse
```

18. cat

1. Display and concatenate files.

$ cat (Will read from STDIN and print to STDOUT every line you enter.)
$ cat file1 [file2] ... (Will concatenate all files in one and print them to STDOUT)

$ cat > filename (Will take whatever you type from STDIN and will put it into the file filename)

2. To exit cat or cat > filename type Ctrl+D to indicate EOF (End of File).

19. head/tail

1. Head shows the top (10) lines of the content.

$ head [filename]
$ head -n # [filename] (where # can be any positive number)
$ head –n 5 Script.sh

2. Tail, display the last (10) lines of the file or the content.

$ tail [filename]
$ tail -n # [filename] (where # can be any positive number)
$ tail –n 5 footer.php

20. more / less

1. Pagers to display contents of large files page by page or scroll line by line up and down.
2. Have a lot of viewing options and search capability.
3. Interactive. To exit: 'q'

$ less filename
/- Word Search ('n' and 'N' to move towards next and previous occurrence)
? - Substring Search ('n' and 'N' to move towards next and previous occurrence)

21. less

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

1. less ("less is more") a bit smarter than the more command

To display contents of a file:
$ less filename
To display line numbers:
$ less -N filename To display a prompt:
$ less –P "Press 'q' to quit" filename Combine the two:
$ less –NP ”New blah blah ... " filename # For more information: $ man less

Use the following screen navigation commands while viewing large log files.
2. CTRL+F – forward one window
3. CTRL+B – backward one window
4. CTRL+D – forward half window
5. CTRL+U – backward half window

In a smaller chunk of data, where you want to locate particular error, you may want to navigate line by line using these keys:
1. j – navigate forward by one line
2. k – navigate backward by one line

The following are other navigation operations that you can use inside the less pager.
G – go to the end of file
g – go to the beginning of file
q or ZZ – exit the less pager

Similar to Vim editor navigation command, you can give 10j to scroll 10 lines down, or 10k to go up by 10 lines.
1. 10j – 10 lines forward.
2. 10k – 10 lines backward.
3. CTRL+G – show the current file name along with line, byte and percentage statistics.
February 13, 2019

22. touch

1. By touching a file you either create it if it did not exists (with 0 length) Or you update its last modification and access times.
2. There are options to override the default behavior.
$ touch file.extension (example ‘todo.txt’)
$ touch Script.sh (generate a shell script with ‘sh’ extension) $ man touch (see manual pages for touch command)

23. cp

1. Copies files / directories.
$ cp [options] <source> <destination>
$ cp file1 file2
$ cp file1 [file2] ... /directory

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

2. Useful option: -i to prevent overwriting existing files and prompt the user to confirm.
3. Both files must be in the same working directory. If they are in various directories, the absolute path must be given.

24. `mv`

1. Moves or renames files/directories.
$ mv <source> <destination> (The <source> gets removed)
$ mv file1 dir/
$ mv file1 file2
$ mv index.html /web/index.html
2. Rename
% mv file1 file2 dir/
% mv dir1 dir2

25. `rm`

1. Removes file(s) and/or directories.
$ rm file1 [file2] ...
$ rm -r dir1 [dir2] ...
$ rm -r file1 dir1 dir2 file4 ...
2. You will remove all files beginning with i and ending with x which are in working directory.
$ rm h*d
3. If you write rm * (you will erase all files from your working directory.)

26. `find`

1. Looks up a file in a directory tree.
$ find . -name name
$ find . 'w*' -or -name 'W*'

27. `mkdir`

1. Creates a directory.
$ mkdir [new directory]
2. Often people make an alias of md for it.

28. `rmdir`

1. Removes a directory.
$ rmdir dirname
2. Equivalent:
$ rm -r dirname

29. `cd`

1. Changes your current directory to a new one.
$ cd /home/asif/Desktop (Absolute path)
$ cd subdir (Assuming subdir is in the current directory.

2. Returns you to your home directory $ cd
3. Moves to the superior directory $ cd ..

30. pwd

1. Displays personal working directory, i.e. your current directory.
$ pwd
$ echo $(pwd) (common in scripting)

31. grep

1. Searches its input for a pattern.
2. The pattern can be a simple substring or a complex regular expression.
3. If a line matches, it's directed to STDOUT; otherwise, it's discarded.
$ echo "OS-19" | grep 19 (Will print the matching line)
$ echo "OS-19" | grep 20 (Will not print the matching line)
$ grep "text you want search" [filename]

32. Pipes

What's a pipe?
◦ is a method of inter-process communication (IPC)
◦ in shells a '|' symbol used
◦ it means that the output of one program (on one side of a pipe) serves as an input for the program on another end.
◦ a set of "piped" commands is often called a pipeline Why it's useful?
◦ Because by combining simple OS utilities one can easily solve more complex tasks
$ man man | less
$ cat Script.sh | head

Tasks

1. Find pages/lines/command containing keyword 'log'
2. List all hidden files
3. Concatenate header.sh, body.sh, and footer.sh into index.sh
4. Use touch and echo to create file with text in it
5. Hide all files in the existing folder
6. Find the following

- find
- less
- "logs"
- file.ext
- /Desktop

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Lab – 3 & 4
Users, Groups and Permission

Objective:

To manage users in Linux systems.

Commands:

- Linux understands Users and Groups.
- A user can belong to several groups (e.g. superuser and Students).
- A file can belong to only one user and one group at a time.
- A particular user, the superuser "root" has extra privileges (uid = "0")
- Only root can change the ownership of a file.
- In UNIX/LINUX, there is a concept of user and an associated group.
- The system determines whether or not a user or group can access a file or program based on the permissions assigned to them.

Understanding Unix File System:

1. The Unix file system looks like an inverted tree structure.
2. You start with the root directory, denoted by /, at the top and work down through sub-directories beneath it.
3. Each node is either a file or a directory of files, where the latter can contain other files and directories.

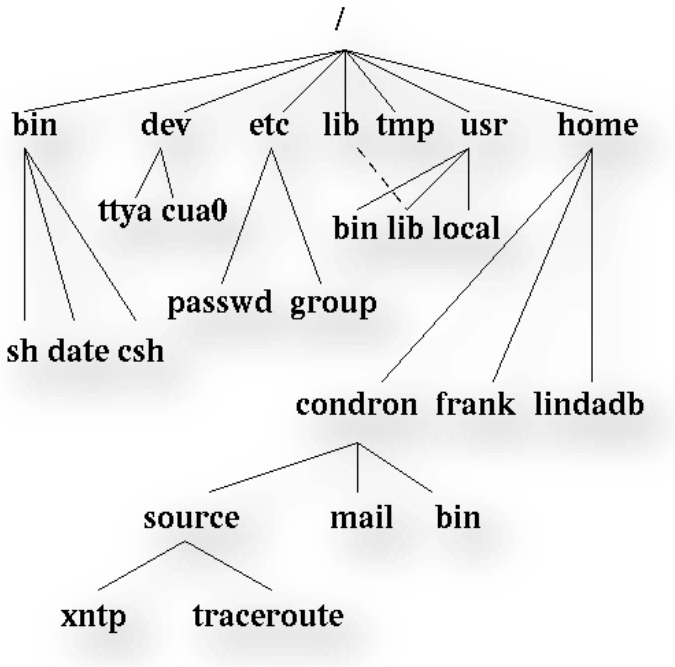


Figure 7 - File System looks like inverted tree

File System:

1. You specify a file or directory by its path name, either the full, or absolute, path name or the one relative to a location.
2. The full path name starts with the root, /, and follows the branches of the file system, each separated by /, until you reach the desired file, e.g.:

   $ cat /home/asif/Desktop/web/index.html
3. A relative path name specifies the path relative to another, usually the current working directory that you are at. Two special directories:

   . the current directory
   .. the parent of the current directory
4. So, if I'm at **/home/asif** and wish to specify the path above in a relative fashion I could use:
   ../Desktop/web/index.html

Standard Directories:

1. **/** The ancestor of all directories on the system; all other directories are subdirectories of this directory, either directly or through other subdirectories.
2. **/bin** Essential tools and other programs (or binaries).
3. **/dev** Files representing the system's various hardware devices. For example, you use the file `/dev/cdrom' to access the CD−ROM drive.
4. **/etc** Miscellaneous system configuration files, startup files, etc.
5. **/home** The home directories for all of the system's users.
6. **/lib** Essential system library files used by tools in `/bin'.
7. **/proc** Files that give information about current system processes.
8. **/root** The superuser's home directory, whose username is root. (In the past, the home directory for the superuser was simply `/'; later, `/root' was adopted for this purpose to reduce clutter in `/'.)
9. **/sbin** Essential system administrator tools, or system binaries.
10. **/tmp** Temporary files.
11. **/usr** Subdirectories with files related to user tools and applications.

1. **useradd**:
   1. Create new user for Students group as:
      # useradd [username]
      # useradd [Username]
      (*NOTE: username and Username are not same in Linux.)*
      # useradd [RegistrationID]

   2. Other options
      # useradd Printers --no-create-home (create user without Home Directory)
      # useradd Printers –shell /sbin/nologin (create user no remote login or shell)

2. userdel:

1. Removes a user

   # userdel [username]

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

2. Remove specified user with home directory

   # userdel –r [username]

3. **groupadd:**
   1. Setup password for the new user as:

      # passwd [RegistrationID]
   2. Create new group named 'Students' as:
      # groupadd Students
   3. Assign users to a particular group using:

      # usermod –g Students [RegistrationID]

      for privileges of superuser/root

      # usermod –g root [RegistrationID]

Permissions:

There are three permissions for any file, directory or application program. The following lists the symbols used to denote each, along with a brief description:

r — Indicates that a given category of user can read a file.

w — Indicates that a given category of user can write to a file.

x — Indicates that a given category of user can execute the file.

Permissions

Each of the three permissions are assigned to three defined categories of users. The categories are:
owner — The owner of the file or application.

group — The group that owns the file or application.

others — All users with access to the system.

One can easily view the permissions for a file by invoking a long format listing using the command ls -l.

For instance, if the user asif creates an executable file named Script.sh, the output of the command ls -l test would look like this:

```
rwxrwxr-x 1 asif Students 0 Feb 10 12:25 Script.sh
```

1. The permissions for this file are listed at the start of the line, starting with **rwx**.
2. This **first set** of symbols define **owner access**.
3. The **next set** of **rwx** symbols define **group access**.

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

4. The **last set** of symbols defining access permitted for **all other users**.
5. This listing indicates that the file is **readable**, **writable**, and **executable** by the user who owns the file (user asif) as well as the group (Students) owning the file.
6. The file is also **world-readable** and **world-executable**, but not **world-writable**.

For Files and Directories:

The ownership of the file or directory can be changed using the command

```
# chown [username] [file/directory name]
```

The group of the file or directory can be changed using the command

```
# chgrp [group] [file/directory name]
```

The permissions of the file can be changed using chmod command

```
# chmod -R ### [filename or directory]
```

-R is optional and used when directory contains sub-directories thereby changing ALL the permissions to ###.

- The #'s can be:

  0 = Nothing
  1 = Execute
  2 = Write
  3 = Execute & Write (2 + 1)
  4 = Read
  5 = Execute & Read (4 + 1)
  6 = Read & Write (4 + 2)
  7 = Execute & Read & Write (4 + 2 + 1)

- Examples are:

  $ chmod 644 index.html (common in web development)
  $ chmod 700 MyPersonalExpenseManager.sh (limited to owner only)
  $ chmod 777 malware.sh (open for all)

Tasks:

1. Create users based on your registration IDs.
2. Make a common folder to share among all students and faculties.
3. Restrict the execute on all shell files in common folder.

Lab – 5, 6 & 7

Introduction to Bash Scripting

Objective:

To study bash scripting in CentOS and UNIX System. To know different constructs and syntax of bash programming alongside control or decisional, looping or iterative and sequential execution structures.

Procedure:

Script:

1. A script is a **program** or **sequence of instructions** that is interpreted or carried out by another program rather than by the computer processor (as a compiled program is). Among the most popular are Perl, REXX (on IBM mainframes), JavaScript, and Tcl/Tk.
2. Scripting languages are also sometimes referred to as *very high-level programming languages*, as they operate at a high level of abstraction, or as **control languages**, particularly for job control languages on mainframes.
3. The spectrum of scripting languages ranges from very small and highly domain-specific languages to general-purpose programming languages used for scripting. Standard examples of scripting languages for specific environments include: **Bash**, for the **Unix.**

Basic Shell Programming:

1. A script is a file that contains shell commands
    1. Data structure: Variables
    2. Control Structure:
        1. Sequence
        2. Decision
        3. Loops/Repetition
2. Shebang line for the bash script
   #! /bin/bash
   #! /bin/sh
3. To execute,
    1. Make executable: **% chmod +x [scriptname.extension]**
    2. Call via: **% ./script**
4. Input
    1. prompting user
    2. command line arguments
5. Decision:
    1. if-then-else
    2. case
6. Repetition

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

    1. do-while, repeat-until
    2. for
    3. select
7. Functions
8. Signal and Interrupts

User Input and Variables:

1. Shell allows to prompt for user input

   read [variablename] [more variables]

   read –p "Enter your First and Last name" firstname lastname
2. words entered by user are assigned to

   firstname and **lastname** last variable gets rest of input line
3. Let for mathematical operations

   let x=12;let y=$x+$x;

4. **#!/bin/bash** (Points to the specific interpreter for script)
5. **read -p "enter your name (first and last): " first last** (Prompts user)
6. **echo "First name: $first"** (Prints the first variable)
7. **echo "Last name: $last"** (Prints the last variable)

Special Shell Variables:

Table 1 - Special Shell Variables

Parameter	Purpose
1) $0	Name of the current shell script
2) $1-$9	Positional parameters 1 through 9
3) $#	The number of positional parameters
4) $*	All positional parameters, "$*" is one string
5) $@	All positional parameters, "$@" is a set of strings
6) $?	Return status of most recently executed command
7) $$	Process id of current process

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Decision Structures:

Bash Control Structure

a) If-then-else
b) Case (switches)
c) Loops
   i) For
   ii) While
   iii) Until
   iv) Select

IF THEN STATEMENT

if [expression]
then
    [statements];
fi
[statements];

are executed only if [expression] is true.

test:

1. Evaluates [expression] and returns true or false.

   $ **test** expression
   $ [result]
2. Example code snippet:

```
if test –w "$1“
then
        echo "file $1 is write-able “
fi
```

If-then-else:

1. executes statements-1 if condition is true
2. executes statements-2 if condition is false

```
if [ condition ]; then
        statements-1
else
        statements-2
fi
```

3. Example:

```
if [ 2 –lt 3 ]; then
        echo ‘True’
else
        echo ‘false’
fi
```

else-if:

1. The word **elif** stands for “else if”
2. It is part of the if statement and cannot be used by itself

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

```
if [ condition ]; then
        statements
elif [ condition ]; then
        statement
else
        statements
fi
```

Table 2 - Relational Operators

Purpose	Numeric	String
1) Greater than	-gt	
2) Greater than or equal	-ge	
3) Less than	-lt	
4) Less than or equal	-le	
5) Equal	-eg	= or ==
6) Not equal	-ne	!=
7) str1 is less than str2		str1 < str2
8) str1 is greater str2		str1 > str2
9) String length is greater than zero		-n str
10) String length is zero		-z str

Compound Condition - !:

1. Operator (!) is used for NOT operation.

```
#!/bin/bash

read -p "Enter years of work: " Years
```

```
if [ ! "$Years" -lt 20 ]; then
        echo "You can retire now.“
else
        echo "You need 20+ years to retire“
fi
```

Compound Condition - &&:

1. Operator (&&) is used for AND operation.

```
#!/bin/bash
Bonus=500;
read -p "Enter Status: " Status
read -p "Enter Shift: " Shift
if [[ "$Status" = "H" && "$Shift" = 3 ]]
then
        echo "shift $Shift gets \$$Bonus bonus“
else
        echo "only hourly workers in“
        echo "shift 3 get a bonus“
fi
```

Compound Condition - ||:

1. Operator (II) is used to perform OR operation in shell scripts.

```
read -p "Enter calls handled:" Chandle
read -p "Enter calls closed: " Cclose
if [[ "$CHandle" -gt 150 || "$CClose" -gt 50 ]]
then
        echo "You are entitled to a bonus”
else
        echo "You get a bonus if the calls”
        echo "handled exceeds 150 or"
        echo "calls closed exceeds 50”
fi
```


100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Operations on file:

Operators	Meaning
1) **-d [filename]**	True if 'file' is a directory
2) **-f [filename]**	True if 'file' is an ord. file
3) **-r [filename]**	True if 'file' is readable
4) **-w [filename]**	True if 'file' is writable
5) **-x [filename]**	True if 'file' is executable
6) **-s [filename]**	True if length of 'file' is nonzero

Example: File Readability

'–r' expression is used to determine the readability of a file. Like:

```
#!/bin/bash
echo "Enter a filename: "
read filename
if [ ! –r "$filename" ]
then
        echo "File is not read-able"
        exit 1
fi
```

Example 2: File Test

Refer to previous slides from '-f','-r' and '-w' writability expression.

```
#! /bin/bash
if [ $# -lt 1 ]; then
        echo "Parameter missing, USAGE: filetest.sh SomeFile.ext";
        exit 1;
fi
if [[ ! -f "$1" || ! -r "$1" || ! -w "$1" ]]
then
        echo "File $1 is not accessible"
        exit 1
fi
```

Example 3: IF Statement

1. The following THREE *if*-conditions produce the same result
    1. # USING DOUBLE SQUARE BRACKETS
    read -p "Do you want to continue?" reply
        if [[$reply = "y"]]; then
            echo "You entered " $reply
        fi
    2. # USING SINGLE SQUARE BRACKETS
    read -p "Do you want to continue?" reply
        if [$reply = "y"]; then
            echo "You entered " $reply
        fi
    3. # USING "TEST" COMMAND
    read -p "Do you want to continue?" reply
        if test $reply = "y"; then
            echo "You entered " $reply
        fi

Example 4: ELSE-IF Statement

```
#!/bin/bash

read -p "Enter Income Amount: " Income
read -p "Enter Expenses Amount: " Expense

#Let keyword defines variable for arithmetic operations
let Net=$Income-$Expense
```


100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

```
if [ "$Net" -eq "0" ]; then
    echo "Income and Expenses are equal - breakeven."
elif [ "$Net" -gt "0" ]; then
    echo "Profit of: " $Net
else
    echo "Loss of: " $Net
fi
```

Case Statement:

1. Use the case statement for a decision that is based on multiple choices. Similar in many programming languages as Switch-case statement. Syntax is as follows: **case word in** **pattern1) command-list1** **;;** **pattern2) command-list2** **;;** **patternN) command-listN** **;;** **esac**	2. Checked against **word** for match 3. May contain the following: 2. ***** 3. **?** 4. **[…]** 5. **[:class:]** 4. multiple patterns can be listed via: **|** (PIPE)

1. Following code is syntax-sensitive, you need to be careful while writing.

```
#!/bin/bash
    echo "Enter Y to see all files including hidden files"
    echo "Enter N to see all non-hidden files"
    echo "Enter q to quit"
    read -p "Enter your choice: " reply
case $reply in
    Y|YES) echo "Displaying all (really…) files"
            ls -a ;; # Executes statement if the pattern has a match
    N|NO)  echo "Display all non-hidden files…"
            ls ;; # Executes statement if the pattern has a match 'N'
    Q)    exit 0 ;;
            *) echo "Invalid choice!"; exit 1 ;; # Exit for no match
esac #END-Case statement
```

2. Following example covers numeric data use in case statements.

```
#!/bin/bash
ChildFare=3
AdultFare=10
SeniorFare=7
read -p "Enter your age: " age
case $age in
        [1-9]|[1][0-2])   # child, if age 12 and younger
                echo "your Fare is" '$'"$ChildFare.00" ;;
        # adult, if age is between 13 and 59 inclusive
        [1][3-9]|[2-5][0-9])
                echo "your Fare is" '$'"$AdultFare.00" ;;
        [6-9][0-9])     # senior, if age is 60+
                echo "your Fare is" '$'"$SeniorFare.00" ;;
esac # END-Case
```

Loops:

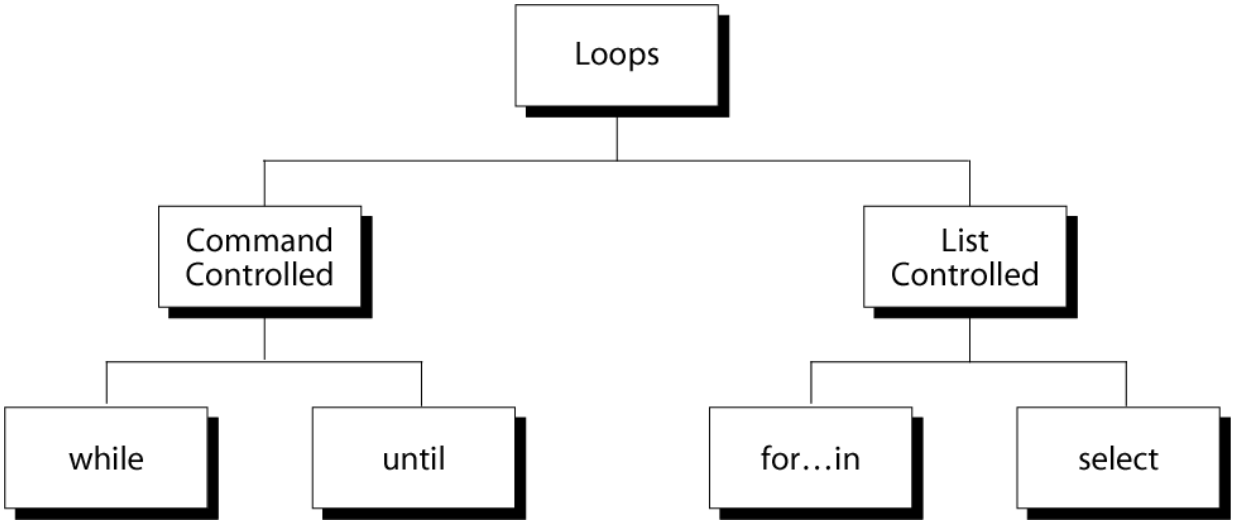


Figure 8 - Loops in Bash

Case Statement:

To execute commands in "command-list" as long as "expression" evaluates to true.

```
while [ expression ]
        do
        command-list
        done
```

(1) Example:

```
COUNTER=0
while [ $COUNTER -lt 10 ]
do
        echo The counter is $COUNTER
```

```
        let COUNTER=$COUNTER+1
    done
```

Example:

```
COUNTER=0
while [ $COUNTER -lt 10 ]
do
        echo 'The counter is' $COUNTER
        let COUNTER=$COUNTER+1
done
```

Until Statement:

To execute commands in "command-list" as long as "expression" evaluates to false

```
until [ expression ]
        do
        command-list
        done
```

Example:

```
COUNTER=20
until [ $COUNTER -lt 10 ]
do
  echo $COUNTER
  let COUNTER-=1
done
```

For Statement:

To execute commands as many times as the number of words in the "argument-list"

```
for variable in argument-list
        do
        commands
        done
```

Example :

```
for i in 7 9 2 3 4 5
do
        echo $i
done
```

Example 2 (Iterate over all arguments):

```
for parameter
do
    echo $parameter
done
```


100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Select Statement:

1. Constructs simple menu from word list
2. Allows user to enter a number instead of a word
3. User enters sequence number corresponding to the word

```
select WORD in LIST

do
        RESPECTIVE-COMMANDS
done
```

4. Example 1:

```
select var in alpha beta gamma
do
        echo $var
done
```

5. Example 2:

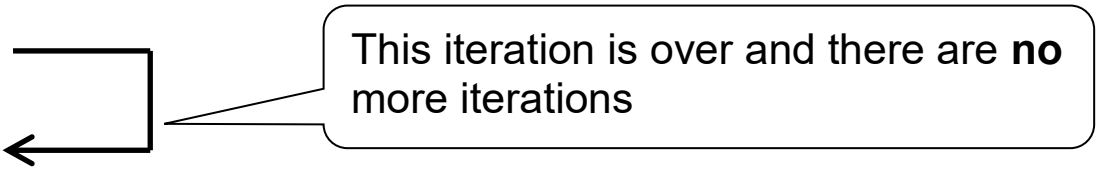
```
PS3="select entry or ^D: "
select var in alpha beta
do
        echo "$REPLY = $var"
done
```

Break and Continue:

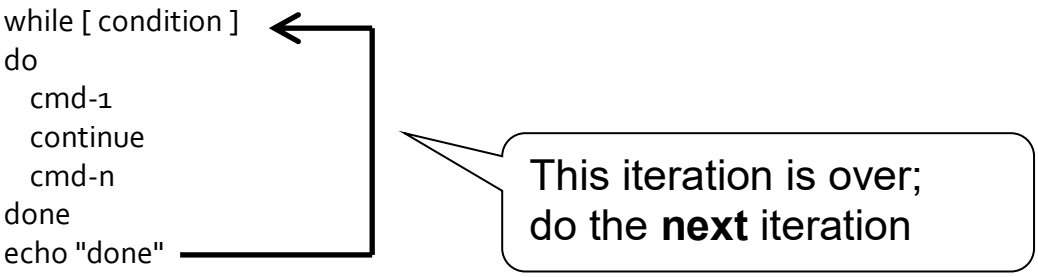
- Interrupt for, while or until loop
- The break statement(s)
  - transfer control to the statement AFTER the done statement
  - terminate execution of the loop
- The continue statement
  - transfer control to the statement TO the done statement
  - skip the test statements for the current iteration
  - continues execution of the loop

break:

```
while [ condition ]
do
  cmd-1
  break
  cmd-n
done
echo "done"
```



Continue:



1. **The following example will help understand select and break statement.**

```
for index in 1 2 3 4 5 6 7 8 9 10
do
        if [ $index –le 3 ]; then
                echo "continue"
                continue
fi
        echo $index
if  [ $index –ge 8 ]; then
                echo "break"
                break
fi
done
```

Shell Functions:

1. A shell function is similar to a shell script
    1. Stores a series of commands for execution later
    2. Shell stores functions in memory
    3. Shell executes a shell function in the same shell that called it
2. Where to define
    1. In .profile
    2. In your script
    3. Or on the command line
3. Remove a function
    1. Use unset (built-in)

User Defined Functions:

4. Must be defined before they can be referenced (due to sequential execution)
5. Usually placed at the beginning of the script

```
function-name () {
```

statements
} # END-function-name

6. Example:
   PrintGreetings () {
   # This is a simple function
   echo ”Hello Student: "
   echo ”Welcome to OS Bash Programming."
   }

   PrintGreetings

Functions Parameters:

1. Need not be declared
2. Arguments provided in script via function call are accessible inside function as $1, $2, $3…
   a) $# (reflects number of parameters)
   b) $0 (still contains name of script, not the name of the function)

Functions with Parameters:

```
checkfile() {
     for file
          do
          if [ -f "$file" ]; then
               echo "$file is a file”
          else
          if [ -d "$file" ]; then
               echo "$file is a directory”
          fi
          fi
     done
     }

checkfile . funtest
```

Variables in Functions:

1. Variables defined with functions are ***global***,
   i.e. their values are known throughout the entire shell program
2. keyword “local” inside a function definition makes referenced variables “local” to that function

```
global=”PathToSomeFileInSpace”
Curiosity () {
        local mars=”PathToSomeFileOnMars”
        echo $global
```

```
        echo $mars
        global="PathToSomeFileOnMars"}

    echo $global
    Curiosity
    echo $global
    echo $mars
```

Handling Signals:

1. Unix allows you to send a signal to any process
2. -1 = hangup
3. -2 = interrupt with
4. No argument = terminate
5. 9 = kill (-9 cannot be blocked)
6. List your processes with
   ps -u userid
7. Examples:
   a) **kill -HUP 1234**
   b) **kill -2 1235**
   c) **kill 1235**
   d) **kill -9 1236** #(1236 is a process id associated by some process)

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Figure 9 - Signals in UNIX

Trap:

1. Default action for most signals is to end process
   a) term: **signal handler**
2. Bash allows to install custom signal handler

   trap 'handler commands' [signals]

3. Example:

   trap 'echo do not hangup' 1 2

4. Reset trap

   trap [signal]
   trap 1

5. Trap for signal '1' is won't terminate the script but '-2' will terminate the and scripted as:

```
#! /bin/bash
# kill -1 won't kill this process
# kill -2 will kill this process

trap 'echo do not hang up' 1

while true
do
        echo "try to hang up"
        sleep 1
done
```

Debugging:

1. Debugging is troubleshooting errors that may occur during the execution of a program/script.
2. The following two commands can help you debug a bash shell script:
   echo
3. use explicit output statements to trace execution
   set
4. The "set" command is a shell built-in command. It has options to allow flow of execution as:
   –v option prints each line as it is read
   –x option displays the command and its arguments
   –n checks for syntax errors
5. Options can turned on or off
   To turn on the option: **set -xv**
   To turn off the options: **set +xv**
6. Options can also be set via she-bang line
   #! /bin/bash -xv

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Subject: Operating Systems (CSCL2205)
Contact Hours: 3

Tasks:

a) Create a short program of a kiosk for restaurant.
b) Create a script to shutdown computer based on user login.

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Lab – 9
User Groups and Shared Directories

Objective:

To study different access permissions with the example of shared directory.

Procedure:

Recall your understanding of permissions in UNIX systems:

1. There are three permissions for any file, directory or application program.
2. The following lists the symbols used to denote each, along with a brief description:

   r — Indicates that a given category of user can read a file.

   w — Indicates that a given category of user can write to a file.

   x — Indicates that a given category of user can execute the file.
3. Each of the three permissions are assigned to three defined categories of users.
4. The categories are:

   owner — The owner of the file or application.

   group — The group that owns the file or application.

   others — All users with access to the system.
5. One can easily view the permissions for a file by invoking a long format listing using the command **ls -l**.
6. For instance, if the user **asif** creates an executable file named **Script.sh**, the output of the command**ls -l** test would look like this:

   rwxrwxr-x 1 asif Students 0 Feb 10 12:25 Script.sh

7. The permissions for this file are listed at the start of the line, starting with rwx.
8. This first set of symbols define **owner access**.
9. The next set of **rwx** symbols define **group access.**
10. The last set of symbols defining access permitted for all **other users**.
11. This listing indicates that the file is readable, writable, and executable by the user who owns the file (user **asif**) as well as the group (**Students**) owning the file.
12. The file is also world-readable and world-executable, but not world-writable.
13. The ownership of the file or directory can be changed using the command

    # chown [username] [file/directory name]

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

14. The group of the file or directory can be changed using the command

    # chgrp [group] [file/directory name]
15. The permissions of the file can be changed using chmod command

    # chmod -R ### [filename or directory]

    -R is optional and used when directory contains sub-directories thereby changing ALL the permissions to ###.
16. The #'s can be:

    0 = Nothing
    1 = Execute
    2 = Write
    3 = Execute & Write (2 + 1)
    4 = Read
    5 = Execute & Read (4 + 1)
    6 = Read & Write (4 + 2)
    7 = Execute & Read & Write (4 + 2 + 1)
17. Examples are:

    $ chmod 644 index.html (common in web development)
    $ chmod 700 MyPersonalExpenseManager.sh (limited to owner only)
    $ chmod 777 malware.sh (open for all)

Shared Directory:

Based on your understanding of permissions of Users and Groups, create the directory based as:

1. Create a shared directory in Centos as:
    1. Create a directory in /var/www/ folder
       sudo mkdir /var/www/project
    2. Add new group called 'project'
       sudo groupadd project
    3. Assign group 'Project' to User 'centoslive'
       sudo usermod -a -G project centoslive

       The flags and arguments used in the above command are:
       -a – which *adds* the user to the group.
       -G – *specifies the group name*.
       project – *group name*.
       centoslive – *existing username.*

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

2. Add Few new users as ‘Developer1’ in Group called ‘Project’

   sudo useradd –G project Developer1
3. Assign password to Developer1

   sudo passwd Developer1
4. Assign permissions to group and directory

   sudo chgrp -R project /var/www/Project/
   sudo chmod -R 2775 /var/www/Project/

Tasks:

1. Complete the following tasks:
   - Create new user called ‘guest’ with password ‘guest’
   - Print the current directory
   - Create folder called ‘project’ and provide read-only permissions to others
   - Create file called ‘Main.html’ and provide permission as 755 or 644
   - Create Groups as Students, LabAdmins, Faculty
   - You may assign permission as Read-only to Students, Faculty with Read-only and Write-only and LabAdmins as All
2. Create **commonshare** for Students and LabAdmins

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Lab – 10 & 11
Processes, Management and Scheduling

Objective:

To study various types of processes in Unix system. To manage and schedule processes in CentOS 6 based on desired requirements and expected outcomes.

Procedure:

Processes:

1. A program/command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.
2. Whenever a **command** is *executed* in UNIX/LINUX, it creates/starts a new **process**.
3. Through a 5 digit ID number UNIX/LINUX keeps account of the processes, this number is called process id or **PID**. Each process in the system has a unique *pid*.
4. At any point of time, none of the processes with same pid exist in the system (because it is the pid that Unix uses to track each process).

Foreground and Background Processes:

1. **Foreground Process:** Every process when started runs in foreground by default, receives input from the keyboard and sends output to the screen. Example as:
   $ pwd
2. When a command/process is running in the foreground, no other processes can be started because the prompt would not be available until the program finishes processing and comes out.
3. **Background Process:** It runs in the background without keyboard input and waits till keyboard input is required. Thus, other processes can be executed in parallel with the process running in background since they do not have to wait for the previous process to be completed.
   Adding & along with the command starts it as a background process
   Example:
   $ pwd &

Processes and Command Line (ps) :

1. Command ps (Process status) can be used to see/list all the running processes. For more information -f (full) can be used along with ps

   $ ps
   $ ps –f
2. For a single process information, ps along with process id is used

   $ ps [PID]
   $ ps 20

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

-a: Shows information about all users
-x: Shows information about processes without terminals
-u: Shows additional information like -f option
-e: Displays extended information

Options of ps:

1. Display every active process on a Linux system in generic (Unix/Linux) format.
   $ ps –A
   $ ps –e
2. To perform a full-format listing, add the -f or -F flag.
   $ ps –ef
   $ ps –eF
3. You can select all processes owned by you (user of the ps command, centoslive in this case), type
   $ ps –x
   $ ps -fU centoslive
   $ ps –fu 501
4. The command below enables you to view every process running with root user privileges
   $ ps -U root -u root
5. If you want to list all processes owned by a certain *group*

   $ ps –fG [groupname]
   $ ps –fG Students
   $ ps –fg Students
6. You can list processes by PID as follows:

   $ ps –fp 1121 (Process ID)
   $ ps –f --ppid 1131
   $ ps -fp 2226,1154,1146
7. To select processes by tty, use the -t flag as follows.

   $ ps -ft tty1

Fields/attributes of a process or ps:

1. UID: User ID that this process belongs to (the person/user/account running it)
2. PID: Process ID
3. PPID: Parent process ID (the ID of the process that started it)
4. C: CPU utilization of process
5. STIME: Process start time
6. TTY: Terminal type associated with the process
7. TIME: CPU time taken by the process
8. CMD: The command that started this process

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Termination of a Process:

1. When running in foreground, hitting CTRL + C (interrupt character) will exit the command. For processes running in background kill command can be used if it's *pid* is known.

   $ ps –f
   $ kill 21

Scheduling:

1. A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms.
2. These algorithms are either **non-preemptive or preemptive**.
    1. Non-preemptive algorithms are designed so that once a process enters the running state, it **cannot** be preempted until it completes its allotted time,
    2. Preemptive scheduling is based on **priority** where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

Shortest Job First:

1. Associate with each process the length of its next CPU burst. The CPU is assigned to the process with the smallest CPU burst (FCFS can be used to break ties).
2. Non-preemptive SJF is *optimal* if all the processes are ready simultaneously– gives minimum average waiting time for a given set of processes.
3. SRTF is *optimal* if the processes may arrive at different times.

Example:

1. Consider the following table of 4 processes with their respective burst time. Assume that all processes arrived at the same time.

Table 4 - Processes and Burst Times

Processes	Burst Time
P1	3
P2	2
P3	9
P4	1

Table 3 - Processes and Sorted Burst Times

Processes	Schedule/Burst
P4	1
P2	2
P1	3
P3	9

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Bash Code:

1. Initialize arrays for Processes and Burst Time as:

```
#!/bin/bash

read –a Processes

read –a BurstTime
```

2. Initialize count

```
let length=${#Processes[@]}
```

3. Setup loops

```
for ((i=0;i<=$length;i++))
do

        for ((j=((i+1));j<=$length;j++))
        do

        done

done
```

4. Swapping

```
if [[ ${BurstTime[i]} – gt ${BurstTime[j]} ]]
then
        temp=${BurstTime[i]}
        BurstTime[i]=${array[j]}
        BurstTime[j]=$temp
fi
```

Class Task:

1. Practice the provided code Burst Time
2. Calculate batch job process sequence for provided data
3. Print Job sequence with Job name and Burst Time

Table 6 - Sample Processes and Burst Times

Processes	Burst Time
firefox	2
nano	8
gedit	4
mail	1
ftp	3
http	7

Table 5 - Expected Sorting using SJF

Processes	Burst Time
?	?
?	?
?	?
?	?
?	?
?	?

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

First Come First Serve:

1. Jobs are executed on first come, first serve basis.
2. Easy to understand and implement.
3. Its implementation is based on FIFO queue.
4. Poor in performance as average wait time is high.

Consider the given scenario in Table 7

Table 7 - Processes with Arrival, Execution and Service Times.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

FCFS Parameters:

1. Waiting Time: The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.
2. Turnaround Time: It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Example:

1. The Process queue is given in Figure (2)
2. **Wait time** of each process is as follows
   AWT: (0+4+6+13) / 4 ≈ 6

Table 8 - Sample Processes

Table 9 - Waiting Time Calculation

Process	Service Time	Arrival Time	WT: STime - ATime
P0	0	0	0 - 0 = 0
P1	5	1	5 - 1 = 4
P2	8	2	8 - 2 = 6
P3	16	3	16 - 3 = 13

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Subject: Operating Systems (CSCL2205)
Contact Hours: 3

Task(s):

1. Upon understanding loops, array and user input, construct a program to calculate AWT of processes given in following table by scheduling through FCFS:

Table 10 - Process Details

Process	Arrival Time	Service Time	Wait Time
firefox	0	891	?
nano	2	931	?
ftp	4	989	?

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Lab – 12
Network Management

Objective:

To manage and administer network in CentOS 6 and Unix based system. This session covers manual or command line instructions as well as graphical.

Procedure:

Computers are connected by many different technologies. A network is a two or more computer that can interconnect in a peer-to-peer or client to server fashion most often over a shared and virtual connection.
In other words, networks provide the connection between computer resources in order to accommodate the flow of information. The following are the potential needs for computer networks.

1. **Information exchange** to exchange data and information between different individual users, it is necessary to interconnect the individual users' computers.
2. **Resource sharing** the cost of computer has reduced. However, the cost of a laser printer, bulk storage, and large enterprise software remains high. When computers are interconnected, there is a possibility that, users connected to the network may share the above-mentioned resources.

Manual Configuration:

This configuration is done by modifying the network configuration directly, using a text editor. This method is typical for servers and is a more advanced way of doing it.

1. Open the following file into a text editor, like VI or Nano.

```
# nano /etc/sysconfig/network
```

2. Change the hostname value to the fully qualified name of your computer.

HOSTNAME=R**[1712111].labs.szabist.edu.pk**

3. Open the configuration file for the first network interface, eth0, into a text editor.

```
# nano /etc/sysconfig/network-scripts/ifcfg-eth0
```

4. Modify the file to look similar to the example below.

```
DEVICE=eth0
TYPE=Ethernet
ONBOOT=yes
BOOTPROTO=dhcp
IPV4_FAILURE_FATAL=yes
NAME="SZABIST"
```

5. Restart the Network service

```
# service network restart
```


100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

6. If you using a Virtual Machine Workstation, make sure that your VM is in Bridge mode.

Static Network Address:

1. Open the configuration file for the first network interface, **eth0**, into a text editor.

```
$ nano /etc/sysconfig/network-scripts/ifcfg-eth0
```

2. Modify the configuration file to look similar to the one below. Remember to change the highlighted values to match your environment's.

```
DEVICE=eth0
TYPE=Ethernet
ONBOOT=yes
BOOTPROTO=none
IPADDR=10.0.[X].[UserSuffix]
PREFIX=24
GATEWAY=10.0.0.9
IPV4_FAILURE_FATAL=yes
NAME="SZABIST-Eth-[10.0.[X].[X]]"
```

Setup DNS:

1. DNS configurations are set in /etc/resolv.conf. Within this file, we can specify the search domains and the name servers. The search domains are used as default suffixes when no domain is added to a hostname.

```
$ sudo nano /etc/resolv.conf
```

2. Add the following lines, modifying it them to match your environment.

```
domain 8.8.8.8
search 8.8.4.4
```

System's Network Configuration Tool:

1. The base CentOS 6 installation includes the tool by default. To install it on a minimal installation, follow these instructions. Other you install it from the yum repository (yum install system-config-network)
2. Launch the System-Config-Network tool from the terminal.

```
$ system-config-network-tui
```

3. Ensure **Device Configuration** is highlighted, and then press **Enter** and Select the network interface you want to configure, and then press **Enter (**Figure 10**)**.

4. For Dynamic IP address allocation, ensure the **Use DHCP** box is checked. Otherwise, for static IP address allocation, enter the static IP address, netmask, default gateway, and DNS server information (Figure 11).

5. Tab to highlight **Ok**, and then press **Enter**.
6. Press tab until **Save** is highlighted, and then press **Enter** to save your changes.
7. Press tab until **Save & Exit** is highlighted, and then press Enter to save your changes.
8. To apply your changes, restart your network services.

```
$ service network restart
```

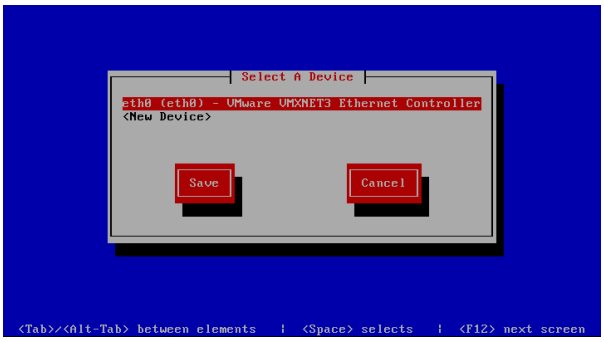


Figure 10 - Device Selection using TUI

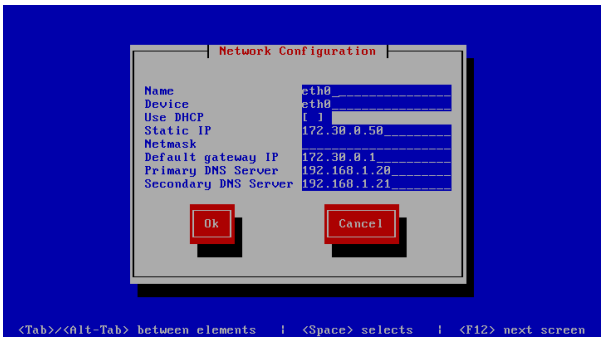


Figure 11 – Network Configuration using TUI

Desktop Configuration:

1. Gnome is the default environment for CentOS 6 Desktop installations. There are plenty of alternative environments that can be installed, but being the default, this one is more widely used. Click to expand the **System** menu, expand **Preferences**, and then click **Network Connections (**Figure 12**)**.
2. From the list of connections in Network Connections application, select the interface you want to configure.
3. Click **Edit** and change **Name** (Figure 13).
4. Click the **IPv4 Settings** tab (Figure 15).

Figure 12 - Network Configuration in CentOS 6

Figure 13 - Network Interface

5. Enter **network settings (**Figure 14**).**
6. Click **apply** and **restart** network (service).

Figure 15 – Interface Selection

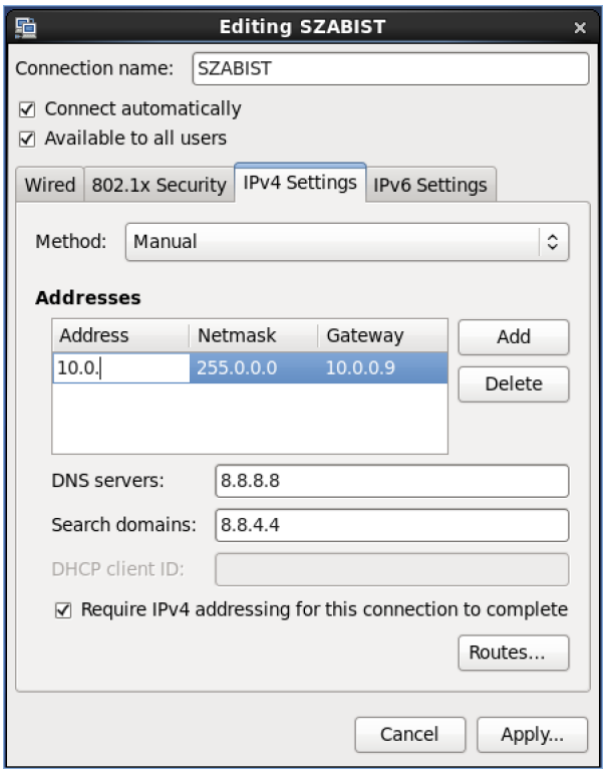


Figure 14 – Network Settings

Remote Connection:

1. Access your Local course folder and Copy "**putty.exe"** available in *Softwares (*Figure 17*).*
2. Enter IP Address of Centos VM and Connect with your username (Figure 16).
3. Make sure to start service (**service sshd start**) or verify with **service sshd status** (Figure 18).

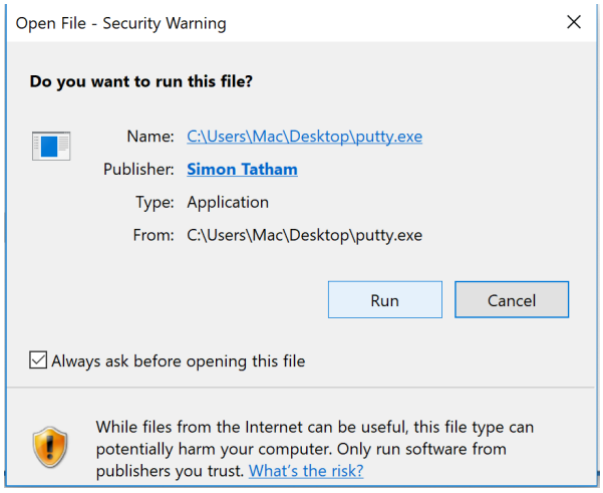


Figure 17 – PuTTy for Windows

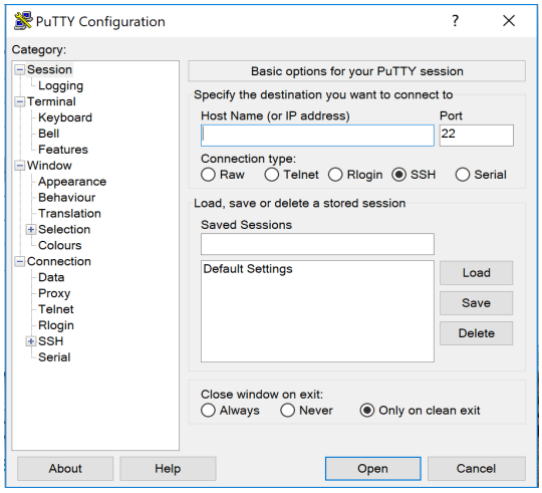


Figure 16 – PuTTy Configuration

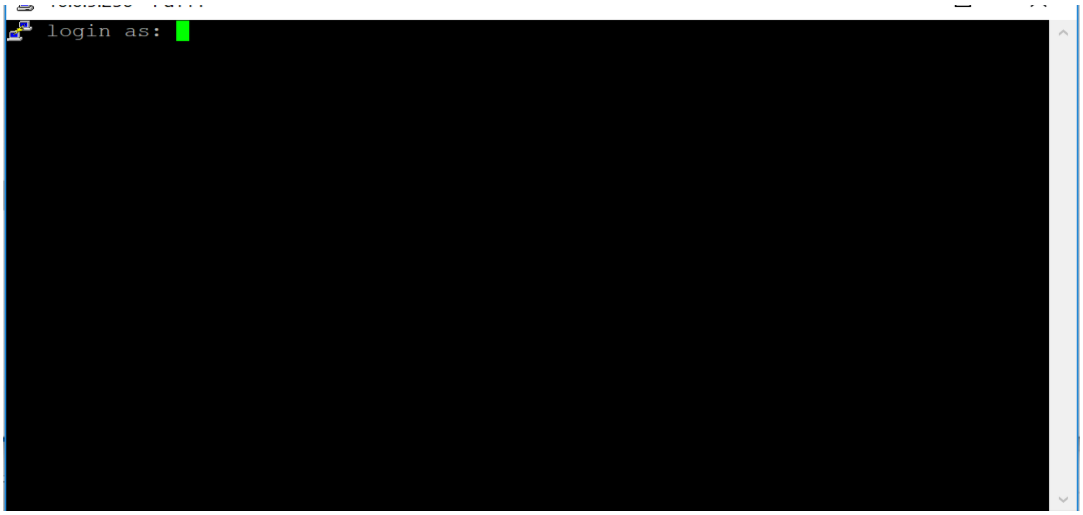


Figure 18 - SSH Login

Tasks:

1. Perform the same task for different Lab settings.
2. Use the domain of other labs.
3. Access the PC designated to FYP students using Ssh.

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Lab – 13 & 14

Package Management and Service Installation

Objective:

To demonstrate packages in Unix System and their management on CentOS 6 using Yum utility.

Procedure:

- A software package is an assemblage of files and information about those files.
- Linux distributions are usually installed as separate software packages, each of which is a particular application, such as a Web browser or a development environment. Each package includes an archive of files and information about the software, such as its name, the specific version and a description. A package management system (PMS), such as rpm or YUM, automates the installation process.

YUM Manager:

Yellow dog Updater, Modified (YUM) is the default package manager used in CentOS (all versions). It is used to install and update packages from CentOS (and 3rd party) Repositories.

1. **Repository**
   a. A **software repository**, colloquially known as a "repo" for short, is a storage location from which software packages may be retrieved and installed on a computer. It can be online/offline/customized.
2. **EXAMPLE:**

```
# yum install PackageName
# yum install firefox
```

Creating YUM Repository:

1. Navigate to /etc/yum.repos.d/
2. Create file as example.repo
3. Modify as

```
[examplerepo]
name=Example Repository
baseurl=http://mirror.cisp.com/CentOS/6/os/i386/
enabled=1
gpgcheck=1
gpgkey=http://mirror.cisp.com/CentOS/6/os/i386/RPM-GPG-KEY-CentOS-6
```

Fields:

- name=Example Repository – ***NAME OF REPOSITORY***
- baseurl=http://mirror.cisp.com/CentOS/6/os/i386/ – ***REPOSITORY URL***
- enabled=1 – ***REPOSITORY STATUS***
- gpgcheck=1 – ***SECURITY CHECK***
- gpgkey=http://mirror.cisp.com/CentOS/6/os/i386/RPM-GPG-KEY-CentOS-6 – ***PRIVACY KEYS***

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

Installation and Configuration of vsftpd:

Very secure File Transfer Protocol Daemon is small utility to initiate a FTP-based server in CentOS and other UNIX systems. To install we do the following:

1. `yum install vsftpd`
2. `yum install ftp`
3. `service vsftpd restart`

4. Configuration file @ /etc/vsftpd/
    1. vsftp.conf
    2. Userslist
    3. FTP Public Directory @ /var/ftp/pub/

5. Create your own directory.

Settings:

Recheck network settings or restart network service

1. Network - Host and client should be on same network.
2. Ping – Ping nearby addresses to verify.
3. Gateway – 10.9.
4. DNS – 8.8.8.8 (for repos)
5. Bridge – Connection Type (Virtual Machine properties) as in Figure 19.
6. Firewall – Allow ports or stop service (iptables)

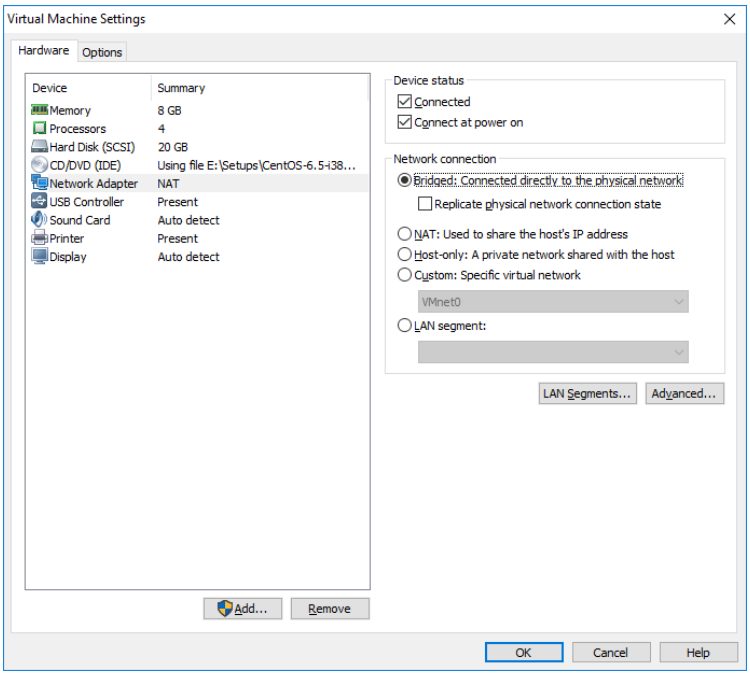


Figure 19 - Network Configuration

Installation and Configuration of httpd:

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computer's HTTP clients.

Dedicated computers and appliances may be referred to as Web servers as well.

Apache HTTP:

The Apache HTTP Server, also called Apache, is a free and open-source cross-platform web server, released under the terms of Apache License 2.0.

Installation

- **`su`**
- **`yum install httpd`**

Configuring HTTPd:

1. **`rm -f /etc/httpd/conf.d/welcome.conf`** – REMOVE DEFAULT WELCOME
2. **`rm -f /var/www/error/noindex.html`** – REMOVE DEFAULT NOT FOUND
3. Open Editor for **`/etc/httpd/conf/httpd.conf`**

```
1. ServerTokens Prod
2. KeepAlive On
3. ServerAdmin root@os.szabist.edu.pk
4. AllowOverride All
5. ServerName os.szabist.edu.pk:80
6. DirectoryIndex index.html index.htm
7. ServerSignature Off
```

4. `/etc/rc.d/init.d/httpd` **`start`**
5. `chkconfig httpd on`
6. Firewall
7. Navigate to /var/www/html/
8. Create file index.html

Sample Page:

- **<html>**
- **<head>**
- **<title> Operating Systems – CSCL2205 </title>**
- **</head>**

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433

- **<body>**
- **<h1> Welcome to my web server</h1>**
- **</body>**
- **<html>**

Apache Directives:

1. **ServerTokens**
   This directive controls whether Server response header field which is sent back to clients includes a description of the generic OS-type of the server as well as information about compiled-in modules.
2. **KeepAlive**
   The Keep-Alive extension to HTTP/1.0 and the persistent connection feature of HTTP/1.1 provide long-lived HTTP sessions which allow multiple requests to be sent over the same TCP connection. In some cases this has been shown to result in an almost 50% speedup in latency times for HTML documents with many images. To enable Keep-Alive connections.
3. **DirectoryIndex**
   The DirectoryIndex directive sets the list of resources to look for, when the client requests an index of the directory by specifying a / at the end of the directory name.
4. **ServerSignature**
   The ServerSignature directive allows the configuration of a trailing footer line under server-generated documents (error messages, mod_proxy ftp directory listings, mod_info output, ...). The reason why you would want to enable such a footer line is that in a chain of proxies, the user often has no possibility to tell which of the chained servers actually produced a returned error message.

Task(s):

1. Create web-based content hosting server (Figure 20).
2. Use local files to share over network via web server.
3. Test media with stream and download.

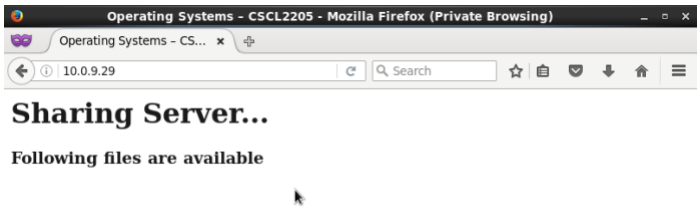


Figure 20 - Sample HTTP Server

References

1. https://kb.iu.edu/d/afjm
2. https://www.ostechnix.com/check-runlevel-linux/
3. Courtesy of Serguei A. Mokhov, mokhov@cs.concordia.ca
4. https://www.computerhope.com/unix/usleep.htm
5. https://www.cyberciti.biz/
6. https://en.wikipedia.org/wiki/Scripting_language
7. https://whatis.techtarget.com/definition/script
8. https://www.tecmint.com/ps-command-examples-for-linux-process-monitoring/
9. https://www.tecmint.com/linux-process-management/
10. https://www.cyberciti.biz/faq/show-all-running-processes-in-linux/
11. https://www.guru99.com/managing-processes-in-linux.html
12. https://www.geeksforgeeks.org/processes-in-linuxunix/
13. https://www.go4expert.com/articles/types-of-scheduling-t22307/
14. https://www.serverlab.ca/tutorials/linux/administration-linux/configure-centos-6-network-settings/

100-C, Block 5 Clifton, Karachi, Pakistan. Phone: 111-922-478 | (021) 35823433