# Birla Institute of Technology & Science, Pilani
# Computer Networks (CS F303/IS F303)
# Getting familiar with the CLI

**Aim:**Getting Familiar with the Linux Command Line Interface
**Objective:** To learn about the various commonly used UNIX commands and the various options available with each.
**Prerequisites:** None
**Resources required: A** computer with any Linux distribution installed
**Description**
The CLI has been in use since the time of mainframes where people used to connect to a central computing resource through a terminal. CLI is popular even today with power users who want to have more control over their system and automate repetitive and mundane tasks. We will be exploring common commands line utilities (aka commands) which we will be using frequently. We will also touch upon file permissions, installing packages and how to get help when you're stuck. Note that only the most common options for a command are discussed. If you are interested, please take a look at the respective man pages for more info
**Getting Help**
Often you may be stuck in one of the tasks when the commands don't function as expected or you may not understand what's going on. Sometimes, you may want to learn more about the various options available for a command so that you can better use it. As a first step, it is suggested that you look up the man pages for the command rather than looking for it on the internet or asking for a TA's assistance as they both assume that you have looked up the man pages already.
Figure 1: Please read the man pages first!

man <command name> will open up the relevant man pages for the command. Sometimes, it is possible that the page that opens up is not the one you are looking for. This is because of name conflicts. Here's how to resolve the issue man pages are organized into sever sections as follows

1. Executable programs or shell commands
2. System calls
3. Library calls
4. Special files
5. File formats and conventions
6. Games
7. Miscellaneous
8. System administration commands
9. Kernel routines

man [section number] <command name> would lead you to the right page. Once in the page, typing / followed by a keyword will search the document and highlight all occurrences. Pressing n and N allows you to scroll through the results.

If you do not know the name of the command then the man command is of no use. In these cases, the apropos utility is helpful. apropos searches the manual page names and descriptions. the syntax is apropos <keyword¿ this searches for the keyword in the man pages database and comes up with a list of results. Now you can use man to find more information about the command
**Installing Packages**
To install new software, in windows you would open an .exe file and run the setup. Similarly, in linux terminology, these are called as packages and are installed through a package manager. Some popular package managers:

aptitude, apt (debian and debian derived distros like ubuntu), yum (fedora, RHEL, CentOS), pacman (Arch Linux). We will use apt for our examples

**Installation**

To install a package, the syntax is apt-get install <packagename>
For example, apt-get install g++

**Removal(Uninstall)**

To remove a package, the syntax is apt-get remove <packagename>
For example, apt-get remove g++

**Search**

To search for an installed package, the syntax is apt search <expression>
For example, apt search g++ will give you the list of all packages which have g++ in their name

**NOTE:** install and remove may require super-user permissions (If you see textttneed to be root! error). To fix this, append sudo to the beginning of the command. For example, texttsudo apt-get install g++

**Redirection**

By default, all UNIX commands take input from STDIN (keyboard) and print their output onto STDOUT (terminal) and their errors to STDERR (terminal). These correspond to file descriptors 0,1 and 2. If you want to change this default behaviour, say to read from a file or save the output into a file, we use redirection. This can be done by using |, > , >>
The | (pipe symbol) is used to transfer the output of the process to the left of it to the input of the process to the right of it. ls | wc redirects the output of ls as the input of wc. Note that this can be extended to several processes. Try out ls | tail | wc and try to understand what's going on.
The >, >> are used usually to write into or read from a file. When > is used, the contents of the file are overwritten with the process's output. ls > list overwrites the contents of list with ls's output
If we want to append to the contents rather than overwrite them, >> is used

**File Permissions**

**Brief Description**

Every Unix file has a set of permissions that determine whether you can read, write, or run the file. Running ls -l displays the permissions.
Here's an example of such a display:
-rw-r--r-- 1 juser somegroup 7041 Mar 26 19:34 endnotes.html
The first column represents the mode of the file. The mode represents the file's permissions and some extra information. There are four parts to the mode; The first character of the mode is the file type. A dash (-) in this position, as in the example,denotes a regular file, meaning that there is nothing special about it. This is by far the most common kind of file. Directories are also common, carrying d in the file type slot. The rest of the file types are listed in the man page of ls.

The rest of a file's mode contains the permissions, which break down into three sets: the user,group, and other permissions, in that order. For example, the rw- characters in the example are the user permissions,r-- are the group permissions, and r-- are the other permissions. The user permissions (the first set) pertain to the user who owns the file. In the preceding example, that's juser.
The second set, group permissions, are for the file's group (somegroup in the example). Any user in that group can take advantage of these permissions. (Use the groups command to see

what group you’re in) Everyone else on the system has access according to the other permissions. These are sometimes called world permissions.

**NOTE** Each read, write, and execute permission slot is sometimes called a permission bit. Therefore, you may hear people refer to parts of the permissions as ”the read bits.” Some executable files have an s in the user permissions listing instead of an x. This indicates that the executable is setuid, meaning that when you execute the program, it runs as the file owner instead of you. Many programs use this setuid bit to run as root to get the special privileges they need to change system files. One example is the passwd program, which needs to change the /etc/passwd file.

**Modifying Permissions**

To change permissions, use the chmod command. First, pick the set of permissions that you want to change, and then pick the bit to change. For example, say that you want to add group (g) and world (o, for”other”) read (r) permissions to file.

You can do it with two commands:

chmod g+r file chmod o+r file

Or you can do it all in one shot like this: chmod go+r file

To remove these permissions, use go-r instead of go+r.

**NOTE** Obviously, you shouldn’t make your files world writable because it gives anyone on your system the ability to change them.

You may sometimes see people changing permissions with numbers, for example: chmod 644 file

This is called an absolute change, because it sets all of the permission bits at once. To understand how this works, you need to know how to represent the permission bits in octal form (each numeral represents an octal number corresponds permission set). If you’re curious about this, look at the chmod man files

Directories also have permissions. You can list the contents of a directory if the directory is readable, but you can only access a file inside if the directory is executable.One common mistake people make when setting the permissions of directories is to accidentally remove the execute bit when using absolute modes.

**Symbolic Links**

A symbolic link is a file that points to another file or a directory, effectively creating an alias. It is similar to a shortcut in Windows. Symbolic links offer a quick way to provide access to an obscure directory path.

In a long directory listing, these links look like this (notice the l as the file type in the file mode):

lrwxrwxrwx 1 ruser users 11 Feb 27 13:52 foo -> /home/bar

If you try to access foo in this directory, the system gives you /home/bar instead. Symbolic links are nothing more than names that point to other names.The names of the symbolic links and the paths to which they point don’t actually have to mean anything. In the preceding example, /home/bar doesn’t need to exist. If /home/bar does not in fact exist, any program that accesses foo just reports that foo doesn’t exist (except for ls funk).

Another problem is that you cannot identify the characteristics of a link target just by looking at the name of the link. You must follow the link to find out if it goes to a file or directory. Your system may also have links that point to other links, which are called chained symbolic links.

To create a symbolic link from target to linkname, use this command: ln -s target linkname

The linkname argument is the name of the symbolic link, the target argument is the path of the file or directory that the link points to, and the -s flag specifies a symbolic link

When making a symbolic link, check the command twice before you run it, because there are a number of things that can go wrong. For example,if you reverse the order of the arguments (ln -s linkname target) you're in for some fun when linkname is a directory that already exists. In this case, ln creates a link named target inside linkname (the link points to itself unless linkname is a full path). Therefore, if something goes wrong when you create a symbolic link to a directory, check that directory for errant symbolic links and remove them.

**Pagers**

When you are viewing a document on the terminal. it is possible that it is so large that it does fit the screen and you would not be able to see the entire document or scroll through it. To enable scrolling through the document, there are UNIX utilities called Pagers which takes in a document as input and splits it into pages for easy viewing. Two commonly used pagers are more and less.

The syntax for these are:

more filename less filename

more is an older utility and you can only scroll down using it.That is, you can go from page 1 to 2 but you cannot go from page 2 to 1. less is more advanced and allows you to scroll both up and down.

Here's how you navigate through a file opened using less

Ctrl+f forward one window

Ctrl+b backward one window

Ctrl+d forward half window Ctrl+u backward half window j navigate forward by one line k navigate backward by one line 10j 10 lines forward

10k 10 lines backward

Ctrl+g show the current file name along with line, byte and percentage stats G go to the end of file g go to the start of file q or ZZ exit the less pager

**NOTE** In some Linux distributions, more is hardlinked to less so that even when you think you are invoking more you are actually invoking less

**Other Commands**

**Basic Commands**

**ls**

The ls command lists the contents of a directory. The default is the current directory. Use ls -l for a detailed (long) listing and ls -F to display file type information Here is a sample long listing,

total 100

drwxr-xr-x 2 root root 4096 Jan 2 23:53 bin drwxr-xr-x 4 root root 4096 Dec 6 23:52 boot drwxr-xr-x 15 root root 4180 Jan 10 11:33 dev drwxr-xr-x 149 root root 12288 Jan 9 13:03 etc drwxr-xr-x 3 root root 4096 Oct 22 04:15 home drwxr-xr-x 24 root root 4096 Dec 6 23:48 lib drwxr-xr-x 2 root root 4096 Apr 11 2014 mnt dr-xr-xr-x 236 root root 0 Jan 9 18:30 proc drwx------ 10 root root 4096 Jan 3 00:29 root lrwxrwxrwx 1 root root 30 Oct 22 06:35 vmlinuz

**cp**

The cp command is used to copy the contents of one file into another. If the destination file is not empty, the contents of the destination are overwritten. If the destination does not exist, the file is created and written into. The syntax is cp <source> <destination>

**mv**

The mv command copies the contents of the source file into the destination file and the source file is deleted. If the destination does not exist, the file is created and written into. The syntax is mv <source> <destination>

**touch**
the touch command is used to create a file of zero bytes. The syntax is touch <filename>. For example touch test creates a file named touch of size zero bytes. Multiple files can be created by listing the file names one after the other. An example is touch foo bar. This creates two files foo,bar of size zero
**rm**
To delete (remove) a file, rm is used. After a file is removed, it's gone. Do not expect to be able to "undelete" anything. rm foo removes the foo file.
**echo**
The echo command prints its arguments to the standard output.It is very useful for finding expansions of shell wildcards and variables.
For example, echo $PWD prints out the current working directory
**Directory Commands**
Unix has a directory hierarchy that starts at /, sometimes called the root. The directory separator is the slash (/), not backslash ( \).There are several standard subdirectories in the root directory, such as /usr, /boot and so on. See man hier for information
A directory specification is called a path, and one that starts at the root (such as /usr/lib) is a full or absolute path. Similarly, a filename with a full path in front (such as /usr/lib/libc.a) is a full pathname.
The path identified by two dots (..) specifies the parent of your shell's current directory, and one dot (.) specifies the current directory. For example, if the current working directory of your shell is /usr/lib, the path ../bin refers to /usr/bin. A path beginning with .. or . is called a relative pathname.
**cd**
The commonly syntax for cd is cd dir. The cd command changes the shell's current working directory to dir. If no directory is mentioned, then cd returns to your home directory. If - is used instead of dir, then cd returns you to the previous directory you were in.
**mkdir**
The mkdir command is used to create a new empty directory. It reports an error of a directory of the same name is present in the specified location.
The syntax is mkdir dirname mkdir creates an empty directory with the name dirname
**rmdir**
The rmdir command is used to remove an empty directory.
The syntax is rmdir dirname
If the directory isn't empty, this command fails. However,you can use rm -rf dir to delete a directory and its contents, but be careful. This is one of the few commands that can do serious damage, especially if you run it as the superuser. The -r option specifies recursive delete, and -f forces the delete operation. Don't use the -rf flags with wildcards such as a star (*). Above all, always double-check your command.
**Conclusions**
In this labsheet we have extensively covered most of the Command line utilities that are widely used. It is important that you practice using these commands to get a better hang of them. Do explore the additional options available for each of the utilities as these make them highly flexible and applicable in many situations.
One important class of utilities that has not been covered are the text editors. Since there are many options available like gedit,leafpad,sublime text,kate,nano,pico,vim,emacs, it is not possible to cover every one of them or choose a favourite one as this depends on individual preferences. It is suggested that you find which suits your needs the best and stick to it.