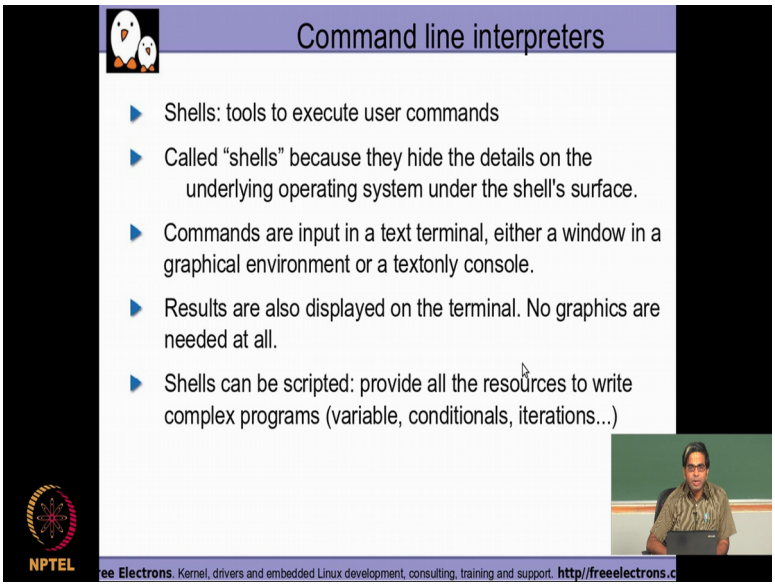
Information Security-3
Sri. Vasan V S, principle consultant
Department of Computer science and Engineering
Indian Institute of Technology Madras
Basics of Unix and Network Administration
Operating Systems
Mod02 Lecture14
Module 14: Shells & Commands on File Handling

So in this module we will actually be looking at the shell, what is a shell? Why do we have to use a shell and some very simple commands on file handling?

(Refer Slide Time: 0:25)



So shell is nothing, but a tool to execute user commands. So when a system is actually booted up, the OS is actually starting up and then the first functionality is the kernel image is actually go into get loaded in my memory and it is actually go into start execution, but as far as end user is concern, the end user requires an interface to run the different kinds of commands that you want to run on the system and then make user system.

So the designers of the unix operating system, basically gave a program called as a shell so that this gives an interface to the user to run the command whatever he (())(1:11) intends to do at that particular point in time and the shell is expected that it first validates the command that the user as actually entered , once it is being validated successfully, it goes and executes that the command that the user has given right now and then acts on the output of the command, so either by displaying the output on to the standard terminal or by taking the output to different location and so on and so fore.

So shell is basically a very very powerful utility without which the end users will not be able to really make use of the system to the most, optimal exchange. So why is an actually called a shell , because as we were saying saying right now, this particular shell program is a one that is sort of abstracting all the complicated internal details of the operating system, needs to do for providing the different services to the end user and makes it very simple by which the user is just needing to understand and remember what command to run for his particular requirement run the command and then the output automatically get appropriately displayed to the user.

So commands are basically input into a text terminal and this text terminal could either be a window in a graphical environment or it could be even console window that is actually directly connected into the hardware system. Now when will the user typically get shell process to make (())(2:40) the commands execute for him. So the user goes through what is called as a login process wherein, in a unix system being a multi user system, every user is actually provided with the separate login and a separate password and the user authenticates the system basically authenticates the user depending on whether the user as actually type the correct login and the password combination for that particular user.

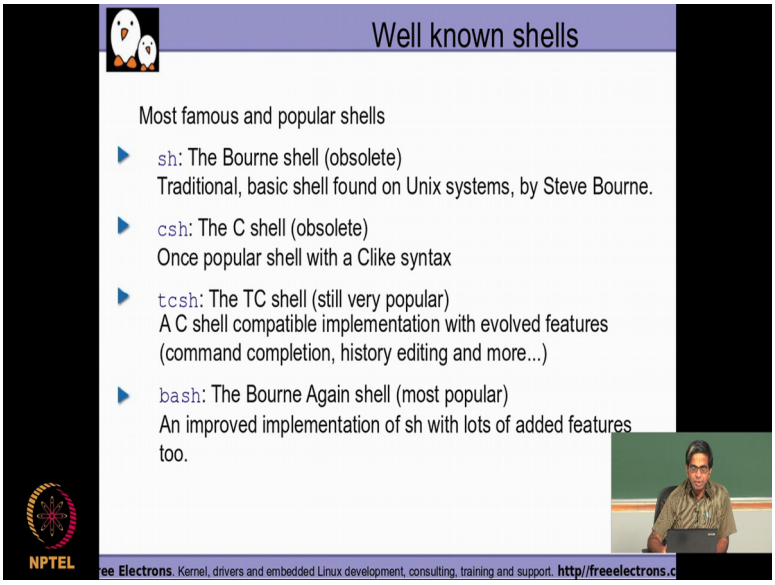
So assuming that the user has actually given the correct user name and the password combination, the system after doing the authentication of this combination provides a user with a shell process and the shell process displays a prompt what is very popularly called as a shell prompt in different unix literature in which, the user is expected to type all the commands one by one that he wants the shell to execute for him, right.

So the results will also be either displayed on the terminal or it could be on the console window depending on the where the commands has been given and later on in the modules we will see how we could also have the set of results not displayed on the terminal, but actually sort of what we called as redirected to some other location in my system, so be it some file in my file system or to another terminal and so on and so fore.

So we will also see subsequently in other modules down below where we will be also talking about how the different kinds of commands could be typically automated by writing what is called as a shell script, so some of us would have already have experience about programming in different kinds of languages. So we will introduce language called as a shell script language with which, we will be able to write programs or scripts that could actually automate lot of task and sometimes, it could be very complex task also, and very easily

automated and just like a normal programming language the shell scripting language also, gives us the mechanism of having variables, so different kinds of Boolean operations, right different kinds of iterations loops like for, while and so on. So we will see all these, possibilities of how we could write shells scripts from a very simple shell script to very complicated shell script.

(Refer Slide Time: 5:14)



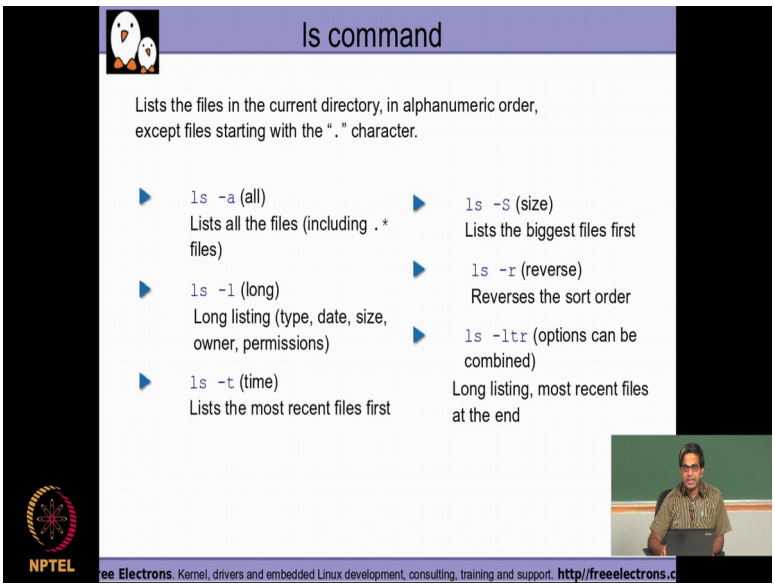
So there are different types of shells that you would be coming across the first and the fore most the original shell was actually called as sh and it was referred to internally as a Bourne shell , the name Bourne shell came in, with the memory of the the person who actually written this program, he was named as Steve Bourne and on his name the shell was also given the name of the Bourne shell, right now this particular Bourne shell is slightly old (())(5:47) version and it sort of obsoleted with the latest unix distributions not having this type of shell, but that is actually the first shell that actually came out as the most popular shell, then you have something called as a csh , which stands for the C shell, the basic reason behind why this particular shell was called as C shell was that or the syntax of its scripting language was very very similar to the C programing language.

So it had very similar syntax to the C programing language from which it derived the name of C shell for itself and that again is something which has obsolete with not many unix distribution supporting it nowadays, then you have something called as tcsh, which is again an expanded C shell with more added features as compared to the original csh, but the most popular shell that is actually used today is what is called as bash, right.

So bash stands for Bourne again shell and it is like the latest avatar (())(6:58) of the original Bourne shell that had come out with lot of features in such a way that it has become so popular that bash shell is right now, the default shell that is actually provided for every user ID that has been created on any unix system today, so whether you take a Linux or a solar rays or HPvx (())(7:20) or a (())(7:20) by default you will find today that by default, you have the bash shell assigned for any kind of user who is actually created on the system.

of course if the user wants or is having a specific preference that you would like to have only a particular shell may be he is a actually a big fan of tcsh or some other shell that could be getting change, but the default shell that actually get assigned for a user in almost all kinds of unix distributions today is nothing, but the bash shells. So we will also be seeing subsequently in the later on modules on how we would actually be making use of the bash shell? What are the different scripting syntaxes that you need to follow as far as bash shell is concern and so on so that we get some sort of comfort feeling on at least writing some simple bash scripts for automating some of were very repetitive (())(8:20) jobs on our unix system.

(Refer Slide Time: 8:25)



So some very basic set of commands, the ls command is the command that is actually used for listing, the files in the current directory that I have, so by default ls is used to list the files in the current directory, but it could also be used to list the files in any other directory by providing it is an argument. So let us say that, for example, I am in my home directory currently and I want to see the contents of another directory like for example, slash tmp. So as we saw in the previous module, tmp is basically a directory location where all my temporary files are going to be stored and if I want to see the contents of the slash tmp

directory when I am physically present in the current shell, in my home directory, I could say ls space slash tmp, right. So it will now list me all the files and the sub-directories that is available inside my slash tmp directory.

So there are different options to the ls command that is also available, so an option in unix command is typically mechanism by which, I enable certain very specific functionality of that particular command. So by the name option just like the name option denotes, it is not mandatory for us to include the option, every time we try to run the command, but if it all we need to exercise certain specific functionality of the command then we need to use that particular option along with the command name in the command line.

So let us see a few options of the ls command , all options in a any unix command always have to start with the hyphen or the minus symbol, right. So as you see here, you have ls minus a, the a stands for all, so now what does this (())(10:19) all mean is? we were talking about the hidden the files in unix wherein any file name that is actually starting with the dot character is considered as a hidden file, by default the ls command will not list all the hidden files and if I want to see the contents of the hidden files also I need to specify this minus a option. So when I say minus a, all file name that is actually starting with a dot that is considered as a hidden file by the definition of hidden file in unix will also be getting listed whenever I use the minus a option here, right.

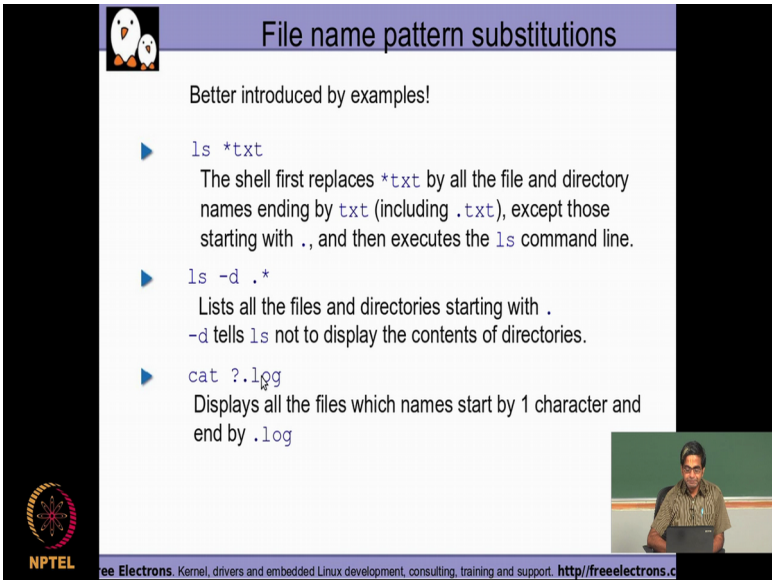
Then another option is minus l, which actually stands for the long, if I want to know more details about every file and sub-directory that is available in a particular directory like, for example, the permissions on that particular entry, the size of the file or the directory, right if I want to know who is owning that file and when as it been modified and so on, then I will typically make use of the minus l option, so if I say ls minus l, it will give me more details about the each and every entry that is actually present in that particular, directory. So the minus l stands for long. ls minus t minus t stands for the time, it basically sorts the list of files in such a way that my recent files will be getting listed at the top and the older files, which has been modified much earlier will be getting listed down below that. So in the order of the modification time of the files in that particular directory, the ls minus t option will be listing me the contents of that particular directory.

So ls minus capital S okay, so we just saw in the previous module that the command name is actually sensitive, the file name is actually case sensitive just like that file name are case sensitive, the options too every command in unix is also case sensitive, so we need to be little

careful with respect to what option we use. So ls minus capital S will basically sort the output of my ls command based on the size of the file. So the biggest files will be listed first and smaller files in terms of the size will be listed below, ls minus r will typically reverse the sort order, so for example, if I say ls minus ltr, right. So minus l means it is long listing as we seen here, minus t means we saw that it is a most recent files first, but what is actually I am introducing the option of r along with minus t option here, right. So what is going to happen is the minus r option is reversing the sort order of the minus t, so what does it mean, because minus t by default is going to list the recent files first, and we are reversing the sort order when we use minus ltr, it is actually go into have the most recent files, listed at the end of it end of listing instead of the beginning of the listing, right.

So that is one example of the minus r option likewise I could use minus r even with minus capital S in which case, the files that are the biggest will be at the end of the listing rather than at the beginning of the listing, so likewise I do have many different options to the ls command, but what we have actually seen here is only some of the very very common options that you would need to make use of typically with an ls command.

(Refer Slide Time: 14:07)



So along with the options, I could also do what is called as file name pattern substitution in the ls. Now, if you see this example, ls star txt, star is what we refer to as a regular expression pattern, which would typically stand for 0 or more characters. Now, what we mean when we say star txt is that, I want to substitute star with zero or more characters and any character, right, but the name of the file or the directory should always be ending with txt, because I do

not have anything after that here, right. So what the command we will try to list down here is that, it will try to find out all the names that have actually ending, with txt, in the current directory and then list them down line by line, right. So that is basically, because of the fact that I am using the ls.

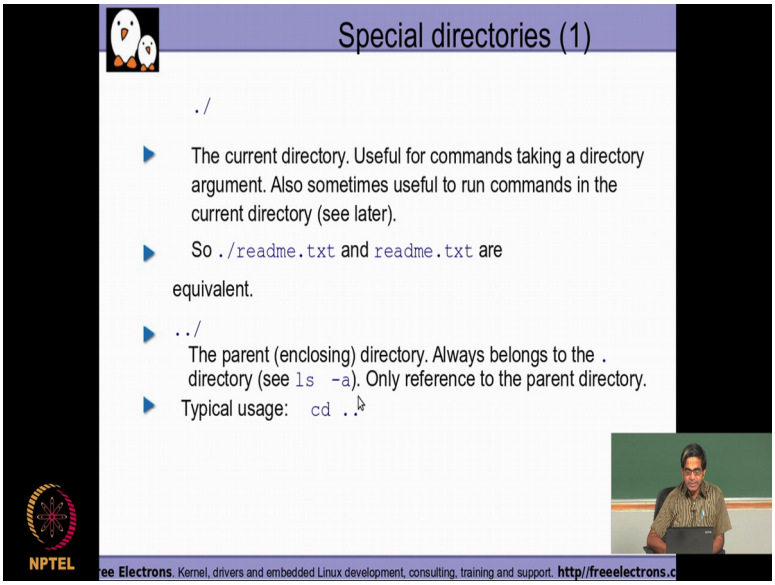
So here if I have let us say two files, which are ending with txt or 5 files ending with txt, but I do not know the names of the files to be explicitly mentioning here as part of the command line, I could actually use this mechanism where, regular expression character like star, will replace any number of characters in that particular position, but it always have to be ending with txt, because that is the pattern that I have actually given here, right.

Now look at this, ls minus d dot star another example, wherein I am going to have all the names starting with dot and if it all (())(15:57) any of those names starting with a dot is a directory with the minus d option, we are saying that do not go inside the directory that particular directory and display the contents of the directory also, but just display me the directory name alone, right. So dot star will typically expand or rather substitute to any pattern that is actually starting with a dot, followed by zero or more characters.

So any kind of a hidden files will be matched by this particular regular expression pattern and because of the fact that we have used minus d, if this regular expression pattern is a directory name it will only display that name of the directory alone as part of the file listing and not go into each and every content of that particular subdirectory. If you come into the next example, cat you see the question mark dot log, the question mark here stands for one single character. So any file name, which actually has one character, start the starting followed by dot log will be matched by this particular pattern substitution.

So let us say that in my particular directory in which I am running this command, if there is a file name called a dot log, okay that particular file name, will be matched by this pattern substitution, because a question mark stands for exactly one character, on the other hand if I have the file name the same directory with (())(17:35) has name of let us say aa dot log, right that particular file name will not be matched by this, because there are two characters aa before my dot log, but the pattern here is saying that it should be only one character. So the file name a dot log will match with this particular pattern substitution, but the file name aa dot log for example will not substitute for this particular pattern substitution, because of the fact that have two characters before the dot log in that particular file name.

(Refer Slide Time: 18:11)

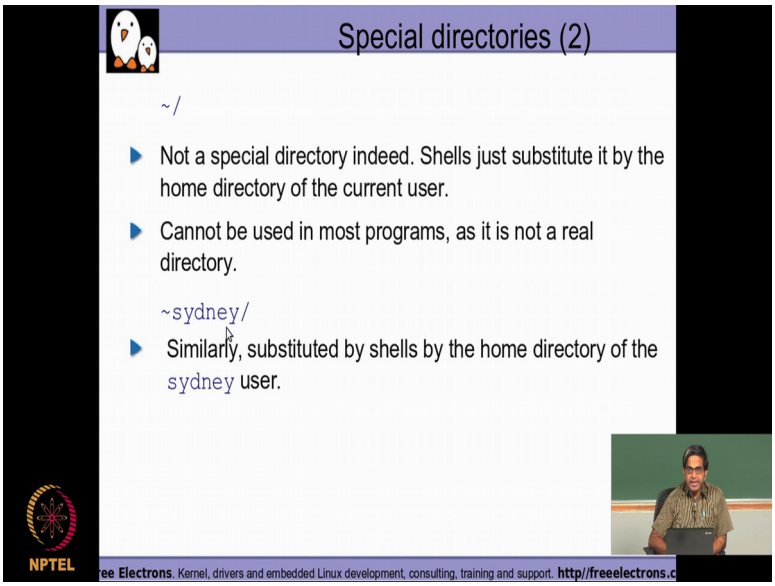


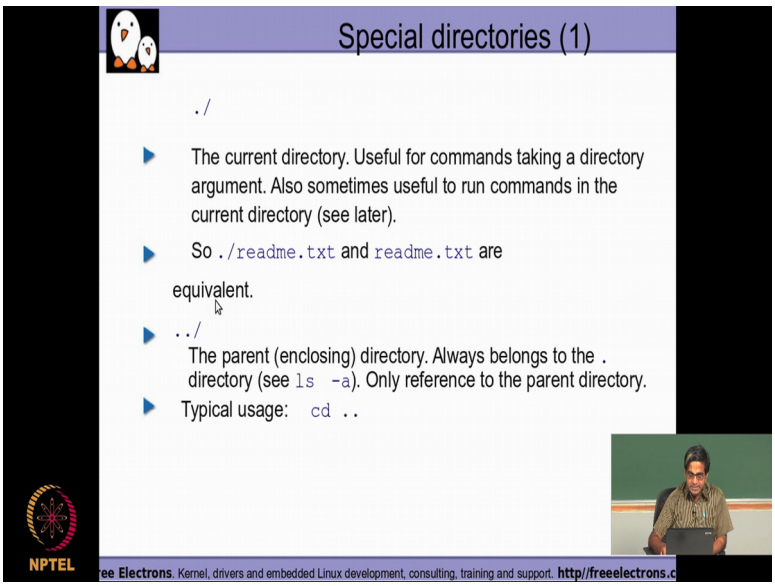
So there are some special directories that we need to understand whenever we are dealing with a unix operating system, one special directory is what is referred to as a dot. So the dot here stands for the current directory. So wherever my shell process is currently executing the commands that directory becomes my current directory. So later on we will see, how I could actually change the current directory, but the dot character in the command line is a representation that it is being referred to the current directory in that particular location. So for example, if I say dot slash readme dot txt and if I also say it is readme dot txt both are actually one and the same, because readme dot txt if you recall what we talked about in the file path, it is not stating with slash that means it is relative path to my current location. So when it is a current location that is nothing but my dot slash, because dot as we as a just discussing stands for the current directory, right.

So the dot here whether I say dot slash readme dot txt or readme dot txt both would actually mean one and the same and we will be actually seeing where we would be using the current directory as a dot in extensive (())(19:35) detail in our subsequent modules, while dot represents a current directory. There is also another a special directory called as a dot dot directory. Now what is a dot dot directory is, it is a parent directory of the current directory, right. So because we will have to understand to the entire file system in unix is typically a tree based model, where you have a parent you have a child and also the grand child and so on in the hierarchy. So every file name will be present inside a particular directory and that directory will have a parent likewise it will be going up till the root. So the root alone will not have the parent directory, right.

So if I want to basically refer to the parent directory you will find that programmers will generally make use of dot dot to refer to the parent directory of my current directory location. So for example, if I want to change into my parent directory or wherever I am right now I will say cd dot dot, so as we will see later cd is basically command to change directory. So I am basically telling I want to do a change directory from the current directory into my parent directory so that is a reason why you find dot dot being mentioned here.

(Refer Slide Time: 21:00) (22:24) (22:30)2 slides





Then another kind of a special directory is what we refer to as the tilde (())(21:06) directory, where , this will referred to the home directory of the current user. So as we were discussing right now unix being a multi-user operating system, every user has some something called as home directory in which he will be typically lock them as soon as he comes into the system

and from that home directory you will be expected to go to whichever other directory you wants to go.

Now as part of the script or as part of referring to a file, if there is a requirement to refer to the home directory , this particular character is actually a short form of representation, which will directly refer to the home directory of the currently locked in user, right. On the other hand if we actually follow this special character with the name of a user, it represents that we want to go to the home directory of this particular user, who has been actually mentioned here, right so that essential difference is if I do not specify the user name after this tilde directory, it refers to my own (())(22:16) home directory, but on the other hand, if I refer to user name after this tilde directory, it refers to the home directory of that particular user. (22:24)

(22:30) So if I say for example cd the command that we actually saw in the previous slide and then say cd tilde Sydney that means I want to change my current directory to the home directory of the Sydney user, on the other hand I say cd space tilde, it means that I want to just change to my own (()))(22:47) home directory, right. So these are very powerful short cut mechanisms by which we will be able to refer to different types of files and also special type of directories , which will come in extremely (())(23:05) for us when we actually start running small or very complex scripts on our unix system. So this brings us to the end of this particular module.