TABLE OF CONTENTS

SECTION-1

MCS-041: Introduction to Unix

SESSION 1:

1) Explore All UNIX Commands given in this manual.

Files

- **ls** --- lists your files.
  ls -l --- lists your files in 'long format', which contains lots of useful information, e.g. the exact size of the file, who owns the file and who has the right to look at it, and when it was last modified.
  ls -a --- lists all files, including the ones whose filenames begin in a dot, which you do not always want to see.
  There are many more options, for example to list files by size, by date, recursively etc.
- **more filename** --- shows the first part of a file, just as much as will fit on one screen. Just hit the space bar to see more or **q** to quit. You can use **/pattern** to search for a pattern.
- **emacs filename** --- is an editor that lets you create and edit a file. **mv filename1 filename2** --- moves a file (i.e. gives it a different name, or moves it into a different directory (see below).
- **cp filename1 filename2** --- copies a file
- **rm filename** --- removes a file. It is wise to use the option rm -i, which will ask you for confirmation before actually deleting anything. You can make this your default by making an alias in your .cshrc file.
- **diff filename1 filename2** --- compares files, and shows where they differ
- **wc filename** --- tells you how many lines, words, and characters there are in a file
- **chmod options filename** --- lets you change the read, write, and execute permissions on your files. The default is that only you can look at them and change them, but you may sometimes want to change these permissions. For example, **chmod o+r filename** will make the file readable for everyone, and **chmod o-r filename** will make it unreadable for others again. Note that for someone to be able to actually look at the file the directories it is in need to be at least executable.
- File Compression
  - **gzip filename** --- compresses files, so that they take up much less space. Usually text files compress to about half their original size,

but it depends very much on the size of the file and the nature of the contents. There are other tools for this purpose, too (e.g. **compress**), but gzip usually gives the highest compression rate. Gzip produces files with the ending '.gz' appended to the original filename.
    - **gunzip** ***filename*** --- uncompresses files compressed by gzip.
    - **gzcat** ***filename*** --- lets you look at a gzipped file without actually having to gunzip it (same as **gunzip -c**). You can even print it directly, using **gzcat** ***filename*** **| lpr**

- printing
    - **lpr** ***filename*** --- print. Use the -P option to specify the printer name if you want to use a printer other than your default printer. For example, if you want to print double-sided, use 'lpr -Pvalkyr-d', or if you're at CSLI, you may want to use 'lpr -Pcord115-d'. See 'help printers' for more information about printers and their locations.
    - **lpq** --- check out the printer queue, e.g. to get the number needed for removal, or to see how many other files will be printed before yours will come out
    - **lprm** ***jobnumber*** --- remove something from the printer queue. You can find the job number by using lpq. Theoretically you also have to specify a printer name, but this isn't necessary as long as you use your default printer in the department.
    - **genscript** --- converts plain text files into postscript for printing, and gives you some options for formatting. Consider making an alias like **alias ecop 'genscript -2 -r \!* | lpr -h -Pvalkyr'** to print two pages on one piece of paper.
    - dvips *filename* --- print .dvi files (i.e. files produced by LaTeX). You can use dviselect to print only selected pages.

Directories, like folders on a Macintosh, are used to group files together in a hierarchical structure.

- **mkdir** ***dirname*** --- make a new directory
- **cd** ***dirname*** --- change directory. You basically 'go' to another directory, and you will see the files in that directory when you do 'ls'. You always

start out in your 'home directory', and you can get back there by typing 'cd' without arguments. 'cd ..' will get you one level up from your current position. You don't have to walk along step by step - you can make big leaps or avoid walking around by specifying pathnames.

- **pwd** --- tells you where you currently are.

Finding things

- **ff** --- find files anywhere on the system. This can be extremely useful if you've forgotten in which directory you put a file, but do remember the name. In fact, if you use **ff -p** you don't even need the full name, just the beginning. This can also be useful for finding other things on the system, e.g. documentation.
- **grep** ***string filename(s)*** --- looks for the string in the files. This can be useful a lot of purposes, e.g. finding the right file among many, figuring out which is the right version of something, and even doing serious corpus work. grep comes in several varieties (**grep**, **egrep**, and **fgrep**) and has a lot of very flexible options. Check out the man pages if this sounds good to you.

About other people

- **w** --- tells you who's logged in, and what they're doing. Especially useful: the 'idle' part. This allows you to see whether they're actually sitting there typing away at their keyboards right at the moment.
- **who** --- tells you who's logged on, and where they're coming from. Useful if you're looking for someone who's actually physically in the same building as you, or in some other particular location.
- **finger** ***username*** --- gives you lots of information about that user, e.g. when they last read their mail and whether they're logged in. Often people put other practical information, such as phone numbers and addresses, in a file called **.plan**. This information is also displayed by 'finger'.
- **last -1** ***username*** --- tells you when the user last logged on and off and from where. Without any options, **last** will give you a list of everyone's logins.
- **talk** ***username*** --- lets you have a (typed) conversation with another user

- **write *username*** --- lets you exchange one-line messages with another user
- **elm** --- lets you send e-mail messages to people around the world (and, of course, read them). It's not the only mailer you can use, but the one we recommend. See the elm page, and find out about the departmental mailing lists (which you can also find in /user/linguistics/helpfile).

About your (electronic) self

- **whoami** --- returns your username. Sounds useless, but isn't. You may need to find out who it is who forgot to log out somewhere, and make sure *you* have logged out.
- **finger** & .plan files
  of course you can finger yourself, too. That can be useful e.g. as a quick check whether you got new mail. Try to create a useful .plan file soon. Look at other people's .plan files for ideas. The file needs to be readable for everyone in order to be visible through 'finger'. Do 'chmod a+r .plan' if necessary. You should realize that this information is accessible from anywhere in the world, not just to other people on turing.
- **passwd** --- lets you change your password, which you should do regularly (at least once a year).
- **ps -u *yourusername*** --- lists your processes. Contains lots of information about them, including the process ID, which you need if you have to kill a process. Normally, when you have been kicked out of a dialin session or have otherwise managed to get yourself disconnected abruptly, this list will contain the processes you need to kill. Those may include the shell (tcsh or whatever you're using), and anything you were running, for example emacs or elm. Be careful not to kill your current shell - the one with the number closer to the one of the ps command you're currently running. But if it happens, don't panic. Just try again :) If you're using an X-display you may have to kill some X processes before you can start them again. These will show only when you use **ps -efl**, because they're root processes.
- **kill *PID*** --- kills (ends) the processes with the ID you gave. This works only for your own processes, of course. Get the ID by using **ps**. If the process doesn't 'die' properly, use the option -9. But attempt without that option first, because it doesn't give the process a chance to finish

possibly important business before dying. You may need to kill processes for example if your modem connection was interrupted and you didn't get logged out properly, which sometimes happens.

- **quota -v** --- show what your disk quota is (i.e. how much space you have to store files), how much you're actually using, and in case you've exceeded your quota (which you'll be given an automatic warning about by the system) how much time you have left to sort them out (by deleting or gzipping some, or moving them to your own computer).
- **du *filename*** --- shows the disk usage of the files and directories in *filename* (without argument the current directory is used). **du -s** gives only a total.
- **last *yourusername*** --- lists your last logins. Can be a useful memory aid for when you were where, how long you've been working for, and keeping track of your phonebill if you're making a non-local phonecall for dialling in.

2) Create a directory.

To create a directory, use the mkdir command. For example, to create a directory named Student within the current working directory:

% mkdir Student

3) Create a Subdirectory in the directory created.

You may create a subdirectory within any directory where you have write permission. For example, to create a directory called/Student/Subdirectory1, assuming that directory /Student already exist:

% mkdir /Student/Subdirectory1

4) Change your current directory to the subdirectory.

To change the current working directory to the subdirectory, use the the cd command (this stands for "change directory"). Assuming that the current working Directory is Student, to change to the subdirectory Subdirectory1:

% cd Subdirectory1

5) Display Calender for the current Month.

To display Calender for the current month use the CAL command.

% CAL

6) Get a directory listing of the parent directory.

To get the Directory listing of the parent directory, use the command
ls - list contents of directory.
Goto the parent directory and use ls command.

% ls

7) How many users were logged onto your system.

To see all the other users that are currently connected to system simply use the following command who. This will simply print out a list of all the users connected (just their usernames however, not their real names), the terminal they're connected to, when they connected and where they connected from.

% who

However to quickly see roughly what the other users of the system are up to use the following command: % w
This will show a top line which contains the date, how long the systems been running, how many users are on and whats called the "load average" which is how many processes are contending for the CPU. This is shown as three numbers: the first is for 1 minute ago, the second 5 minutes ago, the third 15. All you really need to know is that small numbers are good here.

After this w will show a header that says what the columns are for, username, tty, when that user logged in, how long they've been idle, the JCPU and the PCPU and finally what the process is. JCPU is the total CPU time used by all the processes attached to that terminal, the PCPU time is just for the current

process running in the foreground. Quite often these two numbers will be the same. Knowing exactly what these numbers mean or how to use them isn't really essential knowledge, but can prove useful or interesting.

8) Display your name in form of a banner.

The **banner** Command displays a large ASCII art version of input text. It is commonly included in UNIX and Unix-like operating systems. There are two common varieties: one, which prints text horizontally for display to a terminal, and another which prints text in much larger letters for output to a line printer.

% banner Student
% banner –w80 a

9). Display the name of device name on your terminal.

To display the device name of the terminal, use the tty command:
The tty utility writes to the standard output the name of the terminal that is open as standard input.

% tty

10). Move to root directory.

To move to root directory use CD command.

% cd

SESSION 2:

11). Change your directory to the directory exercises. Create a file called example1 using the cat command containing the following text:

Water, water everywhere
And all the boards did shrink;
Water, water everywhere,
No drop to drink.

Assuming that there is a directory called exercises,

To change to directory exercises from the current working directory, use the CD command:

% cd exercises

To create a file in the directory exercises:

% cat > example1.f
Water, water everywhere
And all the boards did shrink;
Water, water everywhere,
No drop to drink.
^D

Be careful not to type ^D when you have the shell prompt, because this might log you out.

12) Use MAN command to obtain further information on the finger command.

The man command displays information from the reference manuals. It displays complete manual pages that you select by *name*, or one-line summaries selected either by *keyword* (-k), or by the name of an associated file (-f). If no manual page is located, man prints an error message.

% man command name

% man finger

Finger - display information about local and remote users

13) List all the processes that are presently running.

To find out information about the processes running on the system use the **ps** command.

% ps

And you will see a list that probably just includes your shell (tcsh). While this is fairly useful in and of itself you'll often find that you're logged in to two or

three TTYs at once doing various things, and so you'll need to find out more than just the processes running on that login.
To see a list of all your running processes use:

% ps -U $USER

Where $USER can be either the environment variable "$USER" (as that contains your username), your actual username, or if you want to see the processes of another user their username.
Another option for ps that is quite useful is the "full information" flag, this can be used as shown below:

% ps –f

This will show you information about your processes with much more information including the UID, PID, PPID, TTY and the command line and all its arguments used to start the command.

If you wanted to know this about another user you could simply use:

% ps -fu username

where "username" is the name of a user whose processes you want to see.

Lastly if you want ps to show you information about all the processes running on the system then you need to -e flag:

% ps -e

Most of time this isn't really needed, but sometimes you will want to be able to find out information like this. It should also be noted that the -e flag can be combined with the -f flag, to show all the processes on the machine and full information about them.

14) List text files in your current directory.

To find and list text files in the all the directories, use the FIND command:

% find /home -name "*.txt" -print 2>/dev/null

will search all user directories for any file ending in ".txt" and output any matching files (with a full absolute or relative path). Here the quotes (") are necessary to avoid filename expansion, while the 2>/dev/null suppresses error messages (arising from errors such as not being able to read the contents of directories for which the user does not have the right permissions).

To find and list text files in the current directory Student:

% find /Student -name "*.txt" -print 2>/dev/null

Another way to find and list text files in the current directory is to use the LOCATE command:

% locate ".txt"

will find all filenames in the file system that contain ".txt" anywhere in their full paths.

One disadvantage of locate is it stores all filenames on the system in an index that is usually updated only once a day. This means locate will not find files that have been created very recently. It may also report filenames as being present even though the file has just been deleted. Unlike find, locate cannot track down files on the basis of their permissions, size and so on.

Another option to find and list text files in the current directory is:

% ls –p | grep –v '/' file * | grep text

% ls -l *.txt ls

15) Make a copy of any text file.

To copy any text file to another text file use the cp command:
cp *file1 file2* is the command which makes a copy of **file1** in the current working directory and calls it **file2.**

First CD to your current directory:

% cd ~/Student

Then, copy the text file,

% cp /unix/examples/copy.txt.

(Note: Don't forget the dot (.) at the end. Remember, in UNIX, the dot means the current directory.)

ELSE

% cp file1 file2.

16) Rename one of your text files in the current directory.

To rename file in current directory use MV move command:

% mv file.txt filename.txt mv junk.txt newfile

17) Delete an unneeded copy of a file.

To delete a file use the rm remove command:

% rm file.txt

18) Print out any file on paper.

To print a file in UNIX uses the lpr command:

% lpr filename.txt

lpr

lpr queues the specified file for printing. If no file is specified, it reads from the standard input. Options that may be specified include printer, number of copies, processing and post-processing option.

lpq

The *lpq* program lists the job in the print queue.

lprm

The command *lprm* may be used to remove a print job from queue. The command options include the printer and job-id or user-name. If the user-name is specified all jobs for that user will be dequeued.

lpc

The command *lpc* is normally used interactively. It allows the super-user to start/stop the daemons, enable/disable queuing or printing jobs, and monitor the status of a printer. The subcommand *help* prints a short list of the commands that may be entered.

19) Send message to another user on your UNIX system, and get them to reply.

The mail command enables the user to send and receive electronic mail messages to and from users on both the Unix system and remote users.
To send a message to a user on your system, type:

% mail *username*

The cursor will move to the next line, and you will get a Subject: prompt. You can now type in the subject of your message, and then press <RETURN>. The cursor will go to the start of the next line and there will be no prompt. You now type in the text of your message. Terminate each line with <RETURN>. When you have finished the text of the message, type an end-of-file character (usually ^D), or a full-stop character.
There are several commands you can type while entering mail:
<CTRL/Z> will cancel the message, and leave the text in a file named dead.letter.
^e invokes a text editor to edit your message.
~v invokes a screen editor to edit your message.
~f reads the contents of the message you have just read, into your message text.
~r *file* reads contents of file into your message text.

20) Create a small text file and send it to another user.

To create a text file, use the cat command:

% cat > Student

To send this file to another user user 1:

% mail user1 < Student

Use the re-direction to output stream operation.

SESSION 3:

21) When you receive a message, save it to a file other than your mailbox.

UNIX uses two mailboxes to hold mail messages

system mailbox (/usr/spool/mail/)
user mail box (..../.../mbox)

Mail arrives in the system mailbox, and is saved in your user mail box after you have read it. The user mail box is normally located in their $HOME directory
To save a current mail message to a file other than the default mailbox, use the command save:

% s file name

When the mail program starts, it uses the default mail folder, which is the system mailbox. Remember the save mail message command, which saved the mail message to a file.
It was stated that the mail program treats this file as a mail folder.
By specifying the mail folder on the command line as an argument, mail will use a different mail folder.

% mail -f personal

22) Send a message to a user on a different computer system.

Sending mail to users on other computer systems is simple using mail. Simply type the full address of the remote user.

% mail Student@ignou.ac.in

23) Try to move to the home directory of someone else in your group. There are several ways to do this, and you may find that you are not permitted to enter certain directories. See what files they have, and what the file permissions are.

To move to the home directory of someone else in the group:

% cd ~ user1

To list the files in the directory and their permissions:

% ls –ldg

24) Try to copy a file from another user's directory to your own.

Before trying to copy the file, see the permissions that have been set for a file, use the ls command with -l option

% ls –l

If the –r permission is set in the group's option of the permission set, and then the file can be copied.
Assuming –r to be set,
To copy a file from another users directory to home directory of current user:

First locate the user's directory.

% finger directory1

We learn that the user1 directory is /user1/directories/directory1

To copy file named file1 from the user1 directory, do:

% cp /user1/directories/directory1 filecopy1

25) Set permissions on all of your files and directories to those that you want. You may want to give read permission on some of your files and directories to members of your group.

To set permissions on files:

Each file and directory has three kinds of permissions:

read --> permission to view, print and copy --> abbreviated r

write --> permission to change the contents --> abbreviated w

execute --> permission to run an executable file --> abbreviated x
(for example, a program) OR
permission to change into a directory

When setting file/directory permissions, Unix divides the world of users into three classes:

you, the owner --> abbreviated u

your group --> abbreviated g

others --> abbreviated o

You may assign read, write or execute permission independently to any of the three classes of users.

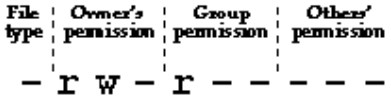
Unix is not capable of permitting files and directories to individual users, but (as explained below) files and directories are usually permitted so that others cannot access them without knowing their absolute pathnames.

Looking at permissions

To see the permissions that have been set for a file, use the ls command with -l option.

% ls -l
-rw-r----- 1 Student users 21 Jul 5 11:08 file1

The first 10 characters of the above line describe the type of the file and the permissions which have been set for it.



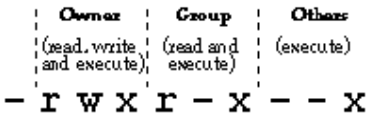
The first character shows the file type. It is - (dash) for a standard file and d for a directory.

The next 9 characters are actually 3 sets of 3 characters each. These 3 sets show the permissions for the owner, the group and others. Within each set, permissions are always

described in the same order: read (**r**), write (**w**) and execute (**x**). If the relevant letter (r, w or x) appears, permission exists. If a - (dash) appears in its place, that kind of permission is denied.

Let's look at the permissions for another file:

In this example, the owner may read the file, write to it (change it) and execute the file. Members of the group may read and execute the file, but not write to it. Other users may only execute the file, not read it or write to it.



Execute access permits the execution of binary files which contain executable programs. Both read and execute access are required to execute a shell script.

Default permissions for new files and directories

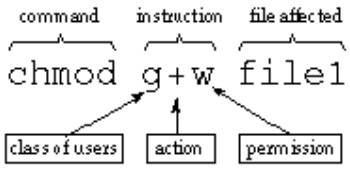
Default permissions are automatically set for files and directories as you create them. The default permission for new files is -rw------- and that for new directories is drwx------.

Changing file permissions

The command chmod (short for **ch**ange **mod**e) is used to change permissions for a file. chmod is used a bit differently from most other Unix commands. To give write permission to the group users for file1 (the first file we examined), we give the command:

chmod g+w file1

This may be understood as follows:



Class		Action		Permission	
u	user (owner)	+	add permission	r	read
g	Group	-	remove permission	w	write
o	Others	=	set permission	x	execute
a	all				

where class, action and permission can be chosen from the table of options at the right above.

More than one class and more than one type of permission can be set at the same time using chmod. For example,

chmod u+x,o=rw file1

adds the execute permission for the owner (u) and sets the permission for others to read and write explicitly (no matter what permissions others had before).

If you do not specify a class, the new permission is applied to all three classes. For example,

chmod +x file

adds execute permission for the owner, group and others.

To change the permission of all files in a directory, use the wildcard symbol "*".For example, the following command would add read permission for others to all files in the current directory:

chmod o+r *

Using numeric arguments with chmod

If you prefer, chmod can use a digit from 0 to 7 to represent the permissions for each class of people. Each digit is the sum of the permission values as shown in the following chart:

Value	Permission	Explanation
4	r	read
2	w	write
1	x	execute

For example, the command

chmod 751 file1

would change the permission for file1 to read, write and execute for the owner; read and execute for the group; and execute only for others. Values and the permissions they correspond to are shown below:

Value	Permission	Explanation
7	Rwx	read, write and execute
6	rw-	read and write
5	r-x	read and execute
4	r--	Read
3	-wx	write and execute
2	-w-	Write
1	--x	Execute
0	---	no access whatsoever

Changing directory permissions

To display the permissions for a directory, use the ls command with the -l and -d options, giving the directory name as the argument; e.g., for the directory project1

% ls -ld project1

drwx--x--x 2 Student users 512 Jul 3 11:26 .

To display the permissions for your current directory, use the -l and -d options on the ls command:

% ls -ld

drwx--x--x 11 Student users 512 Jul 8 14:54 .

Like files, permissions for directories are changed using the chmod command. Either class and action abbreviations (e.g., chmod g+x) or numeric arguments (e.g., chmod 644) may be used to change directory permissions.

Directory permissions have slightly different meanings than permissions for files. Read (r) permission is needed to list the contents of a directory with the ls command. Write (w) permission means that files can be added to or removed from the directory. Execute (x) permission is needed before you can change into a directory with the cd command or pass through a directory as part of a search path.

When you permit a file, you will also need to give execute permission to both your home directory and any subdirectories between your home directory and the file. When you do this, other users will not be able to list the contents of these directories, but they will be able to read or copy the file as long as they know the absolute pathname.
For example, user Student wants to give others permission to read and copy his file outline in subdirectory project1 in his home directory. To do this, he would type the following commands:
chmod o=x /home/Student
chmod o=x /home/Student/project1
chmod o=r /home/Student/project1/outline
Since execute permission does not allow others to see the contents of his directories, Student must tell his colleagues the absolute pathname of the file, which is
/home/Student/project1/outline

26) Create a number of hierarchically related directories and navigate through then using a combination of absolute pathnames (starting with "/") and relative pathnames.

To create hierarchically related directories, use the MKDIR command,

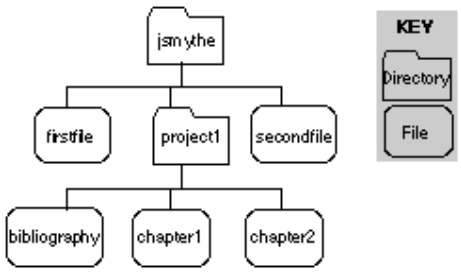
% mkdir /Student/Subdirectory1
% mkdir /Student/Subdirectory2
% mkdir /Student/Subdirectory3

To list the contents of the current directory, use the command:
% ls –r
-a → list hidden files
-c → list file names in multiple-column format

To navigate through the directories using absolute pathnames:

First determine the current directory using the pwd(print working directory) command:
% pwd
/home/Student

Within our home directory (Student), there are several files and a subdirectory which also contains files (see diagram):



Within our home directory (Student), there are several files and a subdirectory which also contains files (see diagram):

It is possible that more than one person could have a file called bibliography. How does Unix distinguish between files with the same name? The full name of each file includes the "path" through the directory hierarchy to that file.

The full names of two different bibliography files might be:

% /home/Student/project1/bibliography and

% /home/Ignou/thesis/bibliography

The names of the two bibliography files shown above are *absolute pathnames*. An absolute pathname starts with a / to represent the root directory, then traces the path through various subdirectories to the file.

Another way to describe a file is by its *relative pathname*. Relative pathnames do not begin with a /. A relative pathname shows how to get to the file from the current working directory. If we are in the directory Student, the relative pathname to our file bibliography is

% project1/bibliography

To navigate use the CD command:

If you are in the directory data, and wish to "back up" one level to the directory chapter2 (the parent of directory data), you could type

cd chapter2

but a faster way is to use the abbreviation for parent directory (..). There is no space between the 2 periods.

cd ..

Another useful abbreviation is ~ (called tilde). This refers to the user's home directory (Student in our example). The easiest way to change back to your home directory from any directory is to type

cd ~

A third abbreviation (.) means the current working directory. For example, assume that we are in the directory data, and wish to copy the file

bibliography from the directory project1 into the directory data. Without using any abbreviations, we would need to type (all on one line):

cp /home/Student/project1/bibliography
/home/Student/project1/chapter3/data

Using the abbreviation for our home directory (~) and the abbreviation for the current working directory (.) we would type only:

cp ~/project1/bibliography .

27) Try using wildcards ("*" and possibly "?").

UNIX allows you to use wildcards (more formally known as *metacharacters*) to stand for one or more characters in a filename.

The two basic wildcard characters are ? and *. The wildcard ? Matches any one character. The wildcard * matches any grouping of zero or more characters.

Assume that your directory contains the following files:

Chap	bite
Chap6	it
Lit	site
big	snit
bin	test.new
bin.old	test.old
bit	

The ? wildcard

The command ls will list all the files. The command

ls ?it

Lit bit

lists only the files Lit and bit. The file snit was not listed because it has two characters before "it". The file it was not listed because it has no characters before "it".

The ? wildcard may be used more than once in a command. For example,

ls ?i?

Lit big bin bit

finds any files with "i" in the middle, one character before and one character after.

The * wildcard

The * wildcard is more general. It matches zero or any number of characters, except that it will not match a period that is the first character of a name.

ls *it

Lit bit it snit

Using this wildcard finds all the files with "it" as the last two characters of the name (although it would not have found a file called .bit).

We could use this wildcard to remove all files in the directory whose names begin with "test". The command to do this is

rm test*

Be careful when using the * wildcard, especially with the rm command. If you had mistyped this command by adding a space between test and *, Unix would look first for a file called test, remove it if found, and then proceed to remove all the files in the directory!

Matching a range of characters with [ ]

The ? wildcard matches any one character. To restrict the matching to a particular character or range of characters, use square brackets [ ] to include a list. For example, to list files ending in "ite", and beginning with only "a", "b", "c", or "d" we would use the command:

ls [abcd]ite

This would list the file bite, but not the file site. Note that the sequence [ ] matches only one character. If we had a file called delite, the above command would not have matched it.

You can also specify a range of characters using [ ]. For instance, [1-3] will match the digits 1, 2 and 3, while[A-Z] matches all capital letters.

ls [A-Z]it

will find any file ending in "it" and beginning with a capital letter (in this case, the file Lit).

Wildcards can also be combined with [ ] sequences. To list any file beginning with a capital letter, we would use:

ls [A-Z]*

Chap1 Chap6 Lit

Matching a string of characters with { }

The method described in the previous section matches a single character or range of characters. It is also possible to match a particular string by enclosing

the string in { } (braces). For example, to list only the files ending in the string "old", we would use

ls *{old}

bin.old test.old

To list all files ending in either "old" or "new", use

ls *{old,new}

bin.old test.new test.old

28) Put a list of the files in your directory into a file called filelist. (then delete it!)

Using the CP command,

% cp ~Student/directory1* filelist

% rm filelist

29) Create a text file containing a small story, and then use the spell program to check the spelling of the words in the file.

To create a file with a short story:

% cat > story

The method described in the previous section matches a single character or range of characters. It is also possible to match a particular string by enclosing the string in { } (braces). For example, to list only the files ending in the string "old", we would use

^z

To perform Spell Check:

% spell story

30) Redirect the output of the spell program to the file called errors.

% spell story > errors

SESSION 4:

31) Type the command ls –l and examine the format of the output. Pipe the output of the command ls –l to the word count program wc, to obtain a count of the number of files in your directory.

The WC command is used to print the number of lines, words, and characters in a file. **Syntax: wc [options] filename**

To find the number of files in a directory, enter

% ls -l | wc –l

ls –l : For a long listing that shows file protections, size, and date

32) Use cut to strip away the reference material and leave just the text field.

Cut command is typically used to extract a certain range of characters from a line, usually from a file.

Syntax: % cut [-b] [-c] [-f list] [-n] [-d delim] [-s] [file]

-b

Bytes; a list following -b specifies a range of bytes which will be returned, e.g. Flags which may be used include

cut -b1-66 would return the first 66 bytes of a line. NB If used in conjunction with -n, no multi-byte characters will be split. NNB. -b will only work on input lines of less than 1023 bytes

-c

Characters; a list following -c specifies a range of characters which will be returned, e.g. cut -c1-66 would return the first 66 characters of a line

-f

Specifies a field list, separated by a delimiter

list

A comma separated or blank separated list of integer denoted fields, incrementally ordered. The - indicator may be supplied as shorthand to allow inclusion of ranges of fields e.g. *4-6* for ranges 4–6 or *5-* as shorthand for field 5 to the end, etc.

-n

Used in combination with -b suppresses splits of multi-byte characters

-d

Delimiter; the character immediately following the -d option is the field delimiter for use in conjunction with the -f option; the default delimiter is *tab*. Space and other characters with special meanings within the context of the shell in use must be enquoted or escaped as necessary.

-s

Used to bypass lines, which contain no field delimiters when -f is specified, unless otherwise indicated.

file

The file used (and accompanying path if necessary) to process as input. If no file is specified then standard input will be used

33) Use tr to strip away any tags that are actually in the text(e.g., attached to the words), so that you are left with just the words.

The tr command (translate characters) is one of the true Unix filters. It copies its standard input to its standard output, while replacing the first character specified on the command line with the second character specified.

Let's consider the text file to be example1.txt,

$ echo <B></B> this is the tag for bold text

Cat example1.txt | tr '<a-z>' 'a-z'

Output: B/B this is the tag for bold text

34) Set a file to be read-only with the chmod (from change mode) command. Interpret the file permissions displayed by the ls –l command.

Consider file1 to be present,

To set the file to be read-only to all,

% chmod a+r file1

To see the permissions of the file, use the ls –l command. To see the permissions of the entire directory use the ls –ld command.

35) Delete one or more directories with the rmdir command. See what happens if the directory is not empty. Experiment (carefully!!) with the rm –r command to delete the directory and its content.

First let's go to home directory of user Student: cd ~Student

Let us create a directory /Student/Directory2

% mkdir /Student/Directory2

If the Directory 2 is not empty, then we get an error message:

Directory2: Directory not empty.

To remove the directory and all its files and contents:

%rm –r Directory2

36) Experiment with the re-directing command output (e.g., ls –l > file1). Try ">>" instead of ">" with an existing text file as the output.

To "redirect" output -- to take what the system would have displayed on the screen and put it in a file instead.

On issuing the ls –l command, we can redirect the output to file1, as:

% ls –l >file1

When you redirect output to a file that already exists, any previous contents are deleted before the command is completed. To prevent the accidental overwriting of files, first issue the command:

set noclobber

then use % ls –l > file1

If file1 already exists then we will get an error:

File1: file exists.

However, if you are certain you want to replace the contents of an existing file with redirected output, use the emphatic form of the redirection command:

% ls >!file1

It is possible to append the redirected output onto the end of an existing file, instead of replacing the contents, by using the append symbol (>>). The following command adds the date to the end of the file file1, without removing its original contents:

date >> file1

If you have issued the set noclobber command to prevent accidental overwriting, you must use the emphatic form:

date >>! File1

37) See whether upper-case versions of any of these commands will work as well as the lower-case versions.

UNIX is *ALWAYS* case-sensitive.

If you want to list your files with the `ls' command, if you enter LS you will be told "command not found."

38) Use the who command to see users logged into the system.

% who → lists all users currently logged into the system. if you want to list your files with the `ls' command, if you enter LS you will be told "command not found."
The general format for output is:

name [*state*] *line time* [*idle*] [*pid*] [*comment*] [*exit*]

where:

name User's login name

state Capability of writing to the terminal

line Name of the line found in /dev

time Time since user's login

idle Time elapsed since the user's last activity

pid User's process id

comment
Comment line in **inittab (4)**

exit Exit status for dead processes

39) Pipe the output of the who command to the sort command.

UNIX **sort** command to sort data

- either alphabetically or numerically (**-n** option)
- in ascending or descending order (**-r** -- sort in reverse option)

By default sort sorts the file in ascending order using the entire line as a sorting key.
The **sort** command can be used in pipes or have its output redirected as desired.

To pipe the output of the **who** command to the input of the **sort** command:

%who | sort >file1

40) Search for your login name in whofile (file1) using the grep command.

The **grep** command search for the pattern specified by the Pattern parameter and writes each matching line to standard output. The **grep** command displays the name of the file containing the matched line if you specify more than one name in the *File* parameter. Characters with special meaning to the shell ($, *, [, |, ^, (,), \) must be in quotation marks when they appear in the *Pattern* parameter.

To find a word within some text, display all lines matching "pattern1",

grep *pattern1* file

% who | grep Student

SESSION: 5

41) Compare two text files with the diff command.

The diff command is used to display two files and prints the lines that are different. It prints a message that uses *ed*-like notation (*a* for append, *c* for change, and *d* for delete) to describe how a set of lines has changed. This is followed by the lines themselves. The < character precedes lines from the first

file and > precedes lines from the second file. The *diff* command displays the only line that differs between the two files.

Let's create an example to explain the output produced by *diff*. Look at the contents of three sample files:

test1	test2	test3
apples	apples	oranges
oranges	oranges	walnuts
walnuts	grapes	chestnuts

When you run *diff* on *test1* and *test2*, the following output is produced:

```
$ diff test1 test2
3c3
< walnuts
--
> grapes
```

42) Count the number of lines words and characters in a file with a wc command.

The "wc" command stands for "word count". It counts the number of characters, words, and lines that are contained in a text stream.

wc /etc/passwd

This command tells you the number of characters, words, and lines in the /etc/passwd file.

wc -l /etc/passwd

This command tells you the number of lines (only) in the /etc/passwd file.

wc -w MyStory

This command counts the number of words in the file named MyStory (which can be useful if you're paid by the word!).

who | wc -l

This command counts the number of users logged into your computer system. The output of the who command is piped into the wc command, which counts the number of lines in the who output.

ps -e | wc -l

This command counts the number of processes running on your computer system.

43) Display you current environment variables with the following command: set or env.

Standard UNIX variables are split into two categories, environment variables and shell variables. In broad terms, shell variables apply only to the current instance of the shell and are used to set short-term working conditions; environment variables have a farther reaching significance, and those set at login are valid for the duration of the session. By convention, environment variables have UPPER CASE and shell variables have lower case names.

ENVIRONMENT variables are set using the **setenv** command, displayed using the **printenv** or **env** commands, and unset using the **unsetenv** command.

To show all values of these variables, type

% printenv | less

The current environment variable settings can be displayed using the setenv command with no arguments.
To use an environment variable in a command, preface it with a dollar sign ($), for example $*NAME*. This tells the command interpreter that you want the

variable's value, not its name, to be used. You can also use ${*NAME*}, which avoids confusion when concatenated with text

44) Concatenate all files in a directory redirected to /dev/null and redirecting standard error to "error file".

To concatenate files in Unix use the cat command.

% cat *> /dev/null/errorfile

* → All files in directory.

45) Display information on current user or other user using finger command.

The **finger** displays information about the system users.
Syntax: **finger** [**-lmsp**]
-s: **Finger** displays the user's login name, real name, terminal name and write status (as a "*" after the terminal name if write permission is denied), idle time, login time, office location and office phone number.

-l: **Finger** displays the user's login name, real name, terminal name and write status (as a "*" after the terminal name if write permission is denied), idle time, login time, office location and office phone number.

-p: Prevents the **-l** option of **finger** from displaying the contents of the "*.plan*", "*.project*" and "*.pgpkey*" files.

-m: Prevent matching of *user* names.

~/.nofinger If finger finds this file in a user's home directory, it will, for finger requests originating outside the local host, firmly deny the existence of that user. For this to work, the finger program,must be able to

see the *.nofinger* file.
This generally means that the home directory containing the file must have the other-users-execute bit set (o+x).

% finger –s
To see other users: % finger username@domain
To see details in less/more than one page % finger username@domain |less/more

46) If you wish, experiment with the sending and receiving mail using the pine email program.

Pine is an easy to use, character based mail client. It supports full screen editing of messages, binary attachments (such as GIF or ZIP files), and other advanced message system features that were not possible using older electronic mail clients.

Features of pine:

- Full Screen Capabilities
- Safety for New Users – confirmation of actions.
- On-Line Help
- **Sending Attachments with Your Message – MIME encoding**
- Name Recognition (Address Books)
- Message Browsing
- Message Printing
- Saving Messages
- Pine is a menu based screen and you should look at the bottom of the screen where valid menu commands are shown

Using PINE:

1. Type **pine** at the UNIX prompt and you will see the pine screen.
2. Enter user name and password.
3. To compose mail, Use the compose command **C - COMPOSE MESSAGE** to compose a message. After composing, to send the mail, use **Ctrl+X** to

send your message. Can you see the choice **^X Send** in the menu bar at the bottom of the screen.

4. To see the mails you have received use the **L - LIST FOLDERS** command from the main menu to see your folders and choose the INBOX folder. Use **N - NextMsg** to read the next message or **P - PrevMsg** to read previous message.
5. To reply use the **R - Reply** command, and to forward mail use the **F - Forward** command.
6. You can use the **D - Delete** command to delete your unwanted mails.
7. If you want to print (or read the message you want to print, then choose the command **O - OTHER CMDS**, then choose command **% - Print**. At the bottom you will see the message **Print message *n* using "attached-to-ansi" ?** , where *n* is the message number. Answer it with **Y [Yes]** . A pop-up print setup screen will appear, and click **OK** button. Your mail content will then be printed from your local printer.
8. You can exit pine by using **Q - QUIT** command when you are in the main menu.

47) Delete all files in the current directory whose name ends in ".bak"

The command ***rm*** is used to delete files from a directory. It is used as:

[1%]**rm *filename***

Where *filename* can be in the current directory

To remove all the files with the *.bak* extension:

[1%]**rm *.bak**

To remove all files with the .bak extension, without asking for confirmation

%rm -f *.bak

48) Display lines 10-14 of any file which contains 25 lines.

To display set of lines in UNIX we use an Ex editor.

Editors available on Unix include:

ed basic line editor

ex line editor

vi screen editor
emacs screen editor
Ex is an enhanced and more friendly version of ed. Vi is a screen-based version of ex.
The command ex is used to invoke the editor. The format of this command is:
% ex [filename]
The p command (for 'print') used to display lines in the file. The format of this command is:
:[*line_range*] p
If no range is supplied the current line is displayed.
Pressing <RETURN> is equivalent to moving on to and displaying the next line. With small files it is possible to display the entire file by pressing <RETURN> until the end of the file is reached.

Line Ranges

Ranges of lines that can be given to edit commands include:
Absolute line number
6 refers to line 6
1,6 refers to lines 1 to 6
Relative line numbers
-2 refers to 2 lines before the current line
+3 refers to 3 lines after the current line
-2,+3 refers to a range from 2 lines before the current line to 3 lines after the current line
Special symbols
$ refers to the last line in the file e.g. $p to display last line, 1,$p to display entire file
. refers to the current line e.g. .,$p to display from the current line to the end
To display lines 10-14 of any file which contains 25 lines, do

% 10,14 p

49) Count how many lines contain the word science in a word file science.txt.
First find the word using the grep command and then pie it using the wc command…

% grep science science.txt | wc –l

50) List the statistics of the largest file(only the largest file) in the current directory.

One line script to find the largest file in the current directory is:

find . -type f -print | xargs ls -ladtr | cut -c34- | sort -n -r | head <1>

Note:
a) The above code should be executed in one line
(i.e. ignore word wrap)

b) Insert -xdev before the -print if you are
interested only in the filesystem you are in
and not others mounted under the directory
you are in.

SESSION: 6

51) Kill any process with help of the PID and run any process at the background.

In order to kill a process there are 2 ways:
First use ps to view what processes are running,

- **kill *pid*** where "pid" is the process ID number listed under the PID column when you use **ps**.
- Sometimes this doesn't work and you have to use a more aggressive version of kill. **kill -9 *pid*** will get the operating system to kill the process immediately. This can be dangerous if the process has control of a lock mechanisms, but your process will stop. If you find yourself completely stuck, terminals have frozen upon you and you just don't know what to do, you can log in again and use **kill -9 -1**. This command will kill all of your processes including your current log in. The next time you log in, everything that was running or was locked up will have terminated.

To run the process in background:

- Suspend the process using Ctrl+Z, This suspends execution of the process, and gives you a command line.
- To move the suspended process to the background, type **bg**. The process then resumes execution, exactly as if you had typed **&** when starting the process.
- To later bring a background process back into the foreground, use the **fg** command. This command takes an argument specifying which background process to place in the foreground. This is the job number displayed when the process was placed in the background. If you cannot remember the job number of the process you wish to bring back to the foreground, the **jobs** command will list all the jobs you are currently running and their numbers.

e.g:
Using ps to view what processes are running, this will give you the following output:

PID	TTY	TIME	COMMAND
067	console	00:05	sh
063	tty02	10:20	/bin/local/usr/zsh
062	tty02	10:22	ps
060	tty00	12:05	/net/pppd

To kill the process 063

% Kill 063

To move process 060 to background:

% bg 060

52) Select a text File and double Space the lines.

Use the SED command to double space the lines in a text file.

% sed filename

53) List all users from etc/passwd in the alphabetically sorted order.

To list all users on a Unix system, even the ones who are not logged in, look at the /etc/password file.

To see in detail, about all the users do: $ cat /etc/passwd

to just see the Unix user names, use the command **"$ cat /etc/passwd | cut -d: -f1|sort**

54) Create a file with duplicate records and delete duplicate records for that file.

To delete duplicate files:
On a single line (you can copy-paste this directly to a shell):

```
OUTF=rem-duplicates.sh; echo "#! /bin/sh" > $OUTF; find "$@" -type f -print0 | xargs -0 -n1 md5sum | sort --key=1,32 | uniq -w 32 -d --all-repeated=separate | sed -r 's/^[0-9a-f]*( )*//;s/([^a-zA-Z0-9./_-])/\\\1/g;s/(.+)/#rm \1/' >> $OUTF; chmod a+x $OUTF; ls -l $OUTF
```

Shell Script :

```
OUTF=rem-duplicates.sh;
echo "#! /bin/sh" > $OUTF;
find "$@" -type f -print0 |
 xargs -0 -n1 md5sum |
  sort --key=1,32 | uniq -w 32 -d --all-repeated=separate |
  sed -r 's/^[0-9a-f]*( )*//;s/([^a-zA-Z0-9./_-])/\\\1/g;s/(.+)/#rm \1/'
>> $OUTF;
chmod a+x $OUTF; ls -l $OUTF
```


Explanation:

1. write output script header
2. list all files recursively under current directory
3. escape all the potentially dangerous characters with xargs
4. calculate MD5 sums
5. find duplicate sums
6. strip off MD5 sums and leave only file names
7. escape strange characters from the filenames
8. write out commented-out delete commands
9. make the output script writable and ls -l it

To create a duplicate copy of the file:
$ cp filename1 filename2

To remove the duplicate copy:

55) Use the "grep" command to search the file example1 for the occurances of the string "water"

$ grep water example1

56) Write grep commands to do the following activites:

- To select the lines from the file that has exactly two characters.
- To select the lines from a file that start with upper case letters.
- To select the lines from a file that has one or more blank spaces.
- To select the lines from a file that end with a period.
- To select the lines in a file and direct them to another file which has digits as one of the characters in that line.

PART-I: MCS-043: DBMS Lab

Table Creation:

```
-- Create table
create table TEACHER
(
 T_NO       VARCHAR2(3) not null,
 F_NAME     VARCHAR2(50),
 L_NAME     VARCHAR2(25),
 SALARY     NUMBER,
 SUPERVISIOR VARCHAR2(3),
 JOININGDATE DATE,
 BIRTHDATE  DATE,
 TITLE      VARCHAR2(3),
 INC        NUMBER
)
tablespace SYSTEM
 pctfree 10
 pctused 40
 initrans 1
 maxtrans 255
 storage
 (
  initial 64K
  minextents 1
  maxextents unlimited
 );
```

```
-- Create/Recreate primary, unique and foreign key constraints
alter table TEACHER
 add constraint TEACHER_PK primary key (T_NO)
 using index
 tablespace SYSTEM
 pctfree 10
 initrans 2
 maxtrans 255
 storage
 (
  initial 64K
  minextents 1
  maxextents unlimited
 );
```



```
-- Create table
create table CLASS
(
 CLASS_NO VARCHAR2(3) not null,
 T_NO     VARCHAR2(3),
 ROOM_NO  VARCHAR2(3)
)
tablespace SYSTEM
 pctfree 10
 pctused 40
 initrans 1
 maxtrans 255
 storage
```

```
  (
   initial 64K
   minextents 1
   maxextents unlimited
  );
-- Create/Recreate primary, unique and foreign key constraints
alter table CLASS
  add constraint CLASS_PK primary key (CLASS_NO)
  using index
  tablespace SYSTEM
  pctfree 10
  initrans 2
  maxtrans 255
  storage
  (
   initial 64K
   minextents 1
   maxextents unlimited
  );
alter table CLASS
  add constraint CLASS_FK foreign key (T_NO)
  references TEACHER (T_NO);
```

```
-- Create table
create table PAYSCALE
(
  MIN_LIMIT NUMBER,
  MAX_LIMIT NUMBER,
  GRADE     VARCHAR2(5) not null
)
tablespace SYSTEM
  pctfree 10
  pctused 40
  initrans 1
  maxtrans 255
  storage
  (
    initial 64K
    minextents 1
    maxextents unlimited
  );
-- Create/Recreate primary, unique and foreign key constraints
alter table PAYSCALE
  add constraint PAYSCALE_PK primary key (GRADE)
  using index
  tablespace SYSTEM
  pctfree 10
  initrans 2
  maxtrans 255
  storage
  (
    initial 64K
    minextents 1
    maxextents unlimited
  );
```

Table: Teacher
t_no, f_name, l_name, salary, supervisor, joiningdate, birthdate, title

Table: Class
class_no, t_no, room_no

Table: Payscale
min_limit, max_limit, grade

Exercise 1:

a)
Select *
from teacher
where birthdate = (select max (birthdate) from teacher);

b)
Select *
from teacher
where title = (select title from teacher where upper(f_name)='JAIDEEP';

c)
Select *
from teacher
where join_date > '10-jul-1995'
and salary in (select salary from teacher where joining date < '10-jul-1995';

e)
Select t.*
from teacher t, payscale s
where t.salary between s.min_limit
and s.max_limit
and s.grade = 'B';

f)
Select t.*
from teacher t , payscale s
where exists (select * from class c where t.t_no = c.t_no)
and (t.salary between s.min_limit and s.max_limit) and t.grade=C';

g)
Select *
from teacher
where supervisor ='YES';

h) Select t.*, s.*
from teacher t, payscale s
where (t.salary between s.min_limit and s.max_limit)
and (s.grade 'C' or s.grade ='B') and (t.f_name like '%L%L%');

i)
Select t.*, c.*
from teacher t, class c
where t.t_no=c.t_no
and c.class_no between 1 and 5;

j) Select *
from teacher
where to_char(to_date(birthdate,'MM')) = to_char(to_date(sysdate,'MM'));

Exercise 2:

a)
Create view supervisor_details
as
select t_no, supervisior from teacher;

b)
Create index class_index on class(t_no) ;

c)
alter view supervisor_details
as
select * from teacher where supervisior='YES';

d)
select *
from teacher
where to_char(joiningdate,'yyyy')+15 > to_char(sysdate,'yyyy');

e)
create TGT view
as
select * from teacher t where t.salary >=12000 and t.title='TGT';

f)
drop view TGT ;

g)
create index teacher_index on teacher(f_name);

h)
drop index class_index ;

i)
create view teacher_info
as
select t_no, f_name, salary, grade
from teacher t, payscale s
where t.salary between s.min_limit and s.max_limit and s.grade = 'B';

j) create view as select * from teacher where (to_char(to_date(birthdate,'yyyy')) - to_char(to_date(system,'yyyy'))) > 40;

Exercise 3:

a) create or replace procedure EX3A(TNO in out varchar2, bonus out float)
is
  sal integer;

begin
  select salary into sal from teacher where t_no = UPPER(tno);

  if sal > 10000 then bonus := sal * 0.10;
  elsif sal >10000 and sal <20000 then bonus := sal * 0.20;
  elsif sal >20000 and sal <25000 then bonus := sal * 0.25;
  elsif sal >25000 then bonus := sal * 0.30;
  end if;

end EX3A;

c) create or replace procedure EX3C(result out varchar)
is
  sal integer;
  teacher_count integer;
begin
  result := 'SUCCESS';

  begin
  select count(1) into teacher_count from teacher t where t.salary = 20000;
  end;

    declare
    cursor sel_tea is select * from teacher t where t.salary = 20000;
    sel_tea_rec sel_tea%rowtype;

    begin
      if not sel_tea%isopen then
      open sel_tea;
      end if;

       if teacher_count <= 5 then

```
        loop
        fetch sel_tea into sel_tea_rec;
        exit when sel_tea%notfound;

          update teacher
          set inc = sel_tea_rec.salary * 0.50
          where t_no = sel_tea_rec.t_no;

        end loop;
        end if;
        close sel_tea;
end;
end EX3C;
```