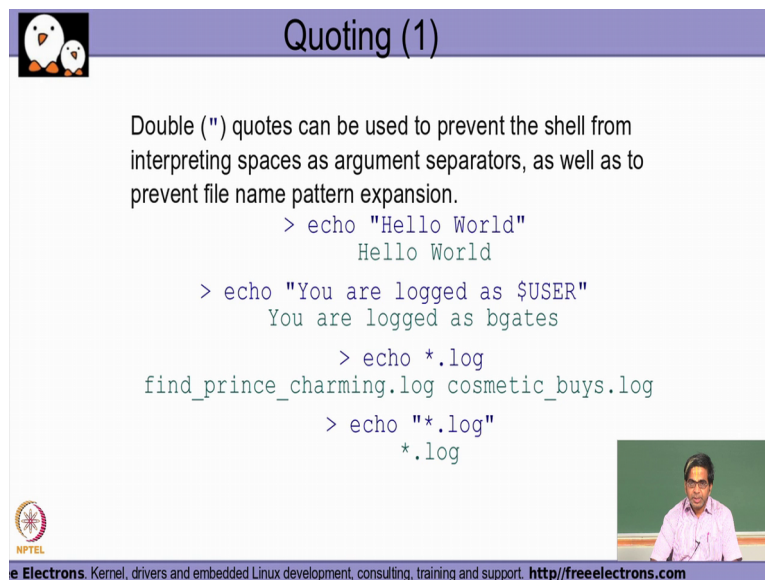


Information Security
Shri Vasan V S, Principal Consultant
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Mod03lec20
LINUX Shell Environment

In this module, we would actually look at the basic environment the shell is actually providing for the user to make use of its features. So the first thing that we look at is, what is called as a quoting.

(Refer Slide Time: 00:26)



The slide, titled "Quoting (1)", features a purple header with a penguin icon on the left. The main content is a terminal window showing the following commands and their outputs:

```
Double (") quotes can be used to prevent the shell from interpreting spaces as argument separators, as well as to prevent file name pattern expansion.  
> echo "Hello World"  
Hello World  
  
> echo "You are logged as $USER"  
You are logged as bgates  
  
> echo *.log  
find_prince_charming.log cosmetic_buys.log  
  
> echo "*.log"  
*.log
```

At the bottom left is the NPTEL logo, and at the bottom right is a small video inset of a man in a pink shirt. The footer contains the text: "Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>"

So when we say quoting it could either be a double quote or it could be a single quote. So first take a look at this behaviour of whenever we give a double quote. A double quote can be used to prevent the shell from interpreting spaces as argument separators as well as it could be used to prevent any kind of a name pattern expansion expansion. Right ...?

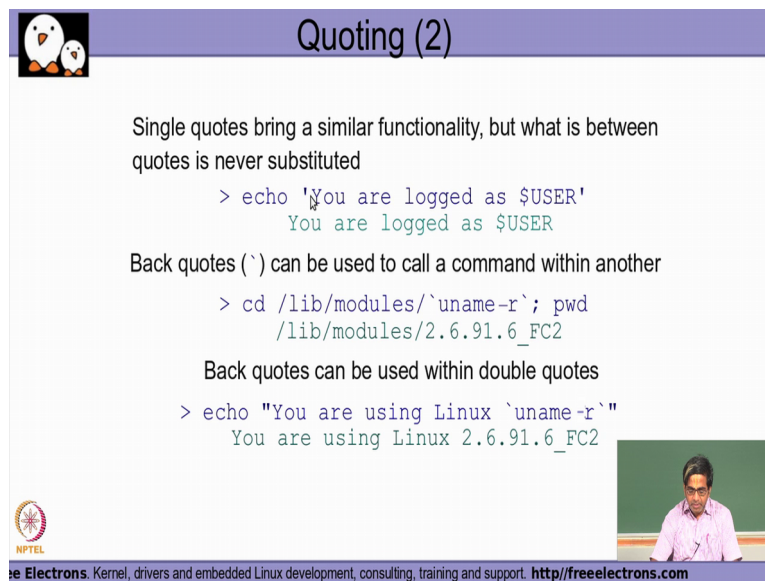
So if I, for example say echo and within double quotes we say any string including the space character because the fact that the space is actually been given within double quotes, the shell will not treat them as an independent argument but this whole thing is we actually treated as a single argument. Right .. ?

On the other hand and if we actually say use any kind of pattern expression like a Dollar user, so dollar user is basically stands for the value of a user environment variable when we actually use that within double quotes the environment variable will be getting replaced with

the value of the environment variable at that point in time and then appropriately made use of. So if you for example say echo within double quotes you are logged as Dollar user then it will be treated as your it will echo go back as you are logged as whatever is your current user ID. Right ? On the other hand if I don't use double quotes and I say star dot log, as we already seen in a previous module star refers to any number of characters followed by dot log. So in the filename pattern substitution, so it will now display me all the files in my current directory which actually has dot log (in its name) so it will display me all the files that is actually having dot log (in its name).

Now on the other hand if I don't want to do this file name pattern substitution for the star I could actually paraphrase it with double quotes and then if I do an echo of it we will see that it is actually displaying it as a normal string saying that it is star dot log itself without doing the pattern substitution for Star.

(Refer Slide Time: 02:33)



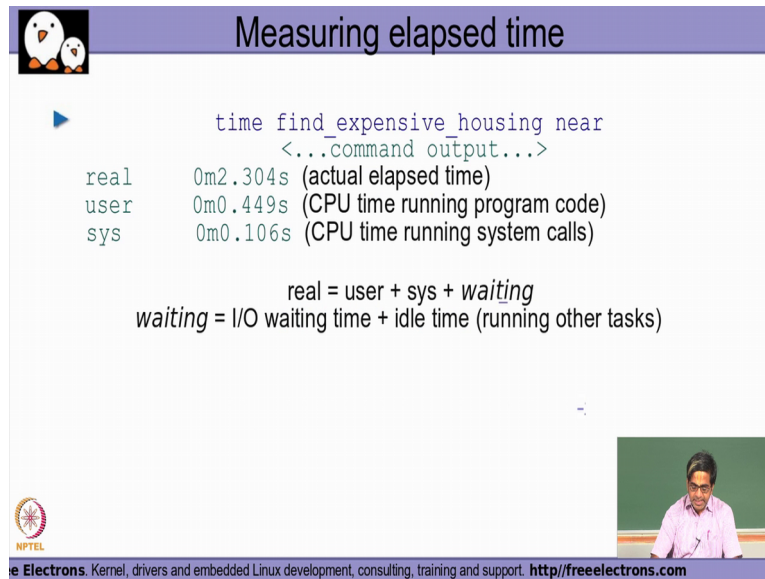
The slide is titled "Quoting (2)" and features a Linux penguin icon in the top left corner. It contains three sections of text and terminal output:

- Single quotes:** "Single quotes bring a similar functionality, but what is between quotes is never substituted".
Terminal output: `> echo 'You are logged as $USER'`
`You are logged as $USER`
- Back quotes:** "Back quotes (`) can be used to call a command within another".
Terminal output: `> cd /lib/modules/`uname-r`; pwd`
`/lib/modules/2.6.91.6_FC2`
- Back quotes within double quotes:** "Back quotes can be used within double quotes".
Terminal output: `> echo "You are using Linux `uname-r`"`
`You are using Linux 2.6.91.6_FC2`

The slide also includes an NPTEL logo in the bottom left, a small video inset of a man in the bottom right, and a footer with the text: "Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>"

Similarly single quotes bring out similar kind of functionality where the environment variables when you use it in a single quote are not treated for expanding the corresponding values to the environment variables but the environment variables even with the dollar characters will be taken as literal constant character strings.

(Refer Slide Time: 02:55)



Measuring elapsed time

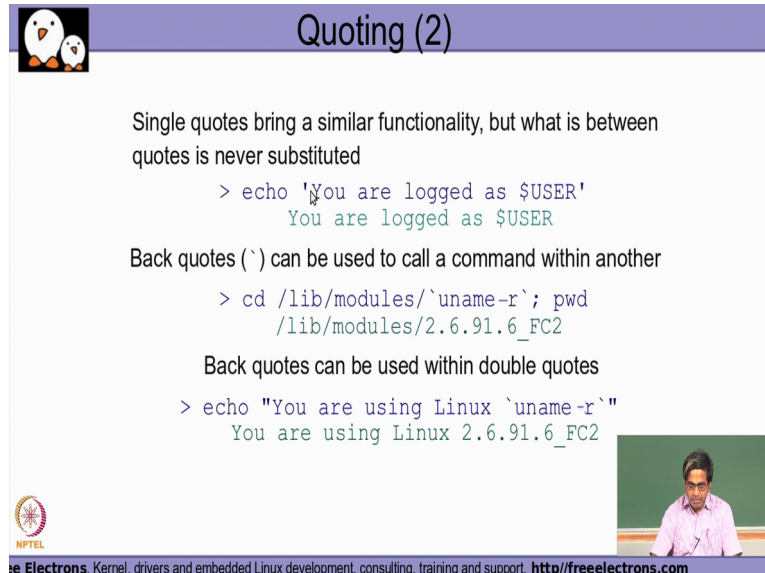
```
time find_expensive_housing near
<...command output...>
real    0m2.304s (actual elapsed time)
user    0m0.449s (CPU time running program code)
sys     0m0.106s (CPU time running system calls)
```

real = user + sys + *waiting*
waiting = I/O waiting time + idle time (running other tasks)

NPTEL
Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So in this case you actually looked at --- you are logged as dollar user when you put it within single quotes.

(Refer Slide Time: 03:00)



Quoting (2)

Single quotes bring a similar functionality, but what is between quotes is never substituted

```
> echo 'You are logged as $USER'
You are logged as $USER
```

Back quotes (`) can be used to call a command within another

```
> cd /lib/modules/`uname-r`; pwd
/lib/modules/2.6.91.6_FC2
```

Back quotes can be used within double quotes

```
> echo "You are using Linux `uname-r`"
You are using Linux 2.6.91.6_FC2
```

NPTEL
Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

It will literally take dollar user as a literal string and then print it as a test without reading user to be in environment variable and put the value of the user environment variable in this particular location which is basically what happens when you use double quotes in your shell command line. Now there is something called as a back code also that is available typically in

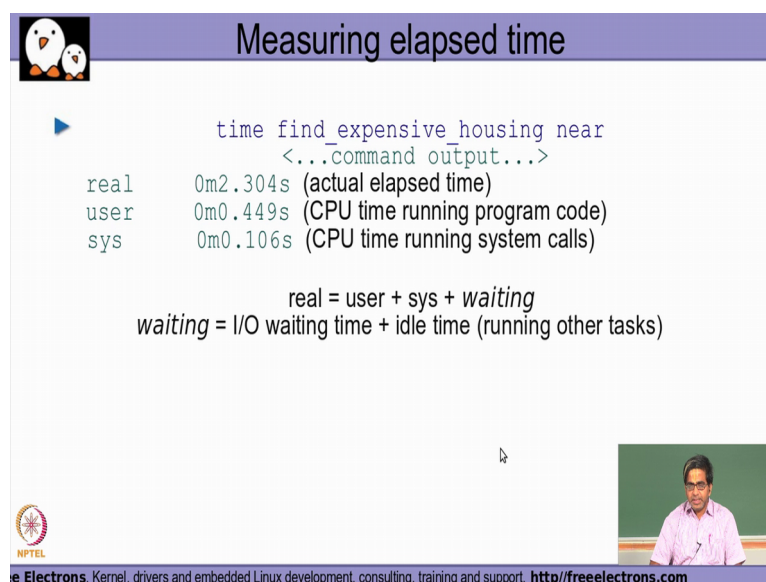
your standard keyboard. This back code character will be available in the same key as you have drilled a character.

So what happens is when whatever has actually been put inside the back quotes.... Right..? starting with a back quote and ending with a back quote will be treated as a command, so if I say `uname minus R` at particular command will actually be run and the output of the command will be replaced in a particular location.

So for example if I try to say `CD slash lib slash modules slash` and within back quote we are running a command called `uname minus R`, the `uname minus R` command will be run so that it will be expected to be a valid command that will be run and the output of the command will be replaced here as part of the shell expansion and then the entire command will be done.

So if you try to do like this and if this is basically your OS version that is currently running then when you try to put `unnamd minus R` within single quotes in this form, it will try to change to the particular directory and the system will expect that there is a directory like `2 dot 6 dot 91 dot 6 underscore fc2` available and so back code can also be used within double quotes where it will also be getting expanded like in a place where you are using the back quote without a double quote.

(Refer Slide Time: 04:57)



Measuring elapsed time

```
time find_expensive_housing near  
<...command output...>  
real    0m2.304s (actual elapsed time)  
user    0m0.449s (CPU time running program code)  
sys     0m0.106s (CPU time running system calls)
```

$real = user + sys + waiting$
 $waiting = I/O\ waiting\ time + idle\ time\ (running\ other\ tasks)$

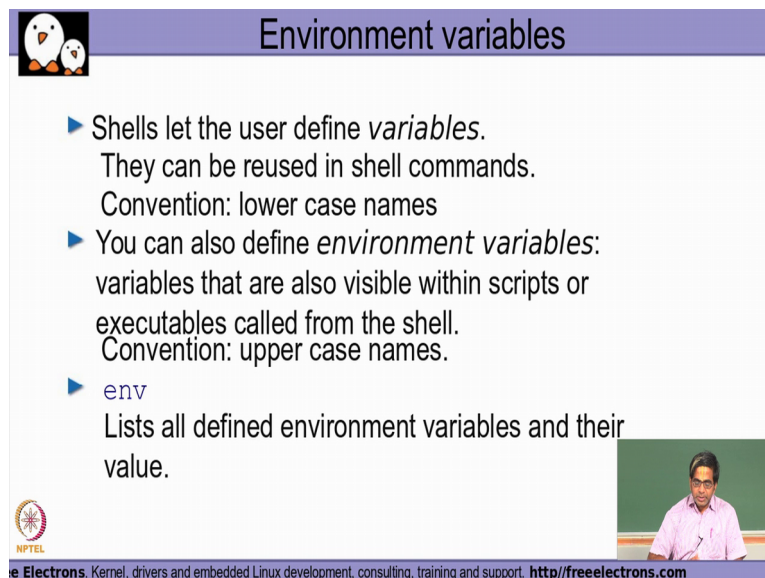
Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So another way of finding the time that is typically used for running a command will be to use a `time` command in which it basically gives you the different types of time that has been spent, how much of real which is actually the elapsed time that has happened and what has

been the component of the time that has actually been spending running the user code and what component of the total real time spent in running the system code and the balance of the time as compared to the total real time will be the time that it is actually spent in waiting.

So the waiting here could be either this particular task itself has been waiting for its IO to complete, or it has been actually idling around because there has been another task that has been given the processor because of which it has been put to sleep waiting for its turn of CPU to be given to it. So time is basically a command that could that you could use to get how much of time it has actually spent in these different environment and for you to get an idea on the kind of performance that your application is actually encountering on a typical running system.

(Refer Slide Time: 06:11)



Environment variables

- ▶ Shells let the user define *variables*. They can be reused in shell commands. Convention: lower case names
- ▶ You can also define *environment variables*: variables that are also visible within scripts or executables called from the shell. Convention: upper case names.
- ▶ `env` Lists all defined environment variables and their value.

NPTEL

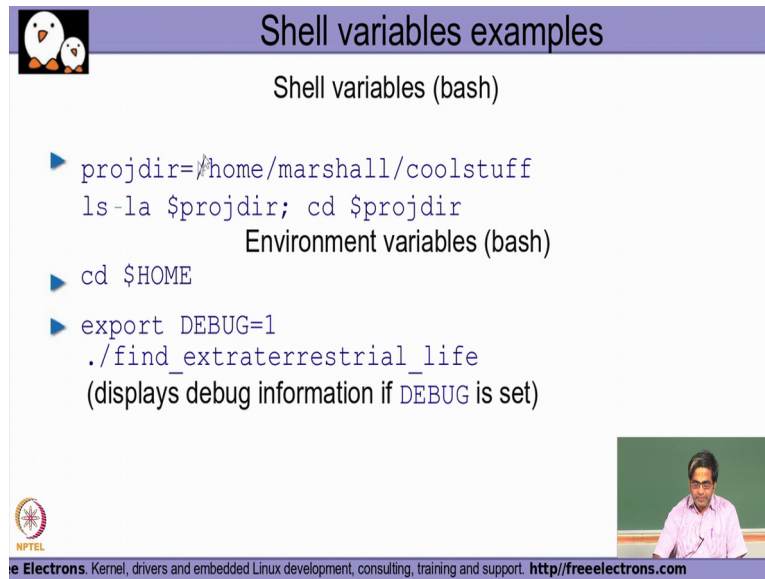
Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So there are environment variables actually available in the Shell. These are very handy whenever we are actually required to make use of certain standard locations as part of our command line in locations or as part of a Shell Scripts. So shells basically let the user define their own variables and also there are standard set of environment variable that are actually available which is provided by the shell in a default manner.

So these get initialised automatically by the shell process when it starts. So all the user defined shell variables by convention are (dour) names always in lowercase and by convention on the environment variables in Shell provide default manner are all in the upper case.

So this is not a hard and fast rule but the standard general convention that is followed to ensure that we basically get a very good quick hang of what is the variable that we are referring to whenever we are going through a large script or for easy readability purposes. So there is a command also called `env` that actually displays the currently defined environment variables and their independent values.

(Refer Slide Time: 07:33)



The slide is titled "Shell variables examples" and features a penguin icon in the top left corner. It is divided into two sections: "Shell variables (bash)" and "Environment variables (bash)".

Shell variables (bash)

- ▶ `projdir=/home/marshall/coolstuff`
- ▶ `ls-la $projdir; cd $projdir`

Environment variables (bash)

- ▶ `cd $HOME`
- ▶ `export DEBUG=1`
- ▶ `./find_extraterrestrial_life`
(displays debug information if `DEBUG` is set)

The slide also includes an NPTEL logo in the bottom left and a small video inset of a speaker in the bottom right. The footer text reads: "Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>"

So this is an example of user defined environment variables that I could initialise to wherein I won't have a shell variable called project projdir equal to some value. So the equal to is basically the differentiator between the variable name and the variable value for this is something like my name value pair with the equal to sign here as a delimited between the two of them.

So I want to have a shell variable, my own user defined shell variable called projdir and I want to set the value of that shell variable to be this particular location and then I could actually use this shell variable that I have defined in any kind of a command like for example CD dollar project dir will actually change the directory location to this particular value that is there of this project dir environmental shell variable. Right ?.. On the other hand I could also use an environment variable a standard environment variable like home which actually denotes my HOME directory and which is automatically set whenever the user is actually logged in. Right ?

So the HOME is one of the environment variables it is set automatically by the bash shell when the user is logging in pointing to the user's HOME directory. So whichever user is actually logging in pointing to the home directory of that particular user this particular environment variable will be set and then you may say cd Dollar HOME it will actually take me to the home directory of the currently logged in user irrespective of my current location.

(Refer Slide Time: 09:10)

Main standard environment variables

Used by lots of applications!

- `LD_LIBRARY_PATH`: Shared library search path
- `DISPLAY`: Screen id to display X (graphical) applications on.
- `EDITOR`: Default editor (vi, emacs...)
- `HOME`: Current user home directory
- `HOSTNAME`: Name of the local machine
- `MANPATH`: Manual page search path
- `PATH`: Command search path
- `PRINTER`: Default printer name
- `SHELL`: Current shell name
- `TERM`: Current terminal type
- `USER`: Current user

NPTEL
Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So likewise the different standard environment variable that is actually available so I could actually use an environment variable like EDITOR which is the variable that is actually used to denote what is the default editor that this particular user would like to make use of, whenever he has a requirement to open up a file and edit. Right ?

Similarly the HOSTNAME environment variable will be the environment variable that is used by the shell to denote what is the name of the local machine typically the HOSTNAME is actually used whenever any other kind of machine on the network needs to access this particular machine and the hostname is the name by which the other machine will come to know of the local machine's name. Right..?

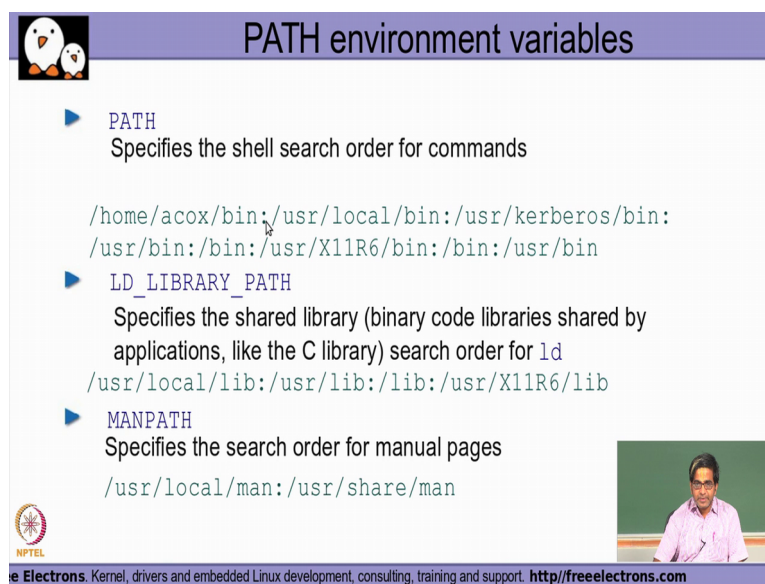
So, PATH is again another environment variable which is very commonly used, which is used to denote what is the PATH in which the command that has been typed by the user has to be searched, so when I type a command like, let's say PES, the SHELL needs to know the different directories in which the commands to possibly be present and the SHELL will try to go and search for this command in each of those directories that is given in the PATH environment variables.

So each of the directories in the PATH environment variable will be delimited by a colon character and it will try to search for this command in each of those directories and if the command is not found in any of the directories, finally it will report back to the user saying that this command is not found and wherever, whichever location this particular command is

found, the shell will just go ahead and read the contents of that particular command file that is basic instructions are available in the command file and then load it and start the process for that particular command execution.

So, Likewise we do have different types of environment variables for user is an environment variable that will be typically containing again by default this variable will get initialised as soon as the SHELL is started. It will contain the current user name who has right now logged into the system and for which this particular shell process as actually started.

(Refer Slide Time: 11:30)



PATH environment variables

- ▶ **PATH**
Specifies the shell search order for commands
`/home/abox/bin:/usr/local/bin:/usr/kerberos/bin:
/usr/bin:/bin:/usr/X11R6/bin:/bin:/usr/bin`
- ▶ **LD_LIBRARY_PATH**
Specifies the shared library (binary code libraries shared by applications, like the C library) search order for ld
`/usr/local/lib:/usr/lib:/lib:/usr/X11R6/lib`
- ▶ **MANPATH**
Specifies the search order for manual pages
`/usr/local/man:/usr/share/man`

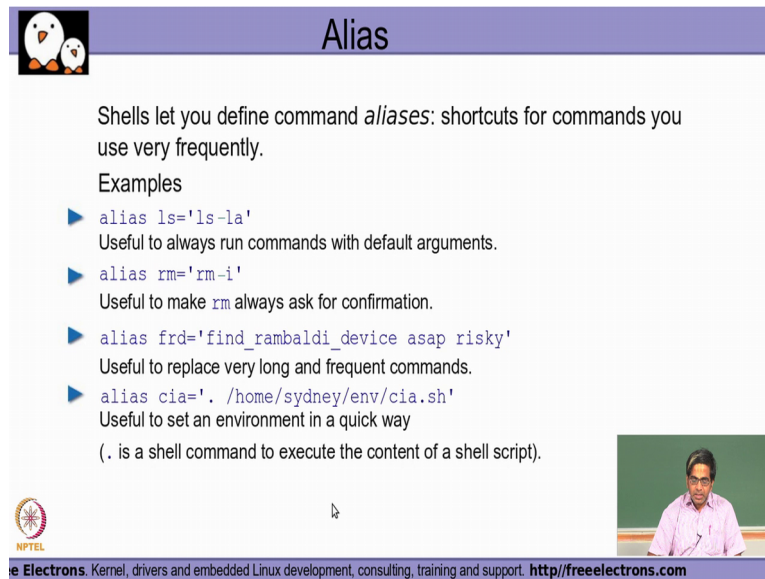
NPTEL
Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So this is how the path environment variable will look like, as we were discussing we will be de limited by colon character so whatever command is actually typed it will go first look at the first directory that is given, it will try to find out if the command is available there, if it is not there it will go and search in the second directory. Likewise it will keep continuing till the end and in whichever directory the command is found, it will just read the contents of that particular command executable file load into memory and then start it off as a Process.

Similarly LD LIBRARY PATH is actually used to find the PATH of a shared library whereas the PATH environment variable is used only for finding the PATH of the common commands or executable needs to be started. MANPATH similarly is used to find the whenever we are trying to run the MAN command for any specific tool or command that we want to get help for, it will go and search the MAN pages in the set of directories that is initialized as part of the MANPATH. So, in this particular example go and search for the MAN page in this

particular first directory mentioned here and then it will go and search if it is not found in the first directory, it will go and search for the MAN page in the second directory that is actually mentioned there.

(Refer Slide Time: 12:50)



Alias

Shells let you define command *aliases*: shortcuts for commands you use very frequently.

Examples

- ▶ `alias ls='ls-la'`
Useful to always run commands with default arguments.
- ▶ `alias rm='rm-i'`
Useful to make `rm` always ask for confirmation.
- ▶ `alias frd='find_rambaldi_device asap risky'`
Useful to replace very long and frequent commands.
- ▶ `alias cia='. /home/sydney/env/cia.sh'`
Useful to set an environment in a quick way
(. is a shell command to execute the content of a shell script).

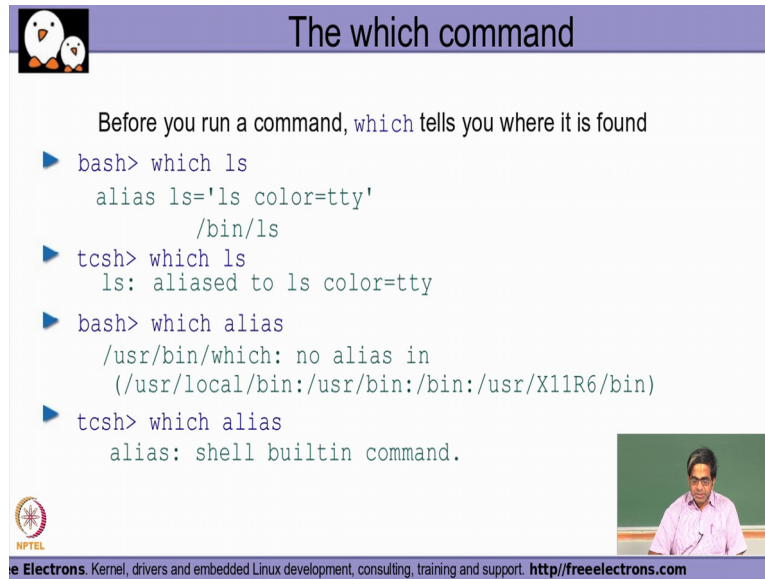
NPTEL

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So, alias is basically a mechanism that is available in the Shell again to have a shortcut for command. So if I have a very common command that I keep executing and the command actually consists of a big sequence of characters which I want to have only a very short form for it, then I can use the alias command wherein I can set alias ls equal to whatever I want to set the task. Right?

So in that way if I set alias frd is equal to this and whenever I want to run this big command, if I just type the frd letters alone as part of my command, the command line SHELL will automatically expand the alias to whatever it has been said to hear and then run that as the command. Right? So in a way it basically sort of simplifies how quickly I am able to give my input to the shell to make it run my command and also subsequently get the output that much more quickly.

(Refer Slide Time: 13:55)



The which command

Before you run a command, `which` tells you where it is found

- ▶ `bash> which ls`
alias ls='ls color=tty'
/bin/ls
- ▶ `tcsh> which ls`
ls: aliased to ls color=tty
- ▶ `bash> which alias`
/usr/bin/which: no alias in
(/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin)
- ▶ `tcsh> which alias`
alias: shell builtin command.

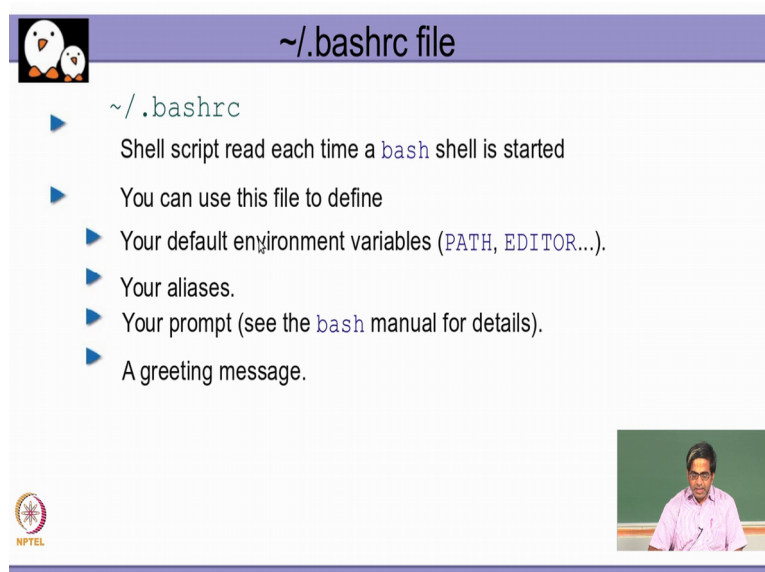
NPTEL

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

There is a command called `which`, which also used to find out where is it that my command is actually available. Right..? Now, if I basically say `which LS`, it lists back telling me if that all there is any alias for a less what is it that is getting expanded to and apart from that if there is a command call alias also separately available.



So if my command `LS` is typically available in slash bin so you find that the command output for this `which` is displaying me whatever is available for this particular argument in my alias as well as in my path environment variable. So there is a slash bin directory is available in my path environment variable and since `LS` command is available under slash bin, it is also displaying displaying that here.

(Refer Slide Time: 14:50)



~/.bashrc file

- ▶ `~/.bashrc`
Shell script read each time a `bash` shell is started
- ▶ You can use this file to define
 - ▶ Your default environment variables (`PATH`, `EDITOR`...).
 - ▶ Your aliases.
 - ▶ Your prompt (see the `bash` manual for details).
 - ▶ A greeting message.





So there is a file called dot bashrc like we have seen in earlier modules. Any names that is actually starting with a dot is a hidden file and the dot bashrc file available in a home directory of the user is a hidden file that contains all the different things that needs to be run as and when the shell process is actually started by step process actually started for the user. So whenever a bash shell process is actually started for a user the bash shell automatically goes and tries to find a file called dot bashrc In the users home directory and then execute whatever is there inside that file. So I could actually use this file for defining the different values I want for my default environment variables. I could set my own aliases. I could set my own prompt.

I could have my own greeting message and so on and so forth. So essentially whatever I want my shell to be initialised with, as and when I start of a new shell, I will put all the sequence of commands and initialisation as part of this dot bashrc file which is available in my home directory so whenever unusual process is started by me, the the shell automatically is programmed to go check out what is in my home directory under the home directory check for the existence of dot bashrc file and if that file is found, execute the file and because of which whatever customizations I have done inside that particular file, they all will be getting executed and initialised.

(Refer Slide Time: 16:30)

Command history (1)

- ▶ `history`
Displays the latest commands that you ran and their number. You can copy and paste command strings.
- ▶ You can recall the latest command:
`!!`
- ▶ You can recall a command by its number
`!1003`
- ▶ You can recall the latest command matching a starting string:
`!cat`



Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So if they want to find out what are the different commands it is actually been executed by the user currently, I could run the history command and it will display me the list of the previously executed commands wherein after that list is there if I just type double exclamation mark in the command line, it will just re re re run the last command it has been given to the Shell.

On the other hand if I want to run a command specifically by a number, Right, I could specify exclamation followed by the number or I want to run a command which is actually starting with a particular sequence, I could say exclamation followed by the sequence. So these are different ways by which I would be able to go back to the history of all the commands that I have actually run and appropriately select the command that I want to run right now.

So if I had basically typed a very long command, I don't want to really retype the command, long command again and thereby waste time, so you can consider this as like I kind of a shortcut mechanism where I can quickly supply to my Shell what is the command that I intend to run right now, instead of typing in the command one by one, character by character, especially when the command is a very long one.

Thank you