SOLVED PAPERS

OF

UNIX AND SHELL

PROGRAMMING

(JUNE-2013 DEC-2013 JUNE-2014

DEC-2014 & JUNE-2015)

USN | | | | | | | | | | |

10CS44

Fourth Semester B.E. Degree Examination, June/July 2013
UNIX and Shell Programming

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting at least TWO questions from each part.***

PART – A

1 a. With a neat diagram, explain the architecture of unix operating system. **(08 Marks)**

b. With the help of a neat diagram, explain the parent-child relationship. Explain unix file system. **(06 Marks)**

c. Explain briefly absolute pathname and relative pathname with examples. **(06 Marks)**

2 a. Give the significance of the seven fields of the "$ls - l$" command. **(07 Marks)**

b. What is file permission? Explain how to use "Chmod" command to set the permissions in a relative manner with an example. **(07 Marks)**

c. Explain the three different modes in which "V_i" editor works. **(06 Marks)**

3 a. Explain the standard input, standard output and standard error with respect to UNIX operating system. **(07 Marks)**

b. Explain the mechanism of process creation. **(07 Marks)**

c. What are environment variables? Explain any four. **(06 Marks)**

4 a. Differentiate between hard link and soft link with examples. **(06 Marks)**

b. Explain "sort" command briefly. Also discuss its important options with examples (any five). **(06 Marks)**

c. Explain the following commands with example:
i) head ii) tr iii) uniq iv) find **(08 Marks)**

PART – B

5 a. Explain 'grep' command with its options. **(08 Marks)**

b. Explain line addressing and context addressing in "sed" with examples. **(06 Marks)**

c. What are extended regular expression (ERE)? Explain any four ERE set used by "grep" and "egrep". **(06 Marks)**

6 a. Explain the use of "test" and [ ] to evaluate an expression in shell. **(06 Marks)**

b. Explain the shell features of "while" and "for" with syntax. **(06 Marks)**

c. Explain the "expr" command applicable to computation and string functions. **(08 Marks)**

7 a. What is AWK? Explain any three built-in functions in AWK. **(07 Marks)**

b. Write short notes on operators and expressions in AWK. **(06 Marks)**

c. Explain built-in variables in AWK. **(07 Marks)**

8 a. List the string handling functions in PERL. Write a program to find number of characters, words as well as to print reverse of a given string. **(08 Marks)**

b. Explain "chop( )" and "split( )" functions with examples. **(06 Marks)**

c. Explain file handling in PERL. **(06 Marks)**

* * * * *

UNIX AND SHELL PROGRAMMING SOLVED PAPER JUNE -2013

1(a) With a neat diagram, explain the architecture of UNIX operating system (8 Marks)

Ans:

Unix Architecture (Components)

• The UNIX operating system (OS) consists of a kernel layer, a shell layer , an application layer & files.

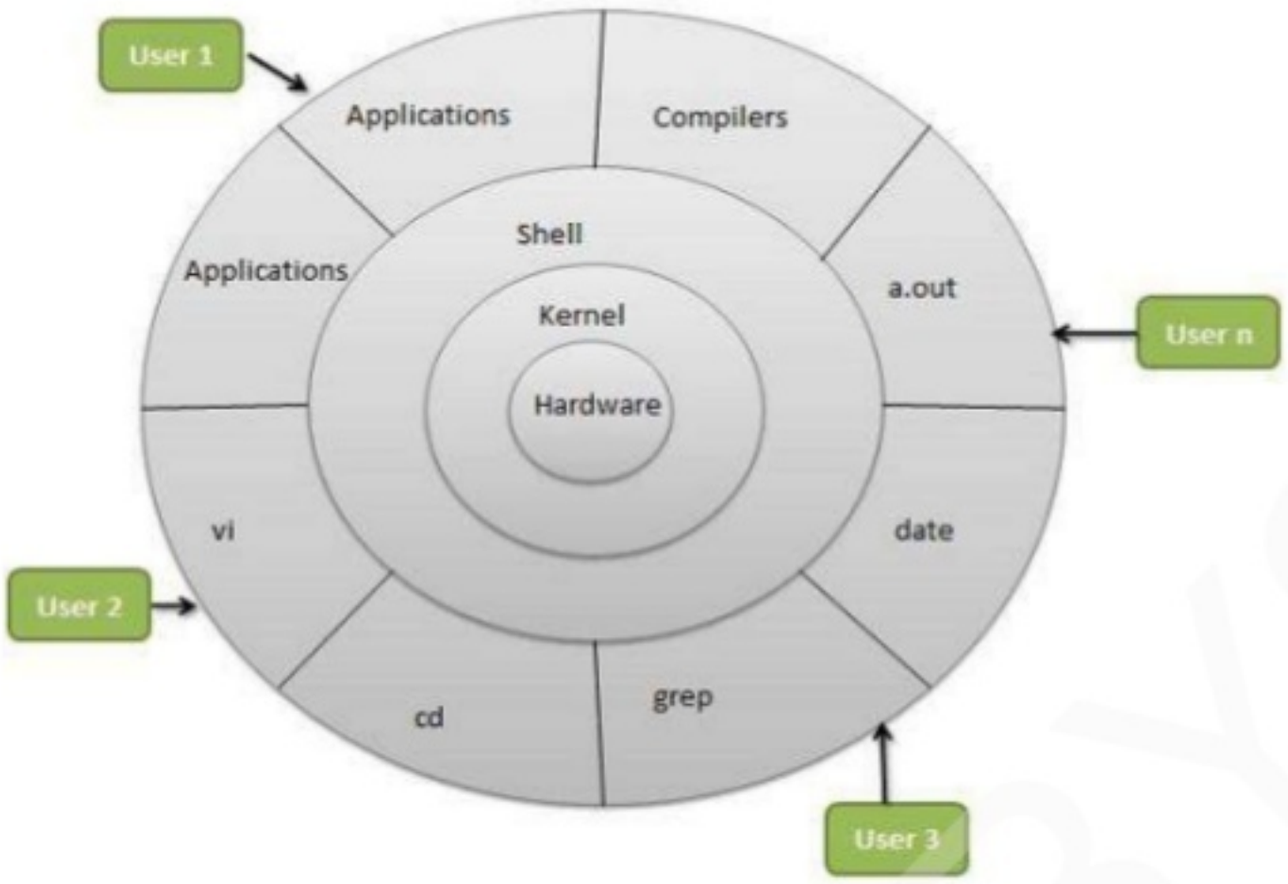


Figure 1.2 : Unix Architecture

Kernel

- The kernel is the heart of the operating system.
- It interacts with the machine's hardware.
- It is a collection of routines written in C.
- It is loaded into memory when the system is booted.
- Main responsibilities:
  1) Memory management
  2) Process management (using commands: kill, ps, nohup)
  3) File management (using commands: rm, cat, ls, rmdir, mkdir)
- To access the hardware, user programs use the services of the kernel via system calls.

System Call

- A system call is a request for the operating system to do something on behalf of the user's program.
- The system calls are functions used in the kernel itself.
- To the programmer, the system call appears as a normal C function call.
- UNIX system calls are used to manage the file system and control processes.
- Example: read(), open(), close(), fork(), exec(), exit()

- There can be only one kernel running on the system.

Shell

- The shell interacts with the user.
- The shell is a command line interpreter (CLI).
- Main responsibilities:
  1) interprets the commands the user types in and
  2) dispatches the command to the kernel for execution
- There can be several shells in action, one for each user who's logged in.
- There are multiple shells that are used by the UNIX OS.
- For example: Bourne shell (sh), the C shell (csh), the Korn shell (ksh) and Bourne Again shell (bash).

Application

• This layer includes the commands, word processors, graphic programs and database management programs.

File

- A file is an array of bytes that stores information.
- All the data of Unix is organized into files.
- All files are then organized into directories.
- These directories are further organized into a tree-like structure called the filesystem.

UNIX AND SHELL PROGRAMMING SOLVED PAPER JUNE -2013

1(b) With the help of a neat diagram, explain the parent-child relationship. Explain UNIX file system. (6 Marks)

Ans:

Parent Child Relationship

- All data in Unix is organized into files.
- All files are organized into directories.
- These directories are organized into a tree-like structure called the filesystem.

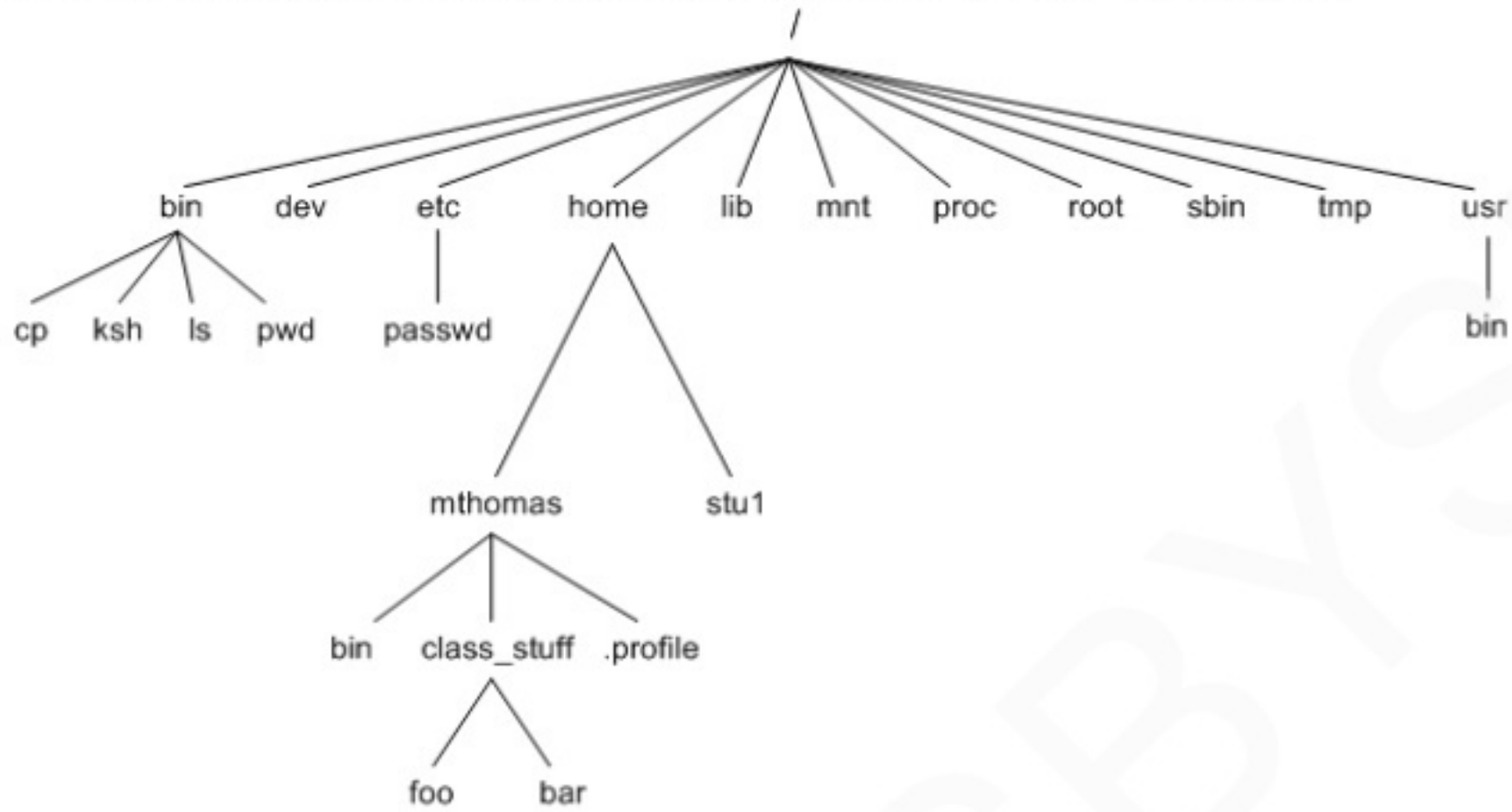


Figure 2.4: Parent Child Relationship

- The feature of UNIX file system is that there is a top, which serves as the reference point for all files.
- This top is called root (represented by a "/").
- The root directory (/) has a number of subdirectories under it.
- The subdirectories in turn have more subdirectories and other files under them.
- Every file apart from root, must have a parent, and it should be possible to trace the ultimate parentage of a file to root.
- In parent-child relationship, the parent is always a directory.

UNIX Filesystem

- Following are the directories that exist on the major versions of Unix

1) / (root directory)
- This contains only the directories needed at the top level of the file structure.

2) /bin
- This contains common programs, shared by the system, the system administrator and the users.

3) /dev
- This contains references to all the CPU peripheral hardware, which are represented as files with special properties. These are device drivers.

4) /etc
- This contains system configuration files.

5) /home
- This contains home directories of the common users.

6) /lib
- This contains library files, including files for all kinds of programs needed by system and users.

7) /sbin
- This contains programs for use by the system and the system administrator.

8) /tmp
- This contains temporary space for use by the system. This space is cleaned upon reboot. So, don't use this for saving any work.

9) /usr
- This contains programs, libraries, documentation etc. for all user-related programs.

10) /var
- This contains storage for all variable files and temporary files created by users, such as log files, mail queue or print spooler area.

1(c) Explain briefly absolute pathname and relative pathname with examples. (6 Marks)

Ans:

Relative and Absolute Pathname

- A pathname is a text string made up of one or more names separated by a "/".
- A pathname specifies how to traverse (navigate) the hierarchical directory names in the file system to reach some destination object.

1) Absolute Pathname

- An absolute pathname begins with a slash (/).
- The Absolute path defines the location of a Directory or a file from the root file system (/).
- The absolute path contains the full path to the directory or file.

2) Relative Pathname

- The relative pathname do not begin with "/".
- It specify the location relative to your current working directory.
    1) . (a single dot) - This represents the current directory.
    2) .. (two dots) - This represents the parent directory.
- Example:
- 'date' command can executed in two ways as follows:

Using Absolute Pathname	Using Relative Pathname
$ /bin/date Thu Sep 7 10:20:29 IST 2017	$ date Thu Sep 7 10:20:29 IST 2017

- Example:
- P1.java can be copied to kumar under home directory in two ways as follows:

Using Absolute Pathname	Using Relative Pathname

$ pwd
/home/kumar
$ cp /home/sharma/P1.java /home/kumar
	$ pwd
	/home/kumar
	$ cp /home/sharma/progs .

- Example:
- cd & mkdir can be used in two ways as follows:

Using Absolute Pathname	Using Relative Pathname

$ pwd
/home/kumar
$ mkdir /home/kumar/progs
$ cd /home/kumar/progs
$ pwd
/home/kumar/progs
$ cd /home/kumar
$ pwd
/home/kumar
	$ pwd
	/home/kumar
	$ mkdir progs
	$ cd progs
	$ pwd
	/home/kumar/progs
	$ cd ..
	$ pwd
	/home/kumar

2(a) Give the significance of the seven fields of the "ls -l" command. (7 Marks)

Ans:

File Attributes and Permissions and Knowing them

- ls command can be used to obtain a list of all filenames in the current directory.
- -l option can be used to obtain a detailed list of attributes of all files in the current directory.
- For example:

```
$ ls -l
Type & Perm   Link   Owner    Group    Size   Date & Time      File Name
-rwxr-xr--    1      kumar    metal    195    may 10 13:45     chap01
drwxr-xr-x    2      kumar    metal    512    may 09 12:55     helpdir
```

- The header includes the following attributes:

1) Type & Perm

- This represents the file-type and the permission given on the file.

File-type

- The first character indicates type of the file as shown below:
  - i) hyphen (-) for regular file
  - ii) d for Directory file
  - iii) l for Symbolic link file
  - iv) b for block special file
  - v) c for character special file

File Permission

- The remaining 9 characters indicates permission of the file.
- There are 3 permissions: read (r), write (w), execute (x).
  - **i) Read:** Grants the capability to read, i.e., view the contents of the file.
  - **ii) Write:** Grants the capability to modify, or remove the content of the file.
  - **iii) Execute:** User with execute permissions can run a file as a program.
- There are 3 types of users: owner, groups and others.
- The permission is broken into group of 3 characters:
  - i) The first 3 characters (2-4) represent the permissions for the file's owner.
  - ii) The middle 3 characters (5-7) represent the permissions for the group to which the file belongs.
  - iii) The last 3 characters (8-10) represents the permissions for everyone else.
- Consider an example -rwxr-xr--
  - i) Owner has read (r), write (w) and execute (x) permission.
  - ii) Group members have read (r) and execute (x) permission, but no write permission.
  - iii) Others have read (r) only permission.

2) Link

- This indicates the number of file names maintained by the system.
- This does not mean that there are so many copies of the file. (Link similar to pointer).

3) Owner

- This represents the owner of the file. This is the user who created this file.

4) Group

- This represents the group of the owner.
- Each group member can access the file depending on the permission assigned.
- The privileges of the group are set by the owner of the file and not by the group members.
- When the system admin creates a user account, he has to assign these parameters to the user:
  i) The user-id (UID) and ii) The group-id (GID)

5) Size

- This represents the file size in bytes.
- It is the number of character in the file rather than the actual size occupied on disk.

6) Date & Time

- This represents the last modification date and the time of the file.
- If you change only the permissions /ownership of the file, the modification time remains unchanged.
- If at least one character is added/removed from the file then this field will be updated.

7) File name

- This represents the file or the directory name.

2(b) What is file permission? Explain how to use "chmod" command to set the permissions in a relative manner with an example. (7 Marks)

Ans:

File Permission

• There are 3 permissions associated with a given file: read (r), write (w), execute (x).

i) Read

➢ Grants the capability to read, i.e., view the contents of the file.

ii) Write

➢ Grants the capability to modify, or remove the content of the file.

iii) Execute

➢ User with execute permissions can run a file as a program.

Changing File Permissions

• A file is created with a default set of permission.
• chmod command can be used to change permission of a file.
• This command can be used in two ways: 1) Relative mode and 2) Absolute mode.

Relative Permissions

• This command can be used to add/delete permission for specific type of user (owner, group or others).
• This command can be used to
→ change only those permissions specified in the command line and
→ leave the other permissions unchanged.
• Syntax:
chmod category operation permission filename
• This command takes 4 arguments:
1) category can be
u → user (owner) g → group o → others a → all (ugo)
2) operation can be
+ → assign
- → remove
= → absolute
3) permission can be
r → read w → write x → execute
4) Filename whose permission has to changed
• Example:

```
$ ls -l xstart
-rw-r—r-- 1 kumar metal 1906 sep 23:38 xstart
$ chmod u+x xstart            // user (u) is added(+) an execute(x) permission
$ ls -l xstart
-rwxr—r-- 1 kumar metal 1906 sep 23:38 xstart
$ chmod ugo+x xstart  OR  chmod a+x xstart  // all(a) are added(+) an execute(x) permission
$ ls –l xstart
-rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart
$ chmod go-r xstart           // group(g) & others(o) are removed(-) a read(r) permission
$ls –l xstart
-rwx—x--x 1 kumar metal 1906 sep 23:38 xstart
```

• This command can also accept multiple file names.
• Example:
$ chmod u+x note1 note2 note3

Recursively Changing File Permissions

• chmod command can be used to change recursively permission of all the files and subdirectories found in the current directory.
• Example:
chmod -R a+x c_progs // current directory c_progs
• Here, all files and subdirectories are made executable for all users in current directory c_progs.

2(c) Explain the three different modes in which "Vi" editor works. (6 Marks)

Ans:

Different Modes of vi

• Three modes of vi editor are: 1) Command-mode 2) Input-Mode and 3) Ex-Mode

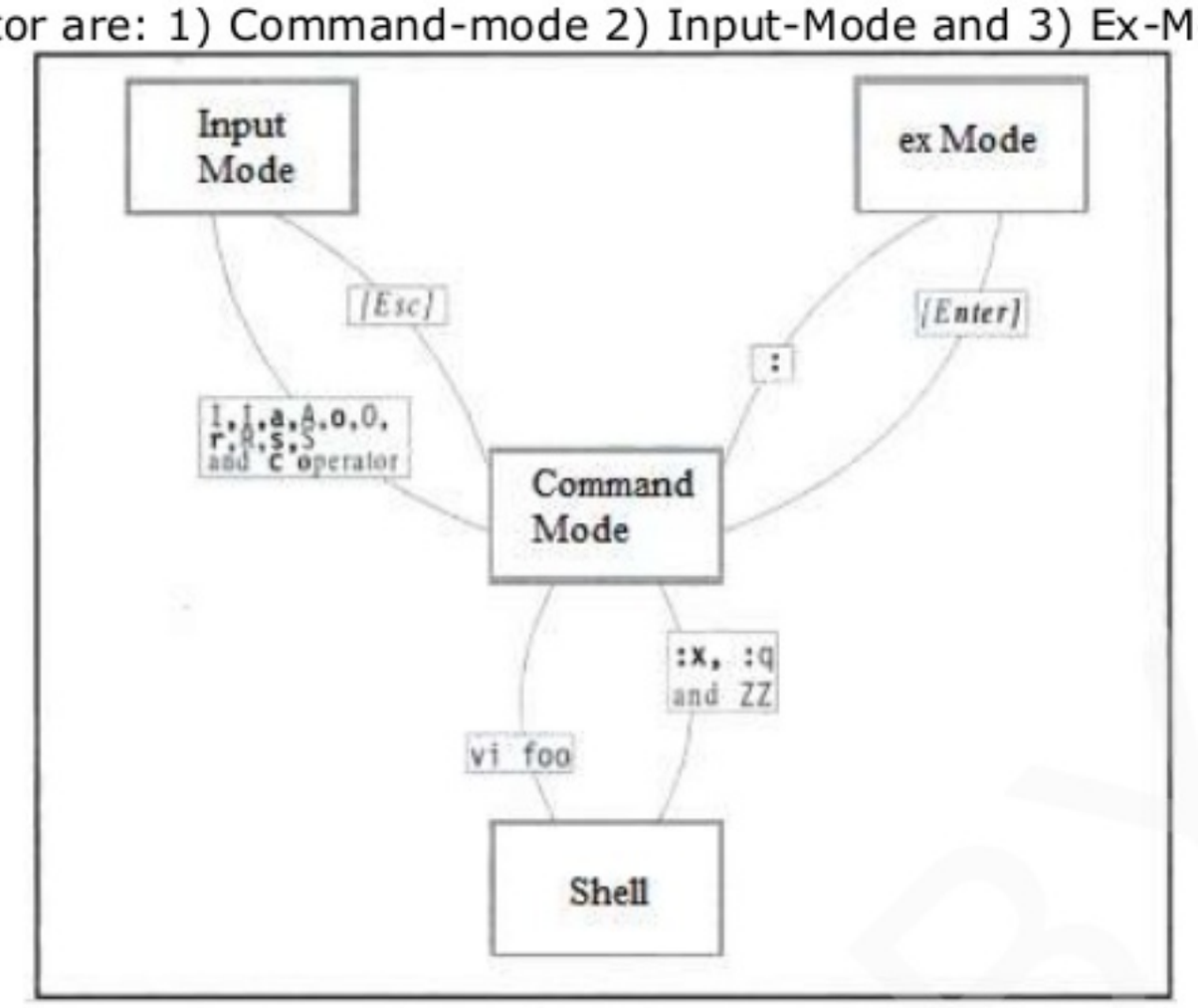


Figure 7.2: Three modes of vi editor

1) Command-Mode

• By default, the file will be opened in a command-mode.

• In this mode,

→ whatever you type is interpreted as a command

→ you can pass commands to act on text

→ you can copy(or yank) and delete text

→ you can move to other modes i.e. Input-Mode or Ex-Mode

→ you can move the cursor in the four directions (h, j, k and l)

• Some of the command-mode commands are:

Command	Function
x	to delete a text command
dd	to delete entire line
P or p	to put the character
y	to yank the text
yy	to copy entire line
u	to undo the recent actions

2) Insert Mode

• To enter text into the file, you must be in the insert mode.

• To switch from command-mode to insert mode, press the key "i".

• In this mode,

→ whatever you type is interpreted as input and placed in the file.

→ you can insert, append & replace text

• Some of the Input-Mode commands are:

Command	Function
i	inserts text
a	appends text
I	inserts at beginning of line
A	appends text at end of line
o	opens line below
O	opens line above
r	replaces a single character
s	replaces with a text
S	replaces entire line

- After a text is typed, the cursor will be positioned on the last character of the last line.
- This last line is known as current line.
- The character where the cursor is positioned is known as current cursor position.
- Input-Mode is used to handle files and perform substitution.
- To switch from insert mode to command-mode, press the key "Esc".

3) Execute Mode (Last Line Mode)

- Actually, the text entered has not been saved on disk but exists in a buffer (a temporary storage).
- To save the entered text, you must switch to the execute mode.
- In this mode,
  → whatever you type is interpreted as a ex-command
  → you can save file
  → you can quit editing mode
- Some of the sample ex commands are:

Command	Function
:W	saves file and remains in editing mode
:x	saves and quits editing mode
:wq	saves and quits editing mode
:w <filename>	save as
:w! <filename>	save as, but overwrites existing file
:q	quits editing mode
:q!	quits editing mode by rejecting changes made
:sh	escapes to UNIX shell
:recover	recovers file from a crash

- To switch from command-mode to execute mode, press the keys ":".
- Anything entered in front of the colon ":" prompt it is taken as an ex command.
- After inserting ex command, press [Enter] to execute the command.
- To switch from execute mode to command-mode, press the key "[Enter]".

3(a) Explain the standard input, standard output and standard error with respect to UNIX operating system. (7 Marks)

Ans:

Redirection : Three Standard Files

- The shell associates three standard files with the terminal:
  → two for display and
  → one for the keyboard.
- When a user logs in, the shell makes available three standard files.
- Each standard file is associated with a default device:
  1) Standard input: The file representing input which is connected to the keyboard.
  2) Standard output: The file representing output which is connected to the display.
  3) Standard error: The file representing error messages that come from the command or shell. This file is also connected to the display.

1) Standard Input

- The standard input can represent three input sources:
  1) The keyboard, the default source.
  2) A file using redirection with the < symbol.
  3) Another program using a pipeline.
- By default, the shell directs standard input from the keyboard.
- Example:

```
$wc
hello
world
<ctrl+d>                    // end of input
2 2 10                      // output
```

• The redirect input symbol (<) instructs the shell to redirect a command's input to come from the specified file instead of from the keyboard.

• Example:

```
$ cat sample.txt
hello
world
$ wc < sample.txt
2 2 10                          // output
```

2) Standard Output

• The standard output can represent three possible destinations:

1) The terminal, the default destination.
2) A file using the redirection symbols > and >>.
3) As input to another program using a pipeline.

• By default, the shell directs standard output from a command to the screen.

• Example:

```
$ cat sample.txt
hello
world
$ wc  sample.txt
2 2 10                          // output
```

• The redirect output symbol (>) instructs the shell to redirect the output of a command to the specified file instead of to the screen.

• Example:

```
$ wc  sample.txt > temp.txt
$ cat temp.txt
2 2 10                          // output of wc stored in temp.txt
```

• >> can be used to append to a existing file.

3) Standard Error

• By default, the shell directs standard error from a command to the screen.

• Example:

```
$ cat empty.txt
cat: cannot open empty.txt              // error because empty.txt is non existent file
```

• The redirect output symbol (>) instructs the shell to redirect the error messages of a command to the specified file instead of to the screen.

• Example:

```
$ cat empty.txt >> errorfile.txt
$ cat errorfile.txt
cat: cannot open empty.txt              // error message of cat stored in errorfile.txt
```

3(b) Explain the mechanism of process creation. (7 Marks)

Ans:

Process

- A process is a program in execution.
- The process is said to be born when the program starts execution.
- The process remains alive as long as the program is active.
- The process is said to be dead when the program completes execution.
- Usually, the name of the process is same as the name of the program being executed.
  For ex: A process named grep is created when grep command is being executed.
- The kernel is responsible for management of process.

Mechanism of Process Creation

1) Fork

- The parent process uses fork() to create a child process.
- After fork() executes, the parent & child have different PIDs and PPIDs.
- When fork is invoked, the kernel replicates address space of the parent process to the child process.
- When a process is forked, the child inherits following attributes of the parent.
  1) User name of the real and effective user (RUID and EUID).
  2) Real and effective group owner (RGID and EGID).
  3) The current directory from where the process was run.
  4) The file descriptors of all files opened by the parent process.
  5) Environment variables like HOME, PATH.
- '.' child runs in its own address space, changes made to these parameters don't affect the parent.
- Forking process is responsible for the multiplication of processes in the system.
- At this point, both parent & child continue execution at the statement following fork.

2) Exec

- Forking only creates a process but it doesn't execute a new program.
- The child process uses exec() to execute its own new program.
- This operation replaces the entire address space with that of the new program.
- The PID and PPID of the child remains unchanged.
- At the end of the execution, exit() is called to terminate the child.

3) Wait

- When the child process starts its execution, the parent process can do 2 things:
  1) Wait to gather the child's exit status.
  2) Continue execution without waiting for the child.
- In first case, the parent uses wait() to wait for the child to complete its task.
- Finally, parent picks up the exit status of the child and continues with other tasks.

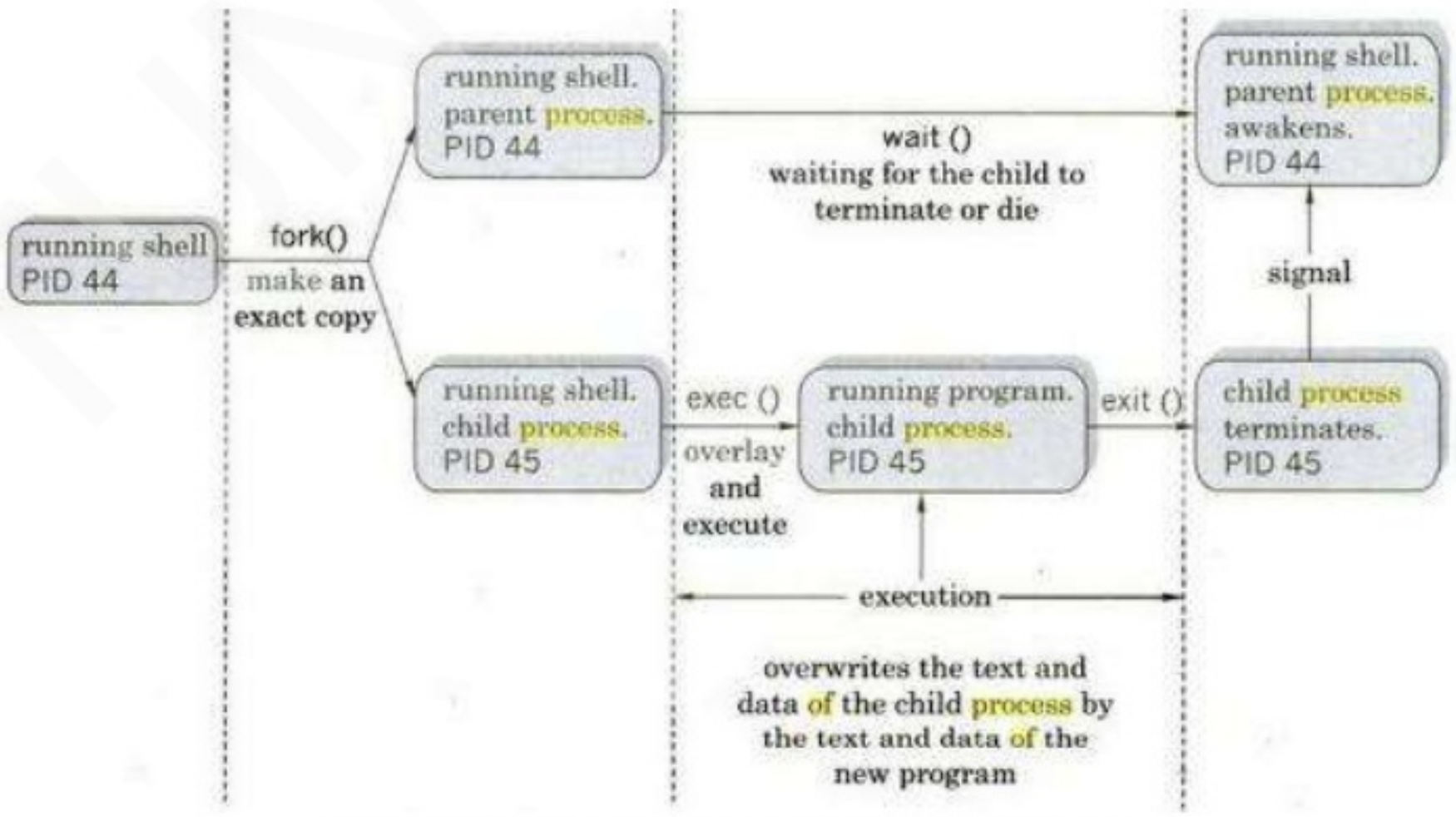


Figure 5.3.1: Mechanism of Process Creation

How the Shell is created?

• When the system goes to multiuser mode, init process calls fork to create a getty process for each active communication port (say pts/01 pts/02).
• Each getty process prints the login prompt on their terminal and then goes off to sleep.
• When a user tries to log in, getty process wakes up.
• getty process calls fork-exec which creates & executes login program to verify login name and password.
• On successful verification, login process calls fork-exec which creates & executes shell program.
• init process goes to sleep mode until termination of child processes.

Why directory change can't be made in a separate process?
• When the user logs out, it is intimated to init, which then wakes up.
• While executing cd command, if a new process is forked as a child process of the current shell, then after the cd command execution is completed, the control would revert back to the parent process(the current shell) and the original directory would be restored back.
• Hence, it would be impossible to change directories.
• Hence cd command is executed within the current shell process itself.

3(c) What are environment variables? Explain any four. (6 Marks)

Ans:

Environment Variable

- Environmental variables are used to provide information to the programs you use.
- These variables control the behavior of the system.
- They determine the environment in which the user works.
- If environment variables are not set properly, the users may not be able to use some commands.
- Environment variables are so called because they are available in the user's total environment
  i.e. the sub-shells that run shell scripts and mail commands and editors.
- Some variables are set by the system, others by the users, others by the shell programs.
- env command can be used to display environment variables.
- For example:

```
$ env
HOME=home/kumar
IFS=' '
LOGNAME=kumar
MAIL= /var/mail/kumar
MAILCHECK=60
PATH=/bin:/usr/bin
PS1='$'
PS2='>'
SHELL=/usr/bin/bash
TERM= tty1
```

1) HOME

- This variable indicates the home directory of the current user.
- This variable is set for a user by the system admin in /etc/passwd.

2) IFS

- This variable contains a string of characters that are used as word separator in the command line.
- The string normally consists of the space, tab and newline characters.

3) LOGNAME

- This variable shows the username.

4) MAIL

- This variable specifies the path to user's mailbox.

5) MAILCHECK

- This variable determines how often the shell checks the file for the arrival of new mail.

6) PATH

- This variable specifies the locations in which the shell should look for commands.
- Usually, the PATH variable can be set as follows:

  $PATH=/bin:/usr/bin

7) PS1 and PS2

- The shell has 2 prompts:

  1) The primary prompt $ is the one the user normally sees on the monitor. $ is stored in PS1.

  ➢ The user can change the primary prompt as follows:

  ```
  $ PS1="C>"
  C>                                  //similar to windows
  ```

  2) The secondary prompt > is stored in PS2.

8) SHELL

- This variable specifies the current shell being used by the users.
- Different types of shells are:

  1) Bourne shell /bin/sh 2) C-shell /bin/csh 3) Korn shell /bin/ksh

- This variable is set for a user by the system admin in /etc/passwd.

9) TERM

- This variable indicates the terminal type that is used.
- Every terminal has certain characteristics that are defined in a separate control file in the terminfo directory.
- If TERM is not set correctly, vi will not work and the display will be faulty.

4(a) Differentiate between hard link and soft link in UNIX with examples. (6 Marks)

Ans:

1) ln: Creating Hard Links

- This command can be used to create a hard link between 2 or more files.
- Syntax:

  ln filename1 filename2
- Example:

```
ln emp.lst employee                    // This command links emp.lst with employee
ls -li emp.lst employee                // To check if 2 files have the same inode number
518 -rwxr-xr-x 2 kumar group 915 may 4 09:58 emp.lst
518 -rwxr-xr-x 2 kumar group 915 may 4 09:58 employee
```

- In the third column, 2 is the link count of 2 files (emp.lst and employee).
- Example:

```
ln employee emp.dat                    // This command links employee with emp.dat
ls -li emp*                            // To check if 3 files have the same inode number
518 -rwxr-xr-x 3 kumar group 915 may 4 09:58 emp.dat
518 -rwxr-xr-x 3 kumar group 915 may 4 09:58 emp.lst
518 -rwxr-xr-x 3 kumar group 915 may 4 09:58 employee
```

- In the third column, 3 is the link count of 3 files (emp.lst employee and emp.dat).
- Multiple files can be linked together. Here, the destination filename must be a directory.
- Example:

```
ln chap1 chap2 chap3 content           // "content" is a directory
```

- A file is considered to be completely removed from the file system when its link count drops to 0.

2) Soft Links

- Soft links are files that hold the pathname of the original file.
- These links doesn't hold the file's content. Rather, they hold the pathname of the file that actually has the content.

ln -s: Creating Soft Links

- This command can be used to create a soft link between 2 or more files.
- Syntax:

  ln -s filename1 filename2
- Example:

```
ln -s note note.sym                    // This command links note with note.sym
ls -li note note.sym
9948 -rw-r--r--      1 kumar group 80 feb 16 14:52 note
9952 lrwxrwxrwx      1 kumar group 04 feb 16 15:07 note.sym -> note
```

- In the second row,

  "l" indicate symbolic link

  -> indicates note.sym contains the pathname for the note

  4 indicates size of symbolic link
- Since 2 files have the different inode numbers, they are not identical.
- If note is deleted, its data will be lost. In this case, note.sym will point to a nonexistent file and becomes a dangling symbolic link (dangling pointer).
- Like other files, a symbolic link has a separate directory entry with its own inode number.

Soft link	Hard link
ln -s	ln
Can create soft link for both files and directories.	Files only.
Symbolic link points the link to the file or directory name.	Hard link is the reference or pointer to the exact file.
The link will not be accessible if the original file or folder is deleted/removed.	Can access.
Different inode value.	Same inode number.
Soft link is possible in different partition.	Not possible.

4(b) Explain sort command briefly. Also discuss its important options with eg. (6 Marks)

Ans:

sort

- Sorting is the ordering of data in ascending or descending sequence.
- This command orders a file.
- By default, it reorders lines in ASCII collating sequence i.e. whitespace first, then numbers, then uppercase letters and finally lowercase letters.
- It can identify fields and it can sort on specified fields.
- Syntax:

  sort filename
- Example:

  $ sort emp.lst
- This default sorting sequence can be altered by using certain options.

sort Options

Sorting Primary Key (-k)

- –k option can be used to sort on a specific field (primary key).
- Example:

  sort –t "|" –k 2 student.lst //primary key is second field which will be sorted

The Sort order in Reverse (-r)

- –r option can be used to sort in reverse order .
- Example:

  sort -t "|" -r -k 2 student.lst

Sorting on Secondary Key

- –k option can also be used to sort on more than one field. (primary key & secondary key).
- Example:

  sort –t "|" –k 5,5 –k 2,2 student.lst //primary key is third field, secondary key is second field
- Here, –k 2,2 indicates sorting starts on the second field and ends on the same field.

Sorting on Columns

- –k option can also be used to sort specific column position within a field.
- Example:

  sort –t "|" –k 5.1,5.2 student.lst //sort on & 2nd column positions within 5th field.
- General format: -k m.n where n is the character position in the mth field.

Numeric Sort (-n)

- –n option can be used to sort on numeric values rather than in ASCII collating sequence.
- Example:

  sort –n student.lst

Removing Repeated Lines (-u)

- –u option can be used to remove repeated (duplicate) lines from a file.
- Only one of the duplicate entries is retained in the output.
- Example:

  cut -d "|" -f 3 student.lst | sort -u | tee desig.txt

Specify Output Filename (-o)

- –o (output) option is used to specify the output filename.
- The input and output filenames can also be the same.
- Example:

  sort –o student.lst student.out // output filename is student.out
  sort –o student.lst -k 1 student.out // sort on first field & output to file student.out

Check for Default Order(-c)

- -c (check) option is used to check whether the file has actually been sorted in the default order.
- Example:

  sort –c student.lst

Merge Sorted Lists(–m)

- –m (merge) option can be used to merge 2 or more files that are sorted individually.
- Example:

  sort –m file1 file2 file3

4(c) Explain the following commands with example: i) head ii) tr iii) uniq iv) find (8 Marks)

Ans(i):

head

- This command is used to display the top of the file.
- By default, it displays the first 10 lines of the file.
- It is mainly useful to verify the contents of a file.
- Syntax:

  head [count option] filename
- Example:

  head emp.lst # displays first 10 lines of emp.lst

head Options

- -n option is used to specify a line count and to display the first n lines of the file.
- Example:

```
$ head -n 3 student.lst                    // displays first 3 lines of student.lst
4       | MH   | 10   | IS    | 111
4       | MH   | 11   | CS    | 401
4       | GW   | 11   | CS    | 402
```

- -t option is used to display filenames in order of their modification time.

Ans(ii):

tr

- This command can be used to translate characters using one or two expressions.
- Syntax:

  tr expression1 expression2 < standard_input
- It takes input only from standard input, it doesn't take a filename as argument.
- By default, it translates each character in expression1 to its mapped counterpart in expression2.
- Example:

  tr 'abc' 'ABC' < emp.lst | head –n 3 //replace abc by ABC

tr Options

- -s can be used for compressing multiple consecutive characters.
- Example:

  tr –s ' ' < emp.lst | head –n 3 //replace multiple whitespace by single whitespace
- -d can be used for deleting characters.
- Example:

  tr –d '|' < emp.lst | head –n 3 //delete "/" in the file emp.lst
- -c can be used for complementing values of expression.
- Example:

  tr –cd '|' < emp.lst //delete all characters except "/" in the file emp.lst

Ans(iii):

uniq

- This command can be used to remove duplicate lines in a file.
- This command simply fetches one copy of each line from a file and writes it to the standard output.
- Since uniq requires a sorted file as input, the general procedure is to sort a file and pipe its output to uniq.
- Example:

  sort emp.lst | uniq – uniqlist // the output is saved in a file

uniq Options

- -u can be used for selecting the nonrepeated lines.
- Example:

  cut –d "|" –f3 emp.lst | sort | uniq –u
- -d can be used for selecting the duplicate lines.
- Example:

  cut –d "|" –f3 emp.lst | sort | uniq –d
- -c can be used for counting frequency of occurrence.
- Example:

  cut –d "|" –f3 emp.lst | sort | uniq –c

Ans(iv):

find

- This command
  - → recursively examines a directory tree to look for files matching some criteria and
  - → then takes some action on the selected files.
- Syntax:

  find path_list selecton_criteria action
- Here,

  path_list consists of one or more directories to be examined
  selection-criteria contains conditions used for matching a file
  action specifies some action to be taken on the selected files
- Example:

  $find / -name a.out -print // locates all files named a.out
  /home/kumar/a.out
  /home/user/a.out
- Here,
  1) path_list **/** indicates that the search should start from root directory.
  2) Then, each file in the list is matched against the selection criteria.
     The selection criteria always consists of an expression in the form –operator argument.
     If the expression matches the file, the file is selected.
  3) Finally, a display selected files on the terminal.

5(a) Explain 'grep' command with its options. (8 Marks)

Ans:

grep

- This command can be used to search a file(s) for lines that have a certain pattern.
- This command
  - → scans the file for a pattern and
  - → displays
    - i) lines containing the pattern or
    - ii) line numbers
- Syntax:

  grep pattern file(s)
- Example:

  grep "MH" student.lst // display lines containing "MH" from the file student.lst
- Patterns with and without quotes is possible.
- Quote is mandatory when pattern involves more than one word.
- Example:

  grep "My Document" student.lst // display lines containing "My Document" from student.lst
- This command can be used with multiple filenames.
- Example:

  grep "MH" student.lst vtu.lst rank.lst

grep Options

- To understand the working of different options, let us consider we have following information in file student.lst.

```
$ cat student.lst
4       | MH   | 10    | IS    | 111
4       | MH   | 11    | CS    | 401
4       | GW   | 11    | CS    | 402
4       | VV   | 11    | CS    | 403
```

Ignores Case

- -i (ignore) option can be used to search all lines containing a pattern regardless of uppercase and lowercase distinction
- Example:

```
$ grep -i "MH" demo_file            // matches all the words such as MH mH Mh mh
4       | MH   | 10    | IS    | 111
4       | MH   | 11    | CS    | 401
```

Deleting Lines

- -v option can be used to print all lines that do not contain the specified pattern in a file.
- Example:

```
grep -v 'MH' student.lst
4       | GW   | 11    | CS    | 402
4       | VV   | 11    | CS    | 403
```

Displaying Line Number

- -n (line number) option can be used to display line numbers containing the pattern.
- Example:

```
grep -n 'MH' student.lst
1
2
```

Counting Lines Containing the Pattern

- -c (line count) option can be used to count number of lines containing the pattern.
- Example:

  grep -c 'MH' student.lst

  2

Displaying Filenames

- -l (list filename) option can be used to list out the files containing the pattern.
- Example:

  grep -l 'MH' *.lst

Matching Multiple Pattern

- -e option can be used to match multiple pattern in a file.
- Example:

  grep -e 'MH' -e 'VV' student.lst

Taking Pattern From File

- -f option can be used to place all matched pattern in a separate file, one pattern per line.

5(b) Explain line addressing and context addressing in "sed" with examples. (6 Marks)

Ans:

sed – The Stream Editor

- sed is a multipurpose tool which combines the work of several filters.
- sed uses **instructions** to act on text.
- An instruction combines an **address** for selecting lines, with an **action** to be taken on them.
- Syntax:

  sed options 'address action' file(s)
- Two types of addressing: 1) Line Addressing and 2) Context Addressing,

1) Line Addressing

sed '3q' emp lst same as head –n 3 emp lst

- The above command selects first three lines and quits.

  sed –n '4,10p' emp.lst
- The above command selects lines starting from 4th till 10th.

  sed –n '$p' emp.lst
- The above command selects last line of the file

  sed –n '1,3p 7,9p' emp.lst
- The above command selects multiple groups of lines.(1 to 3 and 7 to 9)

  sed –n '4,$!p' emp.lst
- The above command selects all lines except 1 to 3. Here, we are negating the action, just same as 1,3p.

2) Context Addressing

- We can specify one or more patterns to locate lines

  sed –n '/director/p' emp.lst
- We can also specify a comma-separated pair of context addresses to select a group of lines.

  sed –n '/dasgupta/,/saxena/p' emp.lst
- Line and context addresses can also be mixed

  sed –n '1,/dasgupta/p' emp.lst

Using regular expressions

- Context addresses also uses regular expressions.

  sed –n '/[aA]gg*[ar][ar]wal/p' emp.lst
- The above command selects all agarwals.

  sed –n '/sa[kx]s*ena/p /gupta/p' emp.lst
- The above command selects saxenas and gupta.
- We can also use ^ and $, as part of the regular expression syntax.

  sed –n '/50.....$/p' emp.lst
- The above command selects all people born in the year 1950.

5(c) What are extended regular expression (ERE)? Explain any four ERE set used by "grep" and "egrep". (6 Marks)

Ans:

Extended Regular Expression (ERE) and egrep

- ERE can be used to match dissimilar patterns with a single expression.
- This uses some additional characters as listed below:

Metacharcter/ Character class	Match
ch+	one or more occurrences of character ch
ch?	zero or one occurrence of character ch
exp1|exp2	exp1 or exp2
(x1|x2)x3	x1x3 or x2x3

- If current version of grep doesn't support ERE, then use egrep but without the –E option.
- -E option treats pattern as an ERE.
- Example:

```
$ grep  -E  "isaa*c" demo_file                    // matches isac isaac or  issaac
$ grep –E  'vijaykumar | jayakumar' demo_file     // matches multiple patterns
$ grep –E  '(vijay | jaya) kumar' demo_file       // matches multiple patterns
```

6(a) Explain the use of "test" and [] to evaluate an expression in shell (6 Marks)

Ans:

test Command and its Shortcut

• Usually, if-construct cannot directly handle the true or false value returned by evaluation of an expression.

• So, test command can be used to handle the true or false value returned by evaluation of an expression.

• Test command
  → uses certain operators to evaluate the condition on its right and
  → returns either a true or false exit status.

• Then, if-construct uses the exit status for making decisions.

• Test command
  → does not display any output
  → sets the parameter $? (exit status).

• Test command works in 3 ways:
  1) Compare two numbers. 2) Compares two strings. 3) Checks files attributes.

1) Numeric Comparison

Operator	Meaning
-eq	Equal to
-ne	Not equal to
-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than or equal

• Syntax:
  test $op1 -operator $op2

• Operators always begin with a – (Hyphen) followed by a two character word .

• Numeric comparison can be done on integer values only. (The decimal values are truncated).

Shorthand for test

• [and] can be used instead of test.
  Test $x –eq $y is equivalent to [$x –eq $y]

2) String Comparison

• Test command is also used for testing strings.

Operator	True if
s1=s2	String s1=s2
s1!=s2	String s1 is not equal to s2
-n stg	String stg is not a null string
-z stg	String stg is a null string
stg	String stg is assigned and not null

3) File Tests

• Test command can be used to check various file attributes such as file type (-, d or l) & file permission (r, w, x).

Test	True if
-e file	File exists
-f file	File exists and is a regular file
-d file	File exists and is a directory
-L file	File exists and is a symbolic link
-r file	File exists and readable
-w file	File exists and is writable
-x file	File exists and is executable
-s file	File exists and has a size greater than zero
f1 –nt f2	File f1 is newer than f2
f1 –ot f2	File f1 is older than f2

6(b) Explain the shell features of i) 'while' and ii) 'for' with syntax. (6 Marks)

Ans (i):

while Statement

- while loop can be used to execute a set of statements repeatedly as long as a given condition is true.
- Syntax:

```
while condition is true
do
        execute statements
done
```

- The statements enclosed between do and done are executed repeatedly as long as condition is true.
- Example: A script to display a message 3 times using while loop.

```
#! /bin/sh
num=1
while [$num -le 3]
        do
                echo " Welcome to Shell Programming "
                expr $num = $num +1;
        done
```

```
Output:
Welcome to Shell Programming
Welcome to Shell Programming
Welcome to Shell Programming
```

Ans (ii):

for Statement

- for loop can be used to iterate over all items(or strings) within a list.
- Syntax:

```
for variable in list
        do
                statements
        done
```

- Here, list consists of a set of items(or strings).
- Each item of the list is picked up and assigned to the "variable".
- The iteration continues until all items are picked from the array.
- Example: A script to display elements of an array.

```
#! /bin/sh
print("Here are the numbers in the list: \n");
for var in 10 20 30 40 50 60;
        do
                echo "$var  \t"
        done
```

```
Output:
Here are the numbers in the list
10 20 30 40 50 60
```

6(c) Explain the "expr" command applicable to computation and string functions. (8 Marks)

Ans:

expr: Evaluate an Expression

- expr command can be used to
  - → evaluate an expression and
  - → output the corresponding value.
- This command combines the following two functions:
  1) Performs arithmetic operations on integers and
  2) Manipulates strings.

1) Numeric Computation

- Five operators used on integers: +, -, *, / and %.
- Syntax:

  expr $op1 operator $op2
- Example:

```
$ x=5 y=3
$ expr  $x + $y              // outputs 8
$ expr  $x - $y              // outputs 2
$ expr $x \* $y          // * must be escaped to prevent shell from interpreting * as wildcard
                             //outputs 15
$ expr  $x / $y              //outputs 1
$ expr  $x % $y              // outputs 2
$ z =`expr $x + $y`          // command substitution to assign a variable
$ echo $z                    // outputs 8
```

2) String Handling

- Three functions used on strings:
  - i) Finding length of string
  - ii) Extracting substring
  - iii) Locating position of a character in a string
- Syntax:

  expr "exp1" : "exp2"
- On the left of the colon (:), the string to be worked upon is placed.

  On the right of the colon(:), a regular expression is placed.

i) Length of the String

- The regular expression ".*" is used to print the number of characters matching the pattern.
- Syntax:

  expr "string" : ".*"
- Example:

  $ expr "vtunotesbysri" : '.*' // outputs 13

ii) Extracting a Substring

- expr command can be used to extract a string enclosed by the escape characters "\(" and "\)".
- Syntax:

  expr "string" : "\(substring \)"
- Example:

  $ expr "vtunotesbysri" : " \(sri \)" // outputs 'sri'

iii) Locating Position of a Character

- expr command can be used to find the location of the first occurrence of a character inside a string.
- Syntax:

  expr "string" : "[^ch]*ch" //ch → character
- Example:

  $ expr "vtunotesbysri" : "[^u]*u" // outputs 3

7(a) i) What is AWK? ii) Explain any three built in functions in AWK (7 Marks)

Ans(i):

• awk is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers.

• It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as

--> writing the line to the standard output or

--> incrementing a counter each time it finds a match.

• Syntax:

awk option 'selection criteria {action}' file(s)

• Here, selection_criteria filters input and selects lines for the action component to act upon. The selection_criteria is enclosed within single quotes and the action within the curly braces. Both the selection criteria and action forms an awk program.

• Example:

```
awk '/manager/ { print }' emp.lst
```

Ans(i):

Built in functions

1) Arithmetic Functions

i) int

• This function can be used to calculate the integral portion of a number (without rounding off).

• Example:

```
param = 5.12345
result = int(param)          //now   result = 5
```

ii) sqrt

• This function can be used to calculate square root of a number.

• Example:

```
param = 144.0
result = sqrt(param)         //now   result = 12
```

2) String Handling Function

i) length

• This function can be used to determine length of a string.

• Example:

```
str = "Hello World"
len = length(str)            //now   len = 11
```

ii) index(s1, s2)

• This function can be used to determine the position of a string s2 within a larger string s1.

• Example:

```
x = index ("abcde", "b")             //now   x = 2
```

iii) substr (stg, m, n)

• This function can be used to extract a substring from a string stg.

• Here, m represents the starting point of extraction and
n represents the number of characters to be extracted.

• Example:

```
str = "Hello, World"
subs = substr(str, 1, 5)             // now subs = Hello
```

iv) split(stg, arr, ch)

• This function can be used to break up a string stg on delimiter ch & store the fields in an array arr[].

• Example:

```
str = "One,Two,Three,Four"
split(str, arr, ",")                 // now  arr[4]={ One Two Three Four }
```

3) system

• This function executes the specified command and returns its exit status.

• Example: The following example displays the current date and also shows the return status.

```
awk 'BEGIN { ret = system("date"); print "Return value = " ret }'
```

• On executing this code, you get the following result:

```
Sun Dec 21 23:16:07 IST 2014                    Return value = 0
```

7(b) Write short notes on i) operators and ii) expressions in AWK. (6 Marks)

Ans(i):

1) Operators

Comparison Operators

- awk also provides the comparison operators like >, <, >=, <= ,==, !=, etc..,
- Example

```
awk 'BEGIN { a = 10; b = 10; if (a == b) print "a == b" }'      // prints a == b
awk 'BEGIN { a = 10; b = 20; if (a < b) print "a < b" }'       // prints a < b ~ and !~ :
```

Regular Expression Operator (metacharacters)

- In awk, metacharacters can be used with regular expression to increase the power and versatility of regular expressions.
- Following two metacharacters are used:

1) Match

- It is represented as ~.
- It looks for a field that contains the match string.
- Example: The following example prints the lines that contain the pattern 9.

```
awk '$0 ~ 9' marks.txt
```

2) Not Match

- It is represented as !~.
- It looks for a field that does not contain the match string.
- Example: The following example prints the lines that do not contain the pattern 9.

```
awk '$0 !~ 9' marks.txt
```

- The operators ~ and !~ work only with field specifiers like $1, $2, etc.,.

Ans(ii):

Expressions

- Expression consists of strings, numbers and variables combined by operators.
- Example:

```
(x+2)*y, x-15, x/y, etc..,
```

- awk does not have any data types and every expression is interpreted either as a string or a number.
- However awk has the ability to make conversions whenever required.
- Expressions also have true and false values associated with them.
- A nonempty string or any positive number has true value.
- Example:

```
if(c)                    //This is true if c is a nonempty string or positive number.
```

- A variable is an identifier that references a value.
- To define a variable, you only have to name it and assign it a value.
- The name can only contain letters, digits, and underscores, and may not start with a digit.
- Case distinctions in variable names are important: Salary and salary are two different variables.
- Example:

```
X= "4"
X= "3"
Print X                          // outputs 4
Print x                          // outputs 3
```

- Strings are enclosed within double quotes and can contain any character.
- Strings that do not consist of numbers have a numeric value of 0.
- Example :

```
z = "Hello"
print z                          // outputs Hello
```

7(c) Explain built-in variables in AWK. (7 Marks)

Ans:

Built-in variables

• awk has several built-in variables.

• These variables are all assigned automatically, though it is also possible for a user to reassign some of them.

• Some built-in variables are:

1) NR

• This variable represents the number of the current record.

• Example: The following example prints the record if the current record contains less than three fields.

```
echo -e "One Two\nOne Two Three " | awk 'NR < 3'        //prints "One Two"
```

2) FS

• This variable represents the (input) field separator and its default value is space.

• You can also change this by using -F command line option.

• This variable must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

• Example:

```
BEGIN { FS = ”|” }
```

3) OFS Variable

• This variable represents the output field separator and its default value is space.

• You can also change this by using -F command line option.

• This variable must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

• Example:

```
BEGIN { OFS=”~” }
```

4) NF variable

• This variable represents the number of fields in the current record.

• Example: The following example prints only those lines that contain more than two fields.

```
echo -e "One Two\nOne Two Three\n" | awk 'NF > 2'      //prints "One Two Three"
```

5) FILENAME

• This variable represents the current file name.

• Like grep and sed, awk can also handle multiple filenames in the command line.

• By default, awk doesn’t print the filename, but you can instruct it to do so.

• Example:

```
awk 'END {print FILENAME}' marks.txt                  //prints "marks.txt"
```

• With FILENAME, you can device logic that does different things depending on the file that is processed.

6) ARGC

• This variable represents the number of arguments provided at the command line.

7) ARGV

• This variable represents an array that stores the command-line arguments.

8(a) List the string handling functions in PERL. Write a program to find number of characters in a word as well as to print reverse of a given string. (8 Marks)

Ans:

String Handling Functions

• Following are some string handling functions:

1) length(): determines the length of its argument.
2) index(s1, s2): determines the position of a string s2 within string s1.
3) substr(str,m,n): extracts a substring from a string str,
where m represents the starting point of extraction and
n indicates the number of characters to be extracted.
substr() can also extract characters from the right of the string, and insert or replace a string.
4) uc(str): converts all the letters of str into uppercase.
5) ucfirst(str): converts first letter of all leading words into uppercase.
6) reverse(str): reverses the characters contained in string str.

• Example: A perl script to illustrate the use of string manipulation functions

```
#!/usr/bin/perl
$x =" abcdijklm";
print length($x);              # outputs 9
print index($x,j);             # outputs 5
substr($x,4,0) ="efgh";        # inserts "efgh" into the string $x from 4th position on the left
                               # 0 indicates non-replacement
print "$x";                    # outputs $x  as abcdefghijklm
$y = substr($x,-3,2);          # extracts 2 characters from the 3rd position on the right
print "$y"                     # outputs $y as kl
$name ="rama";
print uc($name);               #displays RAMA
print ucfirst($name);          # displays Rama
print reverse($name);          # displays amar
```

A program to find number of characters in a word as well as to print reverse of a given string.

```
#!/usr/bin/perl
print ("Enter a string: \n");
$str = <STDIN>;
$len=length($x);
print length is $len \n;
$rev=reverse($x);
print reverse is $rev \n;
```

```
Output:
Enter a string:
rama
length is 4
reverse is amar
```

8(b) Explain i) split() and ii) chop() functions with examples. (6 Marks)

Ans(i):

split()

- split function can be used to break up a line or expression into fields.
- After splitting, the fields can be assigned to
  - → a set of variables ($var1, $var2, $var3) or
  - → an array (@arr).
- Syntax:

```
($var1, $var2, $var3 ……. ) = split( /sep/ , str );
@arr = split( /sep/ , str );
```

- Here, the above 2 lines split the string "str" on the pattern "sep"
  - "sep" is a separator which can be a regular expression or a literal string
  - "str" is a string on which split function works

Default Behaviour

1) When no string is specified explicitly, the split function works on the default variable $_

- Syntax:

```
@arr = split( /sep/ );
```

2) When no field separator is specified explicitly, white spaces are taken as the field separator by default.

- Syntax:

```
@arr = split( str );
```

Splitting into Variables

- After splitting, the fields can be assigned to a set of variables ($var1, $var2, $var3).
- The field will be stored in a scalar-variable as a separate elements.
- Example: A perl script to split command line arguments.

```
#!/usr/bin/perl
print("enter 3 numbers: \n");
$numstring = <STDIN>;
if ($numstring eq " ");
        die("Nothing entered \n")
else
        ($f_num, $s_num, $t_num) = split( / /, $numstring);
print("the third, second and first numbers are \n");
print("$t_num, $s_num and $f_num ");
```

```
Output:
Enter 3 numbers: 100 200 300
The third, second and first numbers are 300, 200 and 100
```

Splitting into an Array

- After splitting, the fields can be assigned to an array (@arr).
- The field will be stored in a array-variable as a separate elements.
- Example: A perl script to split elements of an array.

```
#!/usr/bin/perl
$numstring = (100, 200, 300)
($var1, $var2, $var3) = split( /,/ ,  $numstring);
print("the third, second and first numbers are \n");
print("$t_num, $s_num and $f_num ");
```

```
Output:
The third, second and first numbers are 300, 200 and 100
```

Ans(ii):

chop()

- chop() function can be used to remove or eliminate the last character (new line character).
- chop() returns the character removed or discarded.
- Example: A perl script to demonstrate the use of chop function.

```
#!/usr/bin/perl
print("enter your name:");
$name = <STDIN>;
chop($name);                                    # removes newline character
if($name ne " ")
{
        print ("$name, have a nice day\n");
}
else
{
        print ("you have not entered your name\n");
}
```

```
Output:
Enter your name: rama
rama, have a nice day
```

8(c) Explain file handling in PERL. (6 Marks)

Ans:

File Handles and Handling File

1) File Handle

- File handle is a program variable that acts as a reference between
  1) perl program and
  2) Operating system's file structure.
- File handle names
  → do not begin with the special characters and digits
  → preferably uses uppercase letters.
- Streams are default file handles which are referred as STDIN, STDOUT and STDERR.
  1) STDIN connects to keyboard.
  2) STDOUT & STDERR connect to display screen.
- NULL file handle can be used to allow scripts to get input from
  → STDIN or
  → files listed on the command line.
- NULL file handle is written as <>, and is called diamond operator, angle operator or line-reading operator.
- Example:

```
perl -e 'while (<>)  { print ; }'              // input is taken from keyboard
perl -e 'while (<>)  { print ; }' test.txt'    // input is taken from file
```

2) File Handling Functions

open()

- open( ) function can be used to create a new file or to open an existing file.
- Syntax:

```
open(FILEHANDLE, "access_mode filename");
```

access_mode	Meaning
<	Opens an existing text file for reading purpose
>	Opens a text file for writing, if it does not exist then a new file is created
>>	Opens a text file for writing in appending mode
+>	Opens a text file for reading purpose
+<	Opens a text file for writing purpose

- Example:

```
open(INFILE, "<file1.txt");          // Opens the file file1.txt in read only format.
open (OUTFILE, ">>file2.txt");       // Opens the file file2.txt in append mode.
```

close()

- close( ) function can be used to close an opened file.
- Syntax:

```
close(FILEHANDLE);
```

die()

- die() function can be used to quit the script and display a message for the user to read.
- Syntax:

```
die(LIST);
```

- The elements of LIST are printed to STDERR, and then the script will exit, setting the script's return value to $! (errno).
- Perl script to copy the first three lines of one file "file1.txt" into another file "file2.txt".

```
#!/usr/bin/perl
open(INFILE,"<file1.txt") || die("Cannot open file");        # Open the file1.txt in read mode
open(OUTFILE, ">file2.txt");                                 # Open the file2.txt in write mode
while(<INFILE>)                                             #INFILE & OUTFILE are file handles
{
        print OUTFILE if(1..3);
}
close(INFILE);
close(OUTFILE);
```

USN | | | | | | | | | | |

10CS44

Fourth Semester B.E. Degree Examination, Dec.2013/Jan.2014
UNIX and Shell Programming

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting at least TWO questions from each part.***

PART – A

1 a. Describe briefly the major features of the UNIX operating system. **(08 Marks)**

b. Define a file. With examples, explain the three categories of files supported by UNIX. **(06 Marks)**

c. Briefly describe:
i) System calls ii) PATH iii) HOME. **(06 Marks)**

2 a. Explain the significance of all the fields of *ls –l* output. Which of the attributes can be changed only by the super user? **(08 Marks)**

b. With a neat diagram, explain the three modes of vi editor. **(06 Marks)**

c. Assuming that a file's current permissions are rw – r – x r – –, specify the chmod expression (using both relative and absolute methods) required to change them to:
i) rwx rwx rwx
ii) r – – r – – – – –
iii) – – – r – – r – – **(06 Marks)**

3 a. Devise wild – card patterns to match filenames:
i) Comprising of atleast three characters where the first char is numeric and the last char is not alphabetic.
ii) With three character extensions except the ones with .log extension.
iii) Containing 2004 as an embedded string except at the beginning or end. **(06 Marks)**

b. Explain the three distinct phases of process creation. How is the shell created? **(08 Marks)**

c. What are environment variables? Briefly describe any five of them. **(06 Marks)**

4 a. Distinguish between hard links and symbolic links with suitable examples. **(08 Marks)**

b. Describe the sort filter and illustrate its usage with –k, –u, –n, –r and –c options. **(06 Marks)**

c. i) Use find to locate all files named a.out and all C source files in your home directory tree and remove them interactively.
ii) Display only the names of all users who are logged in and also store the result in users.txt.
iii) Invoke the vi editor with the last modified file. **(06 Marks)**

PART – B

5 a. Explain with suitable examples, the sed filter along with its two forms of addressing. Also describe in brief the substitution feature provided by sed. **(08 Marks)**

b. Describe the grep filter along with any five options. **(06 Marks)**

c. i) Use sed to delete all blank lines from a file named sample.
ii) Use grep to list only the sub-directories in the current directory.
iii) Replace all occurrences of the word "UNIX" with "LINUX" in a file named sample. **(06 Marks)**

6 a. Define a shell script. What are the two ways of running a shell script? Write a shell script to accept pattern and a file and search for the pattern in the file. **(08 Marks)**

b. Explain the shell's for loop giving the possible sources of the list. **(06 Marks)**

c. Write a menu-driven shell script to perform the following:
   i) List of users who are logged in.
   ii) List of files in the current directory.
   iii) List of processes of user.
   iv) Today's date.
   v) Quit to UNIX. **(06 Marks)**

7 a. Describe the awk filter with syntax and example. How are awk arrays different from the ones used in most programming languages? **(08 Marks)**

b. Explain the looping constructs supported by awk. **(06 Marks)**

c. Briefly describe the built-in functions supported by awk for arithmetic and string operations. **(06 Marks)**

8 a. With examples, explain the string handling functions supported by perl. **(08 Marks)**

b. How are split and join used in perl scripts? **(06 Marks)**

c. Write a perl script to determine whether a year is leap year or not. **(06 Marks)**

* * * * *

UNIX AND SHELL PROGRAMMING SOLVED PAPER DEC -2013

1 (a) Describe briefly the major features of the UNIX operating system. (8 Marks)

Ans:

Features of Unix

1) Multiuser

- UNIX is a multiprogramming system.
- Multiple users can access the system by connecting to points known as terminals.
- Several users can run multiple programs simultaneously on one system.

2) Multitasking

- UNIX is a multitasking system.
- A single user can also run multiple tasks concurrently.
- For example:

  At the same time, a user can
  - → edit a file
  - → print another file one on the printer
  - → send email to a friend and
  - → browse www
- The kernel is designed to handle a user's multiple needs.
- In a multitasking environment, a user sees one job running in the foreground; the rest run in the background.
- User can switch jobs between background and foreground, suspend, or even terminate them.

3) Pattern Matching

- UNIX has very sophisticated pattern matching features.
- Regular Expressions are a feature of UNIX.
- Regular Expressions describe a pattern to match, a sequence of characters, not words, within a line of text.

4) Portable

- UNIX can be installed on many hardware platforms.
- Unix operating system is written in C language, hence it is more portable than other operating systems.

5) UNIX Toolkit

- UNIX offers facility to add and remove many applications as and when required.
- Tools include
  - → general purpose tools
  - → text manipulation tools
  - → compilers/interpreters
  - → networked applications and
  - → system administration tools

6) Programming Facility

- The UNIX shell is also a programming language; it was designed for programmer, not for end user.
- It has all the necessary ingredients, like control structures, loops and variables, that establish powerful programming language.
- This features are used to design shell scripts – programs that can also invoke UNIX commands.
- Many of the system's functions can be controlled and automated by using these shell scripts.

7) Documentation

- The principal on-line help facility available is the man command, which remains the most important references for commands and their configuration files.
- Apart from the man documentation, there's a vast ocean of UNIX resources available on the Internet.

UNIX AND SHELL PROGRAMMING SOLVED PAPER DEC -2013

1 (b) Define a file with example, explain 3 categories of files supported by UNIX. (6 Marks)

Ans:

UNIX Files

- A file is the container for storing information.
- The file doesn't store 1) file-size and 2) file-name.
- Some attributes of file are file-type, permissions, links, owner, group-owner etc.
- These file attributes are stored in inode table which is accessible only to kernel.
- Programs, services, texts, images etc are considered to be files.

Basic File Types

- Three categories of files are:
  1) Ordinary file (or regular file) – It contains only data as a stream of characters.
  2) Directory file – it contains files and other sub-directories.
  3) Device file – all devices and peripherals are represented by files.

1) Ordinary (Regular) File

- An ordinary file is a file on the system that contains data, text, or program instructions.
- An ordinary file can be either a text file or a binary file.
  1) A text file contains only printable characters and you can view and edit them.
  - Examples: All C and Java program sources, shell scripts are text files.
  - Every line of a text file is terminated with the newline character.

  2) A binary file contains both printable and non printable characters that cover the entire ASCII range.
  - Examples: Most Unix commands, executable files, pictures, sound and video files are binary files.

2) Directory File

- A directory contains no data, but keeps details of the files and subdirectories that it contains.
- A directory file contains one entry for every file and subdirectory that it houses.
- Each entry has two components namely,
  1) filename and
  2) unique identification number of the file or directory (called the inode number).
- When you create or remove a file, the kernel automatically updates its corresponding directory by adding or removing the entry (filename and inode number) associated with the file.
- Unix directories are equivalent to windows folders.

3) Device File

- All the operations on the devices are performed by reading or writing the file representing the device.
- Advantage of device file is that some of the commands used to access an ordinary file also work with device file.
- Device filenames are generally found in a single directory structure, /dev.
- A device file is not really a stream of characters.
- It is the attributes of the file that entirely govern the operation of the device.
- The kernel identifies a de ice from its attributes and uses them to operate the device.

1 (c) Briefly describe: i) System calls ii) PATH iii) HOME (6 Marks)

Ans(i):

System Call

• A system call is just what its name implies -- a request for the operating system to do something on behalf of the user's program.
• The system calls are functions used in the kernel itself.
• To the programmer, the system call appears as a normal C function call.
• UNIX system calls are used to manage the file system, control processes, and to provide interprocess communication.
• Example: read(), open(), close(), fork(), exec(), exit()

Ans(ii):

PATH Variable

• This variable specifies the locations in which the shell should look for commands.
• Example:

```
$ echo $PATH
/bin: /usr/bin:
```

• When you specify a command like date, the system will locate the associated file from a list of directories specified in the PATH variable and then executes it.
• The PATH variable also includes the current directory.
• Whenever you enter any UNIX command, you are actually specifying the name of an executable file located somewhere on the system.
• The system goes through the following steps in order to determine which program to execute:
    1) Built in commands (such as cd and history) are executed within the shell.
    2) If an absolute path name (such as /bin/ls) or a relative path name (such as ./myprog), the system executes the program from the specified directory.
    3) Otherwise, the PATH variable is used.

Ans(iii):

HOME Variable

• When a user logs into the system, the user will be placed in a directory called home directory.
• Home directory is created by the system when the user account is created.
• The shell-variable HOME indicates the home directory of the current user.
• This variable is set for a user by the system admin in /etc/passwd.
• Example:

```
$ echo $HOME
/home/kumar
```

• Here, absolute pathname is shown.
• Absolute pathname is a sequence of directory names starting from root (/).
• The subsequent slashes are used to separate the directories.

2 (a) i) Explain the significance of all the fields of 'ls - l' output. ii) Which of the attributes can be changed only by the super user? (8 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.2a

Ans(ii):

Superuser

• A superuser has complete control of the system.
• The superuser can run any commands without any restriction.
• The superuser can
    → add user accounts to the system
    → modify user account attributes
    → add groups to the system
    → modify group attributes
    → change the file permissions of every user

2 (b) With a neat diagram, explain the three modes of vi editor. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.2c

2(c) Assuming that a file's current permission are rw_ r_x r__. Specify the chmod expression (using both relative and absolute methods) required to change them to

i) rwx rwx rwx
ii) r_ _ r_ _ _ _ _
iii) _ _ _ r_ _ r_ _ (6 Marks)

Ans:

Current permission: rw_ r_x r__
Assume filename: demofile

	Relative Permissions	Absolute Permissions
i) rwx rwx rwx	chmod u+x g+w o+wx demofile	chmod 777 demofile
ii) r__ r__ ___	chmod u-w g-x o-r demofile	chmod 440 demofile
iii) ___ ___ ___	chmod u-rw g-rx o-r demofile	chmod 000 demofile

3 (a) Devise wild-card patterns to match file names:
i) Comprising of at least three characters where the first char is numeric and the last character is not alphabetic.
ii) With three character extension except the one with .log extension.
iii) Containing 2004 as an embedded string except at the beginning or end. (6 Marks)

Ans:

i) [0-9]?[!A-Za-z]
ii) *.[!l][!o][!g]
iii) ?*2004*?

3 (b) Explain the 3 distinct phases of process creation. How is the shell created? (8 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.3b

3 (c) What are environment variables? Briefly describe any five of them. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.3c

4 (a) Distinguish between hard link and symbolic links with suitable examples. (8 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.4a

4 (b) Describe sort filter and illustrate its usage with -k, -u, -n, -r and -c options. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.4b

4 (c) i) Use find to locate all files named a.out and all C source files in your home directory tree and remove them interactively.
ii) Display only the names of all user who are logged in and also store that result in user.txt
iii) Invoke the vi editor with the last modified file. (6 Marks)

Ans:

i) find /home \(-name a.out -a -name ".C" \) -exec rm { } \
ii) who > user.txt
iii) vi `ls -tr | tail -1`

5 (a) i) Explain with suitable examples, the sed filter along with its two forms of addressing.
ii) Also describe in brief the substitution feature provided by sed. (8 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.5b

Ans(ii):

Substitution Command

• The substitution command(s) will substitute any string that you specify with any other string that you specify.

• Syntax:

[address]s/expression1/expression2/flags

• Example:

sed 's/root/kumar/' emp.lst

• The above command substitutes the first occurrence on a line of the string "root" with the string "kumar".

• sed substitutes only the first occurrence on a line.

• We need to use the g (global) flag to replace all occurrence.

sed 's/root/kumar/g' emp.lst

• We can limit the vertical boundaries too by specifying an address (for first three lines only).

sed '1,3s/|/:/g' emp.lst

• We can also use regular expressions for patterns to be substituted.

sed 's/[Aa]gg*[ar][ar]wal/Agarwal/g' emp.lst

• The above command replaces all occurrence of agarwal, aggarwal and agrawal with simply Agarwal.

Performing multiple substitutions

```
sed 's/<I>/<EM>/g
s/<B>/<STRONG>/g
s/<U>/<EM>/g' form.html
```

5 (b) Describe the grep filter along with any five options. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.5a

5 (c) i) Use sed to delete all blanks lines from a file named sample.
ii) Use grep to list only the sub-directories in the current dictionary.
iii) Replace all occurrences of the word "UNIX" with "LINUX" in a file named sample. (6 Marks)

Ans:

i) $ sed '/^$/d' sample.txt

ii) ls -l | grep ^d

iii) sed 's/UNIX/LINUX /' sample.txt

6 (a) Define a shell script. What are the two ways of running a shell script? Write a shell script to accept pattern and a file and search for the pattern in the file. (8 Marks)

Ans:

Shell Programming

- A shell script contains a list of commands which have to be executed regularly.
- Shell script is also known as shell program.
- The user can execute the shell script itself to execute commands in it.
- A shell script runs in interpretive mode. i.e. the entire script is compiled internally in memory and then executed.
- Hence, shell scripts run slower than the high-level language programs.
- ".sh" is used as an extension for shell scripts.
- Example: A shell script (program1.sh) to read a pattern and filename from the terminal. And search for the pattern in the file.

```
#! /bin/sh
echo "Enter the pattern to be searched: \c"
read pname
echo "Enter the file to be used: \c"
read fname
echo "Searching for pattern $pname from the file $fname"
grep $pname $fname
echo "Selected records shown above"
```

```
Output:
Enter the pattern to be searched : MH
Enter the file to be used: student.lst
Searching for pattern MH from the file student.lst
4       | MH   | 10    | IS    | 111
4       | MH   | 11    | CS    | 401
Selected records shown above
```

- There are 2 ways to execute a shell script:

1) Execute Shell Script Using File Name

- By default, script is not executable. So, the chmod command can be used to make the script executable.
- The scripts are executed in a separate child shell process.
- The child shell reads and executes each statement in interpretive mode.

```
Run:
$ chmod +x program1.sh                    // add executable permission
$ program1.sh                             // execute the script program1.sh
```

2) Execute Shell Script by Specifying the Interpreter

- The user can also execute a shell script by specifying the interpreter in the command line.
- Here, the script neither requires a executable permission nor an interpreter line.

```
Run:
$ sh program1.sh                          //Execute using sh interpreter
$ bash program1.sh                        //Execute using bash interpreter
```

6 (b) Explain the shell's for loop giving the possible sources of the list. (6 Marks)

Ans:

for Statement

- for loop can be used to iterate over all items(or strings) within a list.
- Syntax:

```
for variable in list
        do
                statements
        done
```

- Here, list consists of a set of items(or strings).
- Each item of the list is picked up and assigned to the "variable".
- The iteration continues until all items are picked from the array.
- Example: A script to display elements of an array.

```
#! /bin/sh
print("Here are the numbers in the list: \n");
for var in 10 20 30 40 50 60;
        do
                echo "$var  \t"
        done
```

```
Output:
Here are the numbers in the list
10 20 30 40 50 60
```

Possible Sources of List

- Possible sources of list are
  1) List from variables
  2) List from command substitution
  3) List from wildcards and
  4) List from positional parameters

1) List from Variables

- Example: A script to evaluate & display a set of variables using for-loop.

```
#! /bin/sh
x="Dream"
y="Believe "
z="Achieve"
for var in $x $y $z;
        do
                echo "$var  \t"
        done
```

```
Output:
Dream        Believe        Achieve
```

2) List from Command Substitution

- Command substitution can be used for creating a list.
- Useful: when list is large.
- Example: A script to display current date using for-loop.

```
#! /bin/sh
for var in `date`
        do
                echo "$var  \t"
        done
```

```
Output:
Mon    Nov    4      08:02:45      IST     2017
```

3) List from Wildcards

- The shell can use wildcards for matching filenames.
- Example: A script to print all files with pdf extension.

```
#! /bin/sh
for file in *.pdf
do
        echo "Printing $file \n"
        lp $file
done
```

```
Output:
Printing chap1.pdf
Printing chap2.pdf
Printing chap3.pdf
```

4) List from Positional Parameters

- Example: A script (program4.sh) to read & display a positional parameters using for-loop.

```
#! /bin/sh
for var in "$*"                                  # even "$@" can be used
        do
                echo "$var \t"
        done
```

```
Run:
$ program4.sh A B C
Output:
A       B       C
```

6 (c) Write a menu-driven shell script to perform the following:

i) List of user who are logged in.
ii) List of files in the current directory
iii) List of processes of user
iv) Today's date
v) Quit to UNIX (6 Marks)

Ans:

```
#! /bin/sh
echo " MENU  \n
        1. List of files        2. Processes of user  \n
        3. Today's Date         4. Users of system          5. Quit  \n
        Enter your option:  \c"
read choice
case "$choice" in
                1) ls –l;;
                2) ps –f ;;
                3) date ;;
                4) who ;;
                5) exit ;;
                *) echo "Invalid option"
esac
```

```
Output:
MENU
1. List of files        2. Processes of user
3. Today's Date         4. Users of system          5. Quit
Enter your option: 3
Mon Oct 8 08:02:45 IST 2007                    // date command executed
```

7 (a) i) Describe the awk filter with syntax and example. ii) How are awk arrays different from the ones used in most programming language? (8 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.7a.(i)

Ans(ii):

- They are not formally defined. An array is considered declared the moment it is used.
- Array elements are initialized to zero or an empty string unless initialized explicitly.
- Arrays expand automatically.
- The index can be virtually any thing: it can even be a string.

7 (b) Explain the looping constructs supported by awk. (6 Marks)

Ans:

For Loop

- Syntax

```
for (initialisation; condition; increment/decrement)
 action
```

- Initially, the for statement performs initialization action, then it checks the condition.
- If the condition is true, it executes actions, thereafter it performs increment or decrement operation.
- The loop execution continues as long as the condition is true.
- Example: The following example prints 1 to 5 using for loop

```
awk 'BEGIN { for (i = 1; i <= 5; ++i) print i }'          // prints 1 2 3 4 5
```

While Loop

- The **while** loop keeps executing the action until a particular logical condition evaluates to true.
- Syntax

```
while (condition)
        action
```

- AWK first checks the condition; if the condition is true, it executes the action.
- This process repeats as long as the loop condition evaluates to true.
- Example: The following example prints 1 to 5 using while loop

```
awk 'BEGIN {i = 1; while (i < 6) { print i; ++i } }'      // prints 1 2 3 4 5
```

7 (c) Briefly describe built-in-functions supported by awk for arithmetic & string operations. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.7a.(ii)

8 (a) With examples, explain the string handling functions supported by perl. (8 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.8a

8 (b) How are i) split and ii) join used in perl scripts? (6 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.8b.(i)

Ans(ii):

join()

- join() function
  - → combines its arguments into a single string and
  - → uses the delimiter as the first argument.
- Syntax:

  join (sep, list);

  where "sep" is a separator which can be any string.
- Here, "sep" will be placed between the individual elements of the "list".

  "list" can be
  - → an array name
  - → a list of variables or
  - → string to be joined.
- Example:

```
$result = join(" ", "this", "is" , "an" , "example");        // $result = this is an example
$week = join( "|" , "mon", "tue", "wed");                    // $week = mon | tue | wed
```

- Example: A perl script to illustrate both split and join functions.

```
#!/usr/bin/perl
$x=join( ":" , 10,20,30,40);
print "$x \n";
@y=split(/:/ , $x);
print "@y \n";
$x=join("-", @y);
print "$x \n";
```

Output:

```
10:20:30:40
10 20 30 40
10-20-30-40
```

8 (c) Write a perl script to determine whether a year is leap year or not. (6 Marks)

Ans:

A perl script (program3.pl) to determine whether the given year is leap year or not, which takes year as the command line argument.

```
#!/usr/bin/perl
#leap_year.pl
if (@ARGV == 0)
{
        die("you have not entered the year \n");
}
$year = $ARGV[0] ;
$lastdigit= substr($year, -2 , 2);
if ($lastdigit eq "00")
{
        $yesorno = ($year % 400 == 0 ? "certainly" : "not");
}
else
{
        $yesorno = ($year % 4 == 0 ? "certainly" : "not");
}
print("$year is"  $yesorno  "a leap year");
```

```
Run 1:
$ program3.pl
Output 1:
You have not entered the year

Run 2:
$ program3.pl 2004
Output 2:
2004 is certainly a leap year
```

USN | | | | | | | | | | |

10CS44

Fourth Semester B.E. Degree Examination, June/July 2014
UNIX and Shell Programming

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting atleast TWO questions from each part.***

PART – A

1 a. With a neat diagram, explain the architecture of UNIX operating system. List the features also. **(08 Marks)**

b. Explain the parent–child relationship of UNIX file system with a diagram. **(06 Marks)**

c. Explain with examples :
i) Absolute pathname and relative pathname
ii) Internal and external commands. **(06 Marks)**

2 a. Interpret the significance of seven fields of ℓs – ℓ output. **(07 Marks)**

b. Briefly explain the different ways of setting file permissions. **(07 Marks)**

c. With a diagram, explain 3 modes of Vi editor. **(06 Marks)**

3 a. What are wild cards? Explain the shells wild cards, with examples. **(08 Marks)**

b. What is a process? Explain the process creation mechanism? Why directory change can't be made in separate process. **(08 Marks)**

c. Explain the following environment variables, with examples :
i) HOME ii) PATH iii) IFS iv) SHELL. **(04 Marks)**

4 a. What are hard links and soft link? Explain with examples. **(06 Marks)**

b. Write a short note on find command. **(06 Marks)**

c. Explain the following filters with examples :
i) head iii) tail iii) cut. **(08 Marks)**

PART – B

5 a. Explain grep command with all options. **(10 Marks)**

b. What is sed? With example, explain line addressing and context addressing. **(10 Marks)**

6 a. What is shell programming? Write a shell script to create a menu which displays :
i) List of files ii) Contents of a file iii) Process status
iv) Current date v) Clear the screen vi) Current users of system. **(10 Marks)**

b. Explain shell features of 'for'. With syntax and examples. **(10 Marks)**

7 a. What is an awk? Explain all the built in variables used by awk. **(10 Marks)**

b. With syntax and examples, discuss the control flow statements used by awk. **(10 Marks)**

8 a. Write a Perl script to demonstrate the use of chop function. **(06 Marks)**

b. Write a Perl script to find the square root of command line arguments. **(06 Marks)**

c. Explain the string handling functions of Perl with appropriate examples. **(08 Marks)**

* * * * *

UNIX AND SHELL PROGRAMMING SOLVED PAPER JUNE -2014

1(a) i) With a diagram, explain architecture of UNIX operating system. ii) List the features also. (8 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.1a

Ans(ii): For answer, refer Solved Paper Dec-2013 Q.No.1a

1(b) Explain the parent-child relationship of UNIX file system with a diagram. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.1b

1(c) Explain with examples:

i) Absolute pathname and relative pathname

ii) Internal and external commands. (6 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.1c

Ans(ii):

Internal and External Commands

- Some commands are implemented as part of the shell itself rather than separate executable files. Such commands that are built-in are called internal commands.
- If the command (file) has an independence existence in the /bin directory, it is called external command.
- Examples:

```
$ type echo                    // echo is an internal command
echo is shell built-in
$ type ls                      // ls is an external command
ls is /bin/ls
```

- If the command exists both as an internal and external one, shell execute internal command only.
- Internal commands will have top priority compare to external command of same name.

2(a) Interpret the significance of seven fields of ls -l output. (7 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.2a

2(b) Briefly explain the different ways of setting file permissions. (7 Marks)

Ans:

Changing File Permissions

- A file is created with a default set of permission.
- chmod command can be used to change permission of a file.
- This command can be used in two ways: 1) Relative mode and 2) Absolute mode.

1) Relative Permissions

- This command can be used to add/delete permission for specific type of user (owner, group or others).
- This command can be used to
  - → change only those permissions specified in the command line and
  - → leave the other permissions unchanged.
- Syntax:
  - chmod category operation permission filename
- This command takes 4 arguments:
  - 1) category can be
    - u → user (owner)
    - g → group
    - o → others
    - a → all (ugo)
  - 2) operation can be
    - + → assign
    - - → remove
    - = → absolute
  - 3) permission can be
    - r → read
    - w → write
    - x → execute
  - 4) Filename whose permission has to changed
- This command can also accept multiple file names.
- Example:
  - $ chmod u+x note1 note2 note3

2) Absolute Permissions

- This command can be used to add/delete permission for all type of users (owner, group or others).
- This command can be used to change all permissions specified in the command line.
- Syntax:
  - chmod octal_value filename
- This command takes 2 arguments:
  - 1) octal_value contains 3 octal digits to represent 3 type of users (owner, group or others).
    - 1) First digit is for user
    - 2) Second digit is for group and
    - 3) Third digit is for others
  - Each digit represents a permission as shown below:
    - 4 (100) – read only
    - 2 (010) – write only
    - 1 (001) - execute only
    - 6 (110) – read & write only

For ex: octal_value of 644(110 100 100) means

- → user can read & write only
- → group can read only
- → others can read only

2) Filename whose permission has to changed.

• Example:

Relative Permissions	Absolute Permissions

$ ls -l xstart
-rw-r—r-- 1 kumar metal 1906 sep 23:38 xstart
$ chmod u+x xstart
$ ls -l xstart
-rwxr—r-- 1 kumar metal 1906 sep 23:38 xstart
$ chmod ugo+x xstart OR chmod a+x xstart
$ ls –l xstart
-rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart
$ chmod go-r xstart
$ls –l xstart
-rwx—x--x 1 kumar metal 1906 sep 23:38 xstart
	$ ls -l xstart
	-rw-r—r-- 1 kumar metal 1906 sep 23:38 xstart
	$ chmod 744 xstart
	$ ls -l xstart
	-rwxr—r-- 1 kumar metal 1906 sep 23:38 xstart
	$ chmod 755 xstart OR chmod a+x xstart
	$ ls –l xstart
	-rwxr-xr-x 1 kumar metal 1906 sep 23:38 xstart
	$ chmod 711 xstart
	$ls –l xstart
	-rwx—x--x 1 kumar metal 1906 sep 23:38 xstart

2(c) With a diagram, explain 3 modes of Vi editor. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.2c

3(a) What are wild cards? Explain the shells wild cards, with examples. (8 Marks)

Ans:

Wild Cards and Filename Generation

• The metacharacters that are used to construct the generalized pattern for matching filenames belong to a category called wild-cards.

Wild Card/ Character class	Match
*	Any number of characters including none
?	A single character
[ijk]	A single character either an i, j or k
[x-z]	A single character that is within the ASCII range of the characters x and z
[!ijk]	A single character that is not an i, j, or k
[!x – z]	A single character that is not within the ASCII range of the characters x and z

1) Metacharacters * and ?

• The metacharacter "*" matches any number of characters including none.

• Examples:

```
$ ls chap*      // To list all files that begin with "chap"
chap chap01 chap02 chap03 chap04 chapx chapy chapz
```

• When the shell encounters this command line, it identifies the * immediately as a wild-card. It then looks in the current directory and recreates the command line as below

```
$ ls chap chap01 chap02 chap03 chap04 chapx chapy chapz
```

• The shell now hands over this command to the kernel which uses its process creation facilities to run the command.

• The metacharacter ? matches a single character.

• Example:

```
$ ls chap?              // To list all five-characters filenames beginning with "chap"
chapx chapy chapz
$ ls chap??             // To list six-characters filenames beginning with "chap"
chap01 chap02 chap03
```

• Both * and ? operate with some restrictions.

• The * doesn’t match all filenames beginning with a dot (.) or forward slash(/).

• Example:

```
$ ls .C*                // to list all C extension filenames
$ cd /usr?local         // this doesn't match /usr/local
```

2) Character Class

- The character class comprises a set of characters enclosed by the rectangular brackets, [and],.
- The character class matches a single character in the class.
- For example:

Character Class	Match
[ijk]	A single character either an i, j or k
[x-z]	A single character that is within the ASCII range of the characters x and z.
chap0[124]	chap01, chap02, chap04
[!ijk]	A single character that is not an i, j, or k
[!x – z]	A single character that is not within the ASCII range of the characters x and z

- This can be combined with any string or another wild-card expression.
- Example:

```
$ ls chap0[124]      //Matches chap01, chap02, chap04 and lists if found.
$ ls chap[x-z]       //Matches chapx, chapy, chapz and lists if found.
```

Negating the Character Class (!)

- Not operator (!) can be used to negate the character class.
- For example,

$ls [!a-zA-Z]* //To match all filenames that don't begin with an alphabetic character.

Matching Totally Dissimilar Patterns

```
$ cp $HOME/prog_sources/*.{c,java} .  // To copy all the C and Java source programs from
                                      //another directory to the current directory
$ cp /home/srm/{project,html,scripts/* .    // To copy all files from 3 directories (project,
                                    // html and scripts) to the current directory.
```

3(b) What is a process? Explain the process creation mechanism? Why directory changed can't be made in separate process. (8 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.3b

3(c) Explain the following environment variables with examples:
i) HOME ii) PATH iii) IFS iv) SHELL (4 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.3c

4(a) What are hard links and soft links? Explain with examples. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.4a

4(b) Write a short notes on find command. (6 Marks)

Ans:

find

- This command
  - → recursively examines a directory tree to look for files matching some criteria and
  - → then takes some action on the selected files.
- Syntax:

  find path_list selecton_criteria action
- Here,

  path_list consists of one or more directories to be examined
  selection-criteria contains conditions used for matching a file
  action specifies some action to be taken on the selected files
- Example:

  $find / -name a.out -print // locates all files named a.out
  /home/kumar/a.out
  /home/user/a.out
- Here,
  1) path_list **/** indicates that the search should start from root directory.
  2) Then, each file in the list is matched against the selection criteria.
     The selection criteria always consists of an expression in the form –operator argument.
     If the expression matches the file, the file is selected.
  3) Finally, a display selected files on the terminal.
- The group of filenames with a wild-card pattern can be matched.

  Here, the pattern should be quoted to prevent the shell from looking at it:

  find . -name "*.c" –print // All files with extension .c
  find . -name '[A-Z]*' –print // All files begin with an upper case letter

1) Selection Criteria

Locating a File by inode Number (-inum)

- -inum option can be used to search the files based on inode number.
- Example:

```
find / -inum 13975 -print 2> /dev/null
/usr/bin/gzip
/usr/bin/gunzip
```

File Type and Permission (-type and -perm)

- –type option can be used to search the files based its type.

  f indicates ordinary file
  d indicates directory file
  l indicates symbolic file
- Example:

```
find /TC/BIN -type d -print 2>/dev/null     //shows current and hidden directories
./c_programs
./cpp_programs
```

- –perm option can be used to indicate the permission of file to be matched.
- Example:

  find /TC/BIN -perm 666 -type f -print 2> /dev/null
- Here, –perm 666 selects files having read and write permissions for all categories of users.

Finding Unused File (-mtime and -atime)

- The -mtime and -atime option can be used to search the files based on the modified and accessed time respectively.
- Example:

```
find . -mtime -2 -print                 //list all that have been modified since less last 2 days
find /home -atime +365 -print           // files that are not been accesses for more than a year
```

2) find Operators (!, -o , -a)

• There are 3 operators can be used with the find.

1) ! operator is used before an option to negate its meaning.

• Example:

```
find . ! -name "*.c" –print          // Finds all files excluding .c file.
```

2) -o operator represents an OR condition.

• Example:

```
find /home \( -name "*.sh " -o -name "*.pl" \) -print    //searches files with .sh or .pl extn
```

3) –a operator represents the AND condition.

• Example:

```
find $HOME –perm 777 –a –type –d -print  //all directories providing all access rights to all
```

3) Options Available in the Action Component

Display (-print)

• -print option can be used to display pathnames of matching files on the standard output.

Displaying the listing (-ls)

• -ls option can be used to view the listing for the selected files.

• Example:

```
find . -type f -mtime +2 -mtime -5 -ls
4521 1 –rw-r—r--  11user user       234   Mar 23 11:12 ./c_pgms/prime.c
```

• Here, above command display a special listing of those regular files that are modified in more than two days and less than five days.

Taking Action on Selected Files (-exec and -ok)

• The –exec option can be used to a specific command.

• This takes the command to execute as its argument, followed by {} and finally /;

• Example:

```
find $HOME -type f -atime +365 -exec rm {} \; //Uses rm command to remove all
                                              //ordinary files unaccessed for more than a year.
```

4(c) Explain the following filters with examples:
i) head ii) tail iii) cut (8 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.4c.(i)

Ans(ii):

tail

• This command is used to display the end of the file.

• By default, it displays the last 10 lines of the file.

• Syntax:

```
tail [+- count] filename[s]
```

• Here, count can be one of the two:

1) -count option specifies a line count i.e. number of lines to be selected.

2) +count option specifies starting line number i.e. The line number from where the selection should begin.

• Example:

```
$ tail student.lst                  // Displays last 10 lines of student.lst

$ tail -n 3 student.lst             // Displays the last 3 lines of student.lst
1       | AM   | 10   | IS   | 021
1       | RN   | 10   | IS   | 054
1       | DS   | 11   | CS   | 406

$ tail +9 student.lst               // Displays all lines, beginning from line# 9 to the end of the file.
1       | RN   | 10   | IS   | 054
1       | DS   | 11   | CS   | 406
```

Ans(iii):

cut

- This command can be used to extract the file contents vertically.
- This command can be used to extract required fields or columns from a file.

cut Options

Cutting columns (-c)

- -c option can be used to extract specific column by specifying the column numbers.
- Range of column numbers can be specified using the hyphen(-).
- Comma (,) can be used as the column delimiter.
- Example:

```
$ cat student.lst
4       | MH   | 10    | IS    | 111
4       | MH   | 11    | CS    | 401
4       | GW   | 11    | CS    | 402
4       | VV   | 11    | CS    | 403

$ cut -c 5-6, 10-11 student.lst
MH      10
MH      11
GW      11
VV      11
```

- Example:

cut -c -3, 5-6, 15- student.lst // 15- means column 15 to end of line, -3 is same as 1-3.

Cutting Fields (-f)

- -f option can be used to extract specific field from a file.
- The tab is used as the default field delimiter.
- However, different delimiter can be specified with following 2 options:
  1) -d for the field delimiter &
  2) -f for the field list.
- Example 3:

```
$ cut -d \| -f 2,5 student.lst        OR     cut –d "|" -f 2,5 student.lst
MH      111
MH      401
GW      402
VV      403
```

- The above command extracts the second and fifth fields of student.lst.
- Delimiter "|" has to be escaped (by \ or " ") to prevent the shell from interpreting as the pipeline character.

5(a) Explain grep command with all options. (10 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.5a

5(b) What is sed? With example, explain line addressing and context addressing. (10 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.5b

6(a) What is shell programming? Write a shell script to create a menu which displays:
i) List of files
ii) Contents of a file
iii) process status
iv) Current date
v) Clear the screen
vi) Current users of system. (10 Marks)

Ans:

Shell Programming

- A shell script contains a list of commands which have to be executed regularly.
- Shell script is also known as shell program.
- The user can execute the shell script itself to execute commands in it.
- A shell script runs in interpretive mode. i.e. the entire script is compiled internally in memory and then executed.
- Hence, shell scripts run slower than the high-level language programs.

- Example: A shell script to create a menu using case statement.

```
#! /bin/sh
echo " MENU  \n
        1. List of files          2. Processes of user  \n
        3. Today's Date           4. Users of system    \n
        5. Clear the screen       6. Contents of a file \n
        7. Quit  \n
        Enter your option:  \c"
read choice
case "$choice" in
                1) ls –l;;
                2) ps –f ;;
                3) date ;;
                4) who ;;
                5) clear ;;
                6) echo "enter filename:\c"
                   read filename
                   cat $filename ;;
                7) exit ;;
                *) echo "Invalid option"
esac
```

```
Output:
MENU
1. List of files          2. Processes of user
3. Today's Date           4. Users of system
5. Quit
Enter your option: 3
Mon Oct 8 08:02:45 IST 2007                          // date command executed
```

6(b) Explain the shell features of 'for'. With syntax and examples. (10 Marks)

Ans : For answer, refer Solved Paper Dec-2013 Q.No.6b

7(a) i) What is an AWK? ii) Explain all the built in variables used by awk. (10 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.7a.(i)

Ans(ii): For answer, refer Solved Paper June-2013 Q.No.7c

7(b) With syntax & examples, discuss the control flow statements used by awk. (10 Marks)

Ans:

Control Flow Statements

• Like other programming languages, awk provides conditional statements to control the flow of a program.

• The if statement can be used when the && and || are found to be inadequate for certain tasks.

If statement

• It simply tests the condition and performs certain actions depending upon the condition.

• Syntax

```
if (condition)
    action
```

• We can also use a pair of curly braces as given below to execute multiple actions –

• Syntax

```
if (condition) {
    action-1
    action-1
        .
        .
    action-n
        }
```

• Example: The following example checks whether a number is even or not –

```
awk 'BEGIN {num = 10; if (num % 2 == 0) printf "%d is even number.\n", num }'
```

• On executing the above code, you get the following result –

```
10 is even number.
```

• if can be used with the comparison operators and the special symbols ~ and !~ to match a regular expression.

• When used in combination with the logical operators || and &&, awk programming becomes quite easy and powerful.

• The conditional operator is also called ternary operator it takes three operands.

• You can even replace the if construct with a compact ternary operator.

• Syntax

```
(exp1)? exp2: exp3;
    where exp1 is an expression evaluated to true or false;
        If exp1 is evaluated to true, exp2 is executed;
        If exp1 is evaluated to false, exp3 is executed.
```

8(a) Write a Perl script to demonstrate the use of chop function.(6 Marks)

Ans : For answer, refer Solved Paper June-2013 Q.No.8b.(ii)

8(b) Write a perl script to find the square root of command line arguments. (6 Marks)

Ans:

A perl script (program2.pl) to find the square root of command line arguments

```
#!/usr/bin/perl
foreach $num (@ARGV)
{
    print "The square root of $num is" sqrt($num);
}
```

```
Run:
$ program2.pl 25 36
Output:
The square root of 25 is 5
The square root of 36 is 6
```

8(c) Explain the string handling functions of Perl with appropriate examples. (8 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.8a

USN | | | | | | | | | | |

10CS44

Fourth Semester B.E. Degree Examination, Dec.2014/Jan. 2015
UNIX and Shell Programming

Time: 3 hrs. Max. Marks:100

Note: ***Answer FIVE full questions, selecting atleast TWO questions from each part.***

PART – A

1 a. Describe briefly the UNIX architecture explaining the role played by the kernel and shell in sharing the work load. **(08 Marks)**

b. Draw the tree structure of the file system created by the following commands (assume you are in the directory/usr/office). Why is it not possible to issue the command rmdir/usr/office/right.

```
$ mkdir left
$ mkdir middle
$ mkdir right
$ cd left
$ mkdir left middle right
$ cd ••/middle
$ mkdir dir1 dir2/usr/office/right/dir3.
```

(08 Marks)

c. Explain the concept of absolute path name and relative pathname. **(04 Marks)**

2 a. Which command is used for listing file attributes? Explain briefly the significance of each field of the output. **(06 Marks)**

b. Assuming that a file's current permissions are r w x r – – r – x, specify the chmod expression required to change them to :

i) r w x r w x r – x
ii) r – xr – x – – x
iii) – – – r – – r – x
iv) – – – r w – r – –,

using both relative and absolute methods of assigning permissions. **(08 Marks)**

c. Explain the three modes of vi and explain how you can switch from one mode to another. **(06 Marks)**

3 a. Explain the three sources of standard input and standard output. **(06 Marks)**

b. Explain what these wild – card patterns match :

i) [A – Z]????*
ii) *[0 – 9]*
iii) *[!0 – 9]
iv) *·[!s] [!h]. **(08 Marks)**

c. What is a process? Mention briefly the role of fork – exec mechanism in process creation. **(06 Marks)**

4 a. What are hard-links? Explain two application areas of hard-links. What are the two main disadvantages of the hard-link? **(06 Marks)**

b. Explain these commands with examples : i) umask ii) touch . **(06 Marks)**

c. Explain the following commands :

i) pr ii) tail iii) sort iv) tr. **(08 Marks)**

PART – B

5 a. Explain the grep command with options. **(08 Marks)**
b. What is sed? Explain addressing in sed, with suitable examples. **(08 Marks)**
c. Explain the anchoring characters. **(04 Marks)**

6 a. Explain the special parameters used by the shell. **(06 Marks)**
b. What is shell script? Explain the following statements with syntax and examples :
i) if ii) case iii)while. **(10 Marks)**
c. What is the exit status of a command and where is it stored? **(04 Marks)**

7 a. Explain awk's build-in variables. **(06 Marks)**
b. Write a program in awk to store the totals of the basic pay, da, hra and gross pay of the sales and marketing people. **(06 Marks)**
c. Briefly describe the built-in functions in awk, with examples. **(08 Marks)**

8 a. Write a Perl script to determine whether the given year is a leap year or not. **(07 Marks)**
b. Write a Perl script to convert decimal number to binary. **(07 Marks)**
c. Explain variables and operators in Perl. **(06 Marks)**

* * * * *

UNIX AND SHELL PROGRAMMING SOLVED PAPER DEC -2014

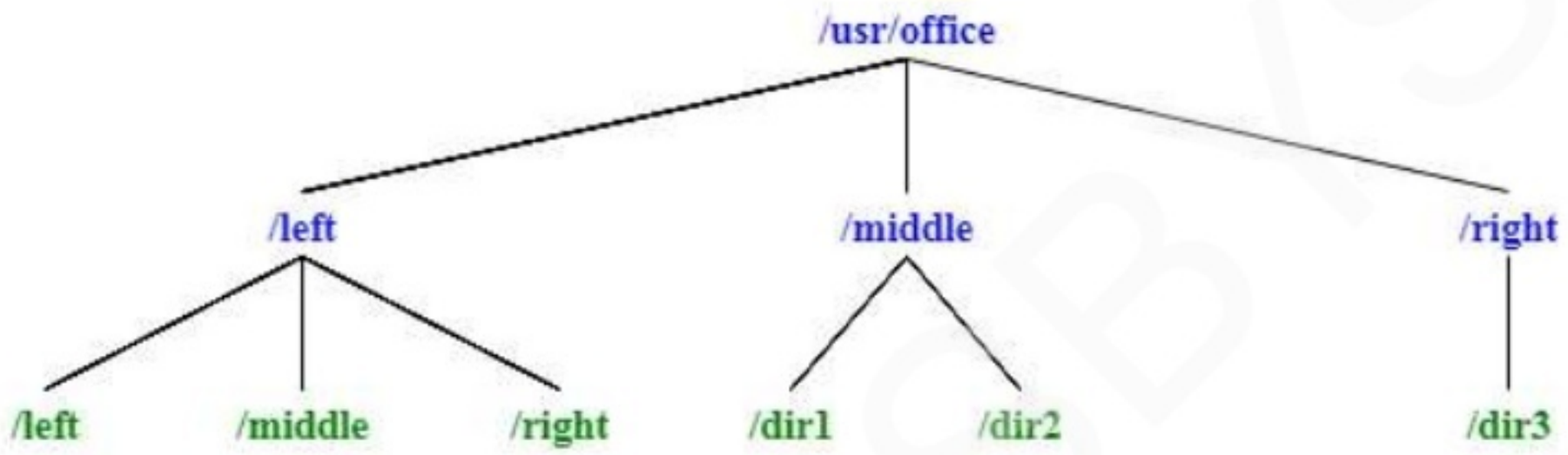
1 (a) Define briefly the UNIX architecture explaining the role played by the kernel and shell in sharing the work load. (8 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.1a

1 (b) Draw the tree structure of the file system created by the following commands (assume you are in the directory /usr/office). Why is it not possible to issue the command rmdir /usr/office/right.

```
$ mkdir left
$ mkdir middle
$ mkdir right
$ cd left
$ mkdir left middle right
$ cd ../middle
$ mkdir dir1  dir2  /usr/office/right/dir3  (8 Marks)
```

Ans:



- Currently, we are in location /usr/office/middle.
- To remove /usr/office/right, we must be in it's parent directory i.e. /usr/office.
- Therefore, it is not possible to issue the command rmdir /usr/office/right.

1 (c) Explain the concept of absolute path name and relative pathname. (4 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.1c

2 (a) Which command is used for listing file attributed? Explain briefly the significance of each field of the output. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.2a

2 (b) Assuming that a file's current permission are rwx r__ r_x, specify the chmod expression required to change them to:

i) rwx rwx r_x
ii) r_x r_x _ _ x
iii) _ _ _ r_ _ r_ x
iv) _ _ _ rw_ r_ _

using both relative and absolute methods of assigning permissions. (8 Marks)

Ans:

Current permission: rwx r__ r_x
Assume filename: demofile

	Relative Permissions	**Absolute Permissions**
i) rwx rwx r_x	chmod g+wx demofile	chmod 775 demofile
ii) r_x r_x _ _ x	chmod u-w g+x o-r demofile	chmod 551 demofile
iii) _ _ _ r_ _ r_ x	chmod u-rwx demofile	chmod 045 demofile
iv) _ _ _ rw_ r_ _	chmod u-rwx g+w o-x demofile	chmod 064 demofile

2 (c) Explain the three modes of vi and explain how you can switch from one mode to another. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.2c

3 (a) Explain the three sources of standard input and standard output. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.3a

3 (b) Explain what these wild-card pattern match:

i) [A-Z]????*
ii) *[0-9]*
iii) *[!0-9]*
iv) *.[!s] [!h] (8 Marks)

Ans:

i) [A-Z]????*

• Matches all files whose names start with an upper case letter and are followed by a 4 characters and are followed by any number of characters.

ii) *[0-9]*

• Matches all files whose names start with any character and are followed by a single digit and are followed by any character.

iii) *[!0-9]*

• Matches all files whose names start with any character and are followed by a single letter and are followed by any character.

iv) *.[!s] [!h]

• Matches all files with a single character extension but not .s or .h files.

3 (c) What is a process? Mention briefly the role of fork-exec mechanism in process creation (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.3b

4 (a) What are hard links? Explain two application areas of hard-links. What are the two main disadvantages of the hard-link? (6 Marks)

Ans:

ln: Creating Hard Links

- This command can be used to create a hard link between 2 or more files.
- Syntax:

  ln filename1 filename2
- Example:

```
ln emp.lst employee                    // This command links emp.lst with employee
ls -li emp.lst employee                // To check if 2 files have the same inode number
518 -rwxr-xr-x 2 kumar group 915 may 4 09:58 emp.lst
518 -rwxr-xr-x 2 kumar group 915 may 4 09:58 employee
```

- In the third column, 2 is the link count of 2 files (emp.lst and employee).
- Example:

```
ln employee emp.dat                    // This command links employee with emp.dat
ls -li emp*                            // To check if 3 files have the same inode number
518 -rwxr-xr-x 3 kumar group 915 may 4 09:58 emp.dat
518 -rwxr-xr-x 3 kumar group 915 may 4 09:58 emp.lst
518 -rwxr-xr-x 3 kumar group 915 may 4 09:58 employee
```

- In the third column, 3 is the link count of 3 files (emp.lst employee and emp.dat).
- Multiple files can be linked together. Here, the destination filename must be a directory.
- Example:

```
ln chap1 chap2 chap3 content           // "content" is a directory
```

- A file is considered to be completely removed from the file system when its link count drops to 0.

Where to use Hard Links? (Applications of Hard Link)

1) ln data/ foo.txt input_files

- The above command creates link in directory input_files.
- With this link available, your existing programs will continue to find foo.txt in the input_files directory.
- It is more convenient to do this that modifies all programs to point to the new path.

2) Links provide some protection against accidental deletion, especially when they exist in different directories.

3) Because of links, the user need not maintain 2 programs as 2 separate disk files if there is very little difference between them.

- A file's name is available to a C program and to a shell script. (emp.c emp.sh).
- A single file with 2 links can behave in 2 different ways based on the name by which it is called.

Disadvantage of Hard Links

1) Directories cannot be linked.

2) Files across 2 different file system cannot be linked.

For example: Cannot link a file in the /usr file system to another file in the /home file system.

4 (b) Explain these commands with examples:
i) unmask ii) touch (6 Marks)

Ans(i):

umask and Default File Permissions

- When user creates a file/directory, the permissions assigned to the file/directory depends on
  → system's default setting and
  → value of the variable umask.
- umask command can be used to set & display the user mask of file/directory.
- The value of user mask variable informs the system which permissions are to be denied, rather than granted.
- umask command can be used to change the permissions to required values during file/directory creation itself.
- The default permissions:
  1) rw-rw-rw- (octal 666) for regular file
  2) rwxrwxrwx (octal 777) for directory
- The default is transformed by subtracting the user mask from it to remove one or more permissions.
- Syntax:
  umask [octal_value]
- Example:
  $ umask // displays the current user mask setting
  022
- Example:
  $ umask 042 // sets new umask value
- The default is transformed by subtracting the user mask from it to remove one or more permissions.
- Example:
  For regular file:
  666 Default system value
  -022 umask value
  644 Default user value → 110 100 100 → rw-r—r--
- When user mask is set, it is temporary only for that session. When logout the settings, revert back to the default.

Ans(ii):

touch

- This command can be used to update the access and modification times of a file to the current time.
- Syntax:
  touch filename
- There are 3 cases:

1) touch command without options and expression.

- Here, both times are set to the current time and creates the file, if it doesn't exist.

2) touch command without options but with expression.

- Here, the expression consists of MMDDhhmm (month, day, hour and minute).
- Example:

```
touch 03161430 emp.lst
ls -l emp.lst
-rw-r--r-- 1    kumar metal 870 mar 16 14:30 emp.lst
ls -lu emp lst
-rw-r--r-- 1    kumar metal 870 mar 16 14:30 emp.lst
```

3) touch command with options and expression.

- The –m and –a options change the modification and access times, respectively
- Example:

```
touch -m 02281030 emp.lst
ls -l emp.lst
-rw-r--r-- 1 kumar metal 870 feb 28 10:30 emp.lst
touch -a 01261650 emp.lst
ls -lu emp.lst
-rw-r--r-- 1 kumar metal 870 jan 26 16:50 emp.lst
```

4 (c) Explain the following commands:
i) pr ii) tail iii) sort iv) tr (8 Marks)

Ans(i):

pr

• This command can be used to add suitable headers, footers and formatted text.
• This command adds five lines of margin at the top and bottom.
• The header shows the date and time of last modification of the file along with the filename and page number.
• Example:

```
pr dept.lst
May 06 10:38 1997 dept.lst page 1

01:accounts:6213
02:progs:5423
03:marketing:6521
04:personnel:2365
05:production:9876
06:sales:1006

...blank lines...
```

pr Options

• The different options for pr command are:
-k prints k (integer) columns
-t to suppress the header and footer
-h to have a header of user's choice
-d double spaces input
-n will number each line and helps in debugging
-on offsets the lines by n spaces and increases left margin of page

Ans(ii): For answer, refer Solved Paper June-2014 Q.No.4c.(ii)
Ans(iii): For answer, refer Solved Paper June-2013 Q.No.4b
Ans(iv): For answer, refer Solved Paper June-2013 Q.No.4c.(ii)

5 (a) Explain the grep command with options. (8 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.5a

5 (b) What is sed? Explain addressing in sed, with suitable examples. (8 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.5b

5 (c) Explain the anchoring characters. (4 Marks)
Ans:

Anchoring Characters

• The caret '^' and the dollar sign '$' are meta-characters that respectively match the empty string at the beginning and end of a line.
• They are termed anchors, since they force the match to be "anchored" to beginning or end of a line, respectively.
• Example:

```
grep "^2" emp.lst
```

• The above command selects lines starting with 2 in the file emp.lst.

```
grep "7...$" emp.lst
```

• The above command selects lines ending with ranges between 7000 to 7999 in the file emp.lst.

6 (a) Explain the special parameters used by the shell. (6 Marks)

Ans:

Command Line Arguments

- Shell scripts can accept arguments from the command line.
- The arguments are assigned to special shell variables called shell parameters.
- The shell parameters are reserved for specific functions.
- Different shell parameters:

1) $#: Stores the number of command-line arguments.
2) $0, $1, $2, $3: These are called positional parameters which represent command line arguments.
   - $0: Stores the filename of the current script.
   - $1: Stores the first argument.
   - $2: Stores the second argument
   - $3: Stores the third argument
3) $*: Stores all the arguments entered on the command line ($1 $2 ...).
4) "$@": Stores all arguments entered on the command line, individually quoted ("$1" "$2")
5) $?: Stores the exit status of the last command that was executed.
6) $$: Stores Pid of the current shell.
7) $!: Stores PID of the last background job.

- Example: A shell script (program2.sh) to read and display various shell parameters from the command line.

```
#!/bin/sh
echo "Total Number of Parameters : $#"
echo "File Name: $0"
echo "First Parameter : $1"
echo "Second Parameter : $2"
echo "Quoted Values: $*"
echo "Quoted Values: $@"
$echo "Exit value: $?"
echo "PID of current shell: $$"
```

```
Run:
$ program2.sh "RAJA RAM" "MOHAN ROY"
Output:
Total Number of Parameters : 2
File Name : program2.sh
First Parameter : RAJA RAM
Second Parameter : MOHAN ROY
Quoted Values: RAJA RAM MOHAN ROY     // stored as "RAJA RAM MOHAN ROY"
Quoted Values: RAJA RAM MOHAN ROY     // stored as array= {"RAJA RAM" , "MOHAN ROY"}
Exit value: 0                         // zero implies success
PID of current shell: 12345
```

6 (b) What is shell script? Explain the following statements with syntax and examples:
i) while ii) if iii) case (10 Marks)

Ans (i): For answer, refer Solved Paper June-2013 Q.No.6b.(i)

Ans (ii):

if Statement

- if statement is basically a "two-way" decision statement.
- This is used when we must choose between two alternatives.
- Three forms of if…else statement:
  1) if...fi statement
  2) if...else...fi statement
  3) if...elif...else...fi statement
- Syntax 1:

```
if command is successful
then
        execute statements
fi
```

- Syntax 2:

```
if command is successful
then
        execute statements
else
        execute statements
fi
```

- Syntax 3:

```
if command is successful
then
        execute statements
elif command is successful
  then
        execute statements
  else
        execute statements
fi
```

- Here is how it works:
  1) If the command succeeds, the statements within then-block are executed.
  2) If the command fails, the statements within else-block are executed.
- Example: A script to check whether an integer is positive or negative.

```
#! /bin/sh
echo "Enter any non zero integer: \n"
read num
if [$num -gt 0]; then
        echo "Number is positive number"
else
        echo "Number is negative number"
```

```
Output:
Enter any non zero integer:
5
Number is positive number
```

Ans (iii):

case Statement

- case statement is basically a "multi-way" decision statement.
- This is used when we must choose among many alternatives.
- This also handles string tests, but in a more efficient manner than if statement.
- Syntax:

```
case expression in
        pattern1) statement1 ;;
        pattern2) statement2 ;;

        pattern3) statement3 ;;
        …
esac
```

- Here is how it works:
  1) Firstly the expression is matched with pattern1.
  2) If the match succeeds, then statement1 will be executed.
  3) If the match fails, then the expression is matched with pattern2 and this process continues.
- Each statement is terminated with a pair of semicolon (;;).
- This can match only strings but cannot handle numeric and file tests.
  However, this can also handle numbers but treating them as strings.
- This is very effective when the string is fetched by command substitution.
- Example: A script to display appropriate message based on grades (A to D).

```
#! /bin/sh
echo "enter grade A to D \n"
read grade
case "$grade" in
        A) echo "Excellent!" ;;
        B) echo "Well done" ;;
        C) echo "You passed" ;;
        D) echo "Better try again" ;;
        *) echo "Invalid grade" ;;
esac
echo "Your grade is $grade"
```

```
Output:
enter grade A to D
B
Well done
Your grade is B
```

6 (c) What is the 'exit' status of command and where is it stores? (4 Marks)

Ans:

exit and Exit Status of a Command

- exit command can be used to terminate a program(or script).
- This command returns value which will be available to the script's parent process.
- The $? variable contains exit status of the last command executed.
- Exit status is a numerical value returned by every command upon its completion.
- A command returns an exit status of
  1) zero (0) upon successful execution and
  2) non-zero upon unsuccessful execution i.e. an error condition.
- Exit status can be used to devise program-logic that branches into different paths depending on success or failure of a command.
- Example: A shell script to find relationship between 2 numbers.

```
#! /bin/usr
x=5; y=7
test $x –eq  $y; echo "5=7: $? \n"
test $x –ne  $y; echo "5!=7: $? \n"
test $x –gt  $y; echo "5>7: $? \n "
test $x –ge  $y; echo "5>=7: $? \n "
test $x –lt   $y; echo "5<7: $? \n "
test $x –le  $y; echo "5<=7: $? \n"
```

```
Output:
5=7: 1                  // Returns nonzero exit status i.e. failure → False
5!=7: 0                 // Returns zero exit status i.e.  success → True
5>7: 1                  // False
5>=7: 1                 // False
5<7: 0                  // True
5<=7: 0                 // True
```

7 (a) Explain awk's build-in variables. (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.7c

7 (b) Write a program in awk to store the totals of the basic pay, da, hra and gross pay of the sales and marketing people. (6 Marks)

Ans:

```
BEGIN{
        printf "enter the basic salary: Rs"
        getline bp<"/dev/tty"
        if(bp<10000)
        {
                hra=.15*bp
                da=.45*bp
        }
        else
        {
                hra=.2*bp
                da=.5*bp
        }
        gs=bp+hra+da
        printf "gross salary=Rs.%.2f\n",gs
        }
```

7 (c) Briefly describe the built-in functions in awk, with examples (8 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.7a.(ii)

8 (a) Write a Perl script to determine whether the given year is leap year or not. (7 Marks)

Ans: **For answer, refer Solved Paper Dec-2013 Q.No.8c**

8 (b) Write a Perl script to convert decimal number to binary. (7 Marks)

Ans:

A perl script(program4.pl) to convert a given decimal number to binary equivalent.

```
#!/usr/bin/perl
foreach $num (@ARGV)
{
        $temp = $num;
        until ($num == 0)
        {
                $bit = $num % 2;
                unshift(@bit_arr, $bit);
                $num = int($num/2);
        }
$binary_num = join(" ",@bit_arr);
print ("Binary form of $temp is $binary_num\n");
}
```

```
Run:
$ program4.pl 4 7 65
Output:
Binary form of 4 is 100
Binary form of 7 is 111
Binary form of 65 is 1000001
```

8 (c) Explain variables and operations in Perl. (6 Marks)

Ans:

1) Variables

- The variables have no data type.
- The variables need not be initialization.
- The variables have to be prefixed with $ symbols (& they are also called scalars).
- $ symbol is used for both
  1) variable definition ex: $x=12
  2) variable evaluation ex: $z=$x+1
- Example:

```
$var=10;
print $var;                                    //outputs 10
```

- Features of variables:

1) If a string is used for numeric computation or comparison, the string will be converted to a number.
2) An undefined variable is assumed to be a null string and a null string is numerically 0.

- Example:

```
perl –e '$x++; print ("$x\n");'                // returns 1 i.e. 0+1.
```

3) If the first character of a string is not number, the entire string becomes numerically equivalent to zero.
4) If a string is present in the middle of an expression, the string will be converted to an integer.

- Example:

"12k34" is converted to the integer 12, not 12k34.

5) To compare 2 strings, the ASCII value of each character starting from the left will be matched.

- Example:

x is greater than yy1234 because x is greater than y in the ASCII collating sequence.

6) For numeric comparison, operators like ==,!=,!=,>,<,>= and <= are used.
For string comparison, operators like eq, ne, gt, lt, ge and le are used.
7) A string can be unquoted or single/double quoted.
8) A ternary/conditional operator (?:) can be used.

- Example:

```
$max=($x>$y)? $x : $y;                         // Sets max to x if x>y; otherwise sets max to y.
```

9) Every comparison/expression returns a value, and can set this value to a variable.

- Example:

```
$max=($x>$y)                                   // Sets max to 1 if x>y; otherwise sets max to 0.
```

10) Some more examples:

```
$x =$y =$z =5;                                 // Multiple assignment
$name ="rama \t\t krishna \n";                 // Two tabs and newline
$y ='A'; $y++;                                 // y becomes B
$z ="PO1"; $z++;                               // z becomes PO2
$todays_date = 'date'                          // Uses Command substitution
$name = 'rama"
$result ="\U$name\E";                          // $result is RAMA
$result ="\u$name\E"                           // $result is Rama
```

2) Concatenation Operators . and Repetition Operator x

- Following 2 operators can be used to operate on strings:
  1) The . operator, which joins two strings together.
  2) The x operator, which repeats a string;.
- The . operator joins the second operand to the first operand.
- Example:

```
$a = "rama" . "krishna";                       // $a is now ""ramakrishna"
$x ="perl"; $y=".com";   $z=$x . $y;           //  $z is now "perl.com"
```

- This join operation is also known as string concatenation.
- The x operator (the letter x) makes n copies of a string, where n is the value of the right operand:
- Example:

```
$a="R"x3;                                      //  $a is now RRR
$x="@"x5;                                      //  $x is now @@@@@
```

USN | | | | | | | | | | |

10CS44

Fourth Semester B.E. Degree Examination, June/July 2015
UNIX and Shell Programming

Time: 3 hrs. Max. Marks:100

Note: ***Answer any FIVE full questions, selecting atleast TWO questions from each part.***

PART – A

1 a. Explain the architecture of UNIX operating system with a neat diagram. **(08 Marks)**

b. Illustrate with a diagram, the typical UNIX file system and explain different types of files supported in UNIX. **(08 Marks)**

c. Explain internal and external commands with example. **(04 Marks)**

2 a. Which command is used for listing file attributes? Briefly describe the significance of each field of the output. **(08 Marks)**

b. A file's current permissions are rw – r – x r – – x. Specify the chmod expression required to change them for the following :

i) r w x r w x r w x

ii) r – – r – – – – –

iii) – – – – – – – – –

Using both the relative and absolute methods of assigning permissions. **(06 Marks)**

c. What are the different modes of vi editor? Explain with a diagram. **(06 Marks)**

3 a. Explain the three standard files with respect to UNIX operating system. **(06 Marks)**

b. Explain the mechanism of process creation using system calls in UNIX. **(06 Marks)**

c. Explain the following environment variables with examples :

i) SHELL

ii) LOGNAME

iii) PATH

iv) [illegible]

(08 Marks)

4 a. Distinguish between hard links and soft links with suitable examples. **(06 Marks)**

b. Explain the following filters with options :

i) pr

ii) sort.

(08 Marks)

c. Use find command to locate from your home directory :

i) All files with the extension ·html

ii) All flies having inode number 9076

iii) All directories having permissions 666

iv) All files not accessed for more than a year

v) All but the C program files

vi) All files named a·out and all "C" source files and remove them interactively. **(06 Marks)**

PART – B

5 a. Explain grep command with all options. **(08 Marks)**

b. Briefly explain the different ways of addressing used in sed with example. **(06 Marks)**

c. Explain BRE (Basic Regular Expression) character subset used for constructing regular expressions. **(04 Marks)**

d. Write the commands for the following :

i) Use sed to delete all blank lines from a file named sample

ii) Use sed to replace all occurrences of the word "UNIX" with "LINUX" in a file named sample. **(02 Marks)**

6 a. What is shell programming? Write a menu – driven shell script to perform the following :

i) List of users who are logged in

ii) List of files in the current directory

iii) Today's date

iv) Quit to UNIX. **(08 Marks)**

b. Explain with an example "while" and "for" loop in shell programming. **(06 Marks)**

c. Briefly explain set and shift commands in UNIX to manipulate positional parameters with example. **(06 Marks)**

7 a. What is AWK? Explain any three built – in functions in AWK. **(07 Marks)**

b. Explain associative arrays in AWK. **(06 Marks)**

c. Explain built – in variables in AWK. **(07 Marks)**

8 a. Explain the string handling functions supported by PERL and also write a PERL script to convert a given decimal number to binary equivalent. **(12 Marks)**

b. Explain the following in PERL with example :

i) split

ii) join. **(08 Marks)**

* * * * *

UNIX AND SHELL PROGRAMMING SOLVED PAPER JUNE -2015

1(a) Explain the architecture of UNIX operating system with a neat diagram (8 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.1a

1(b) i) Illustrate with a diagram, the typical UNIX file system and ii) explain different types of files supported in UNIX (8 Marks)
Ans(i): For answer, refer Solved Paper June-2013 Q.No.1b
Ans(ii): For answer, refer Solved Paper Dec-2013 Q.No.1b

1(c) Explain internal and external commands with example (4 Marks)
Ans: For answer, refer Solved Paper June-2014 Q.No.1c.(ii)

2(a) Which command is used for listing file attributes? Briefly describe the significance of each field of the output (8 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.2a

2(b) A file's current permissions are rw_ r_x r__, specify the chmod expression required to change them for the following:
i) rwx rwx rwx ii) r__ r__ ___ iii) ___ ___ ___
Using both the relative and absolute methods of assigning permissions (6 Marks)
Ans: For answer, refer Solved Paper Dec-2013 Q.No.2c

2(c) What are the different modes of vt editor ? Explain with a diagram (6 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.2c

3(a) Explain the three standard files with respect to UNIX operating system (6 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.3a

3(b) Explain the mechanism of process creation using system calls in UNIX (6 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.3b

3(c) Explain the following environment variables with example :
i) SHELL ii) LOGNAME iii) PATH iv) PS2 (8 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.3c

4(a) Distinguish between hard links and soft links with suitable example (6 Marks)
Ans: For answer, refer Solved Paper June-2013 Q.No.4a

4(b) Explain the following filter with options:
i) pr ii) sort (6 Marks)
Ans(i): For answer, refer Solved Paper Dec-2014 Q.No.4c.(i)
Ans(ii): For answer, refer Solved Paper June-2013 Q.No.4b

4(c) Use find command to locate from your home directory
i) All files with the extension-html
ii) All files having inode number 9076
iii) All directories having permission 666
iv) All files not accessed for more than a year
v) All but the C program files
vi) All files named a- out & all 'C' source files and remove them interactively (6 Marks)

Ans:

```
i) find $HOME -type f -name "*.html"  -print
ii) find $HOME -type f -inum 9076 -print
iii) find $HOME -type d -perm 666 -print
iv) find $HOME -type f -atime +365 -print
v) find $HOME -type f -name "*.[!c]"  -print
vi) find $HOME -type f \(-name  a.out  -a  -name "*.c" \)  -exec rm {} \
```

5(a) Explain grep command with all options (8 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.5a

5(b) Briefly explain the different ways of addressing used in sed with example (6 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.5b

5(c) Explain BRE character subset used for constructing regular expression (4 Marks)

Ans:

Basic Regular Expression (BRE)

- grep uses an expression of a different type to match a group of similar patterns.
- This command
  - → uses an elaborate metacharacter set and
  - → can perform amazing matches.
- Character subset of BRE is explained below:

1) Character Class

- The character class comprises a set of characters enclosed by the rectangular brackets, [and].
- The character class matches a single character in the class.
- When you use range, make sure that the character on the left of the hyphen has a lower ASCII value than the one on the right.
- caret (^) can be used to negate the character class.
- For example:

Character Class	Match
[pqr]	a single character p, q or r
[x-z]	A single character between characters x to z.
^[pqr]	a single character which is not p, q or r

- Example:

```
$ grep -i "[Mm][Hh]" demo_file          // matches all the words such as MH mH Mh mh
```

2) Asterisk (*)

- The asterisk (*) refers to the immediately preceding character.
- It indicates zero or more occurrences of the previous character.
  - g* → nothing or g, gg, ggg, etc.
  - lg* → l or lg, lgg, lggg, etc.
- Example:

```
$ grep  "isaa*c" demo_file              // matches isac isaac or isaaac
```

3) Dot (.)

- A dot (.) matches a single character.
- Example:

```
$ grep  "2..." demo_file                // matches all 4 character words beginning with 2
```

- Regular expression **".*"** → signifies any number of characters or none
- Example:

```
$ grep  prog.c.* demo_file              // matches all c and cpp extension filenames
```

4) Specifying Pattern Locations (^ and $)

- Following two metacharcters can match a pattern at the beginning or end of a line.

Metacharcter	Match
^Pattern	Pattern at beginning of line
Pattern$	Pattern at end of line

- Anchoring a pattern is often necessary when it can occur in more than one place in a line, and we are interested in its occurrence only at a particular location.
- Example:

```
grep "^2" emp.lst          //Selects lines starting with 2
grep "7...$" emp.lst       //Selects lines where salary between number b/w 7000 to 7999
```

5(d) Write the command for the following:
i) Use sed to delete all blank lines from a file named sample
ii) Use sed to replace all occurrences of the word : 'UNIX' with 'LINUX' in a file named sample (2 Marks)

Ans:

i) sed '/^$/d' sample.txt
ii) sed 's/UNIX/LINUX/' sample.txt

6(a) What is shell programming ? Write a menu-driven shell script to perform the following
i) List of users who are logged in
ii) List of files in the current directory
iii) Today's date
iv) Quit to UNIX (8 Marks)

Ans: For answer, refer Solved Paper Dec-2013 Q.No.6c

6(b) Explain with an example 'while' and 'for' loop in shell programming (6 Marks)

Ans : For answer, refer Solved Paper June-2013 Q.No.6b

6(c) Briefly explain set and shift commands in UNIX to manipulate positional parameters with example (6 Marks)

Ans:

1) set Command

• set command can be used to assign positional parameters ($1, $2 and $3) to command line arguments.
• This command can be used for picking up individual fields from the output of a program.
• Example:

```
$ set 98 23 62
```

• Here, above line assigns

→ 98 to $1
→ 23 to $2
→ 62 to $3.

• This command can also be used to assign the other parameters $# and $*.
• Example:

```
$ set `date`
$ echo $*
Mon Nov 4 08:02:45 IST 2017
```

• Example:

```
$ echo "The date today is $2 $3, $6"
The date today is Nov 4, 2017
```

2) shift Command

• shift command is a shell built-in that operates on the positional parameters.
• Each time shift command is called, it shifts/transfers all the positional parameters down by one.
• For example: $2 becomes $1
$3 becomes $2
$4 becomes $3, and so on.
• Example:

```
$ echo "$@"                          # $@ and $* are interchangeable
Mon Nov 4 08:02:45 IST 2017
$ echo $1 $2 $3
Mon Nov 4

$ shift                              # shifts 1 place
$ echo $1 $2 $3
Nov 4 08:02:45

$shift 2                             # shift 2 places
$echo $1 $2 $3
08:02:45 IST 2017
```

3) set -- : Helps Command Substitution

• Problem with set command:
When set command is used with command substitution, the output of the command may begin with a -(hypen). In this case, set command interprets -(hypen) as an option and does not work correctly.
• For example:

```
$set 'ls -l student,lst'
-rwxr-xr--: bad option
```

• Solution: Use --(double hypen) immediately after set command.

```
$set --  'ls -l student.lst'
 -rwxr-xr-- 2  kumar  group  163   Jul     13      21:36 student.lst
```

7(a) What is AWK? Explain any three built-in functions in AWK (7 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.7a.(i+ii)

7(b) Explain associative arrays in AWK (6 Marks)

Ans:

Associative Arrays

- awk doesn't treat array indexes as integers.
- Awk arrays are associative, where information is held as key-value pairs.
- The index is the key that is saved internally as a string.
- When we set an array element using mon[1]="mon", awk converts the number 1 to a string.
- There's no specified order in which the array elements are stored.
- As the following example suggests, the index "1" is different from "01":

```
awk 'BEGIN {
        direction ["N"] = "North" ; direction ["S"] ;
        direction ["E"] = "East" ; direction ["W"] = "West" ;]
        printf("N is %s and W is %s \n", direction["N"], direction ["W"]);


        mon[1] = "Jan"; mon["1"] = "january" ; mon["01"] = "JAN" ;
        Printf("mon is %s\n", mon[1]);
        Printf("mon[01] is also %s\n",mon[01]);
        Printf("mon[\"1\"] is also %s \n", mon["1"]);
        Printf("But mon[\"01\"] is %s\n", mon["01"]);
              }
```

```
Output:
N is North and W is West
mon is january
mon[01] is also january
mon["1"] is also january
But mon["01"] is JAN
```

- There are two important things to be learned from this output.
- First, the setting with index "1" overwrites the setting made with index 1.
- Accessing an array element with subscript 1 and 01 actually locates the element with subscript "1" .
- Also note that mon["1"] is different from mon["01"].

7(c) Explain built-in variables in AWK (7 Marks)

Ans: For answer, refer Solved Paper June-2013 Q.No.7c

8(a) (i) Explain the string handling function supported by PERL and (ii) also write a PERL script to convert a given decimal number to binary equivalent (12 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.8a

Ans(ii): For answer, refer Solved Paper Dec-2014 Q.No.8b

8(b) Explain the following in PERL with example

i) split ii) join (8 Marks)

Ans(i): For answer, refer Solved Paper June-2013 Q.No.8b.(i)

Ans(ii): For answer, refer Solved Paper Dec-2013 Q.No.8b.(ii)