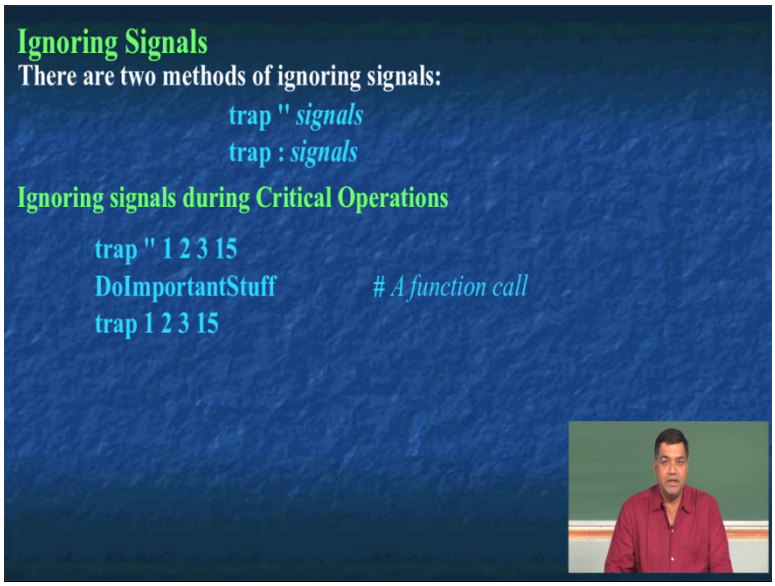
Information Security 3
Sri M J Shankar Raman,
Consultant Department of Computer Science and Engineering,
Indian Institute of Technology Madras
Module 41
Shell Subshell

We saw that a signal can be captured by a shell program and then a corresponding service routine can be executed; now we also saw how to handle these signals using the trap command. So we saw that these signals can be ignored by putting a return statement and there's also another formal way of doing it, ok which we'll see in this session.

(Refer Slide Time: 00:48)



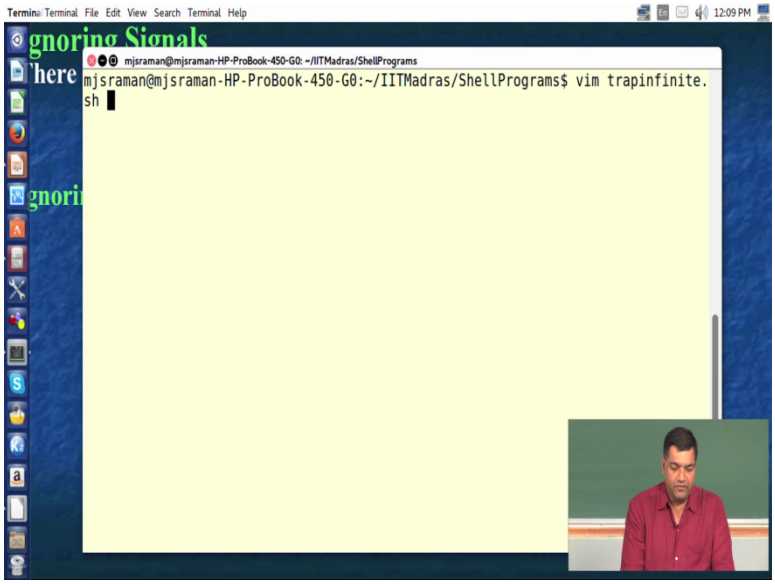
So there are two methods of ignoring signals the, actually we'll consider just three methods ok, the first method we saw by that in the last session that by putting a return statement in the signal handling routine, we'll be able to take action on the signal or we can just ignore it. So the other ways of doing it is something like this you can put a trap and then give a null and then give the list of signals for which you don't want any action to be taken or you can put a colon symbol after the trap command and then ignore whatever signals you want to ignore.

So why do we need to ignore signals? So as if discussed in the previous class if you are doing certain critical operations ok for example you are copying one file from one location to another

location and you do not want a user to interrupt while you are copying the files between these two locations so in that case what you can do is, you can disable all the signals from the particular for the user. So like you put trap space colon space and then you give all the signal numbers because of which the any signal that is generated to unsent to this process will actually be ignored by the process. Now here is an example, ok so in this case what we are trying to do is we are trying to ignore signals 1 2 3 and 15 ok using the trap and then using the first syntax and then we are trying to do.

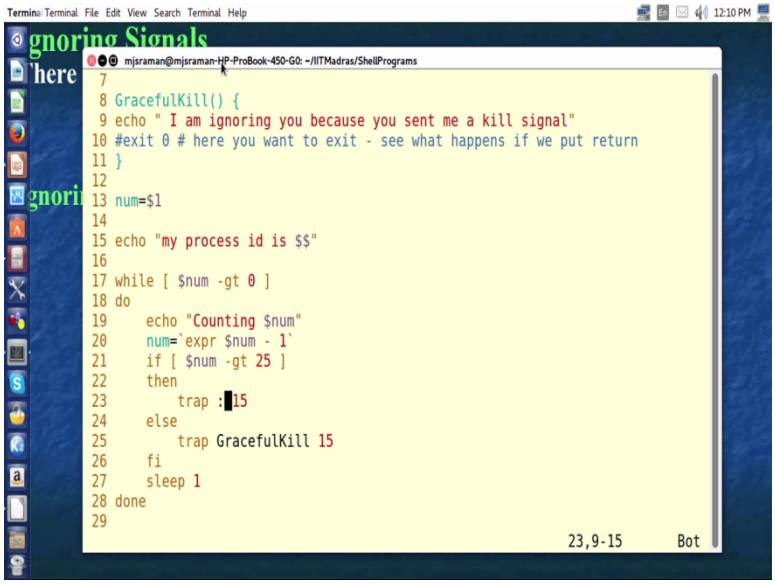
Whatever important stuff you want to do in the shell scripting and then finally we are just then enabling these trap again back by putting trap and then you can put the function name or you can just put trap and then you can give these numbers and so because of which you will be able to enable the signals back, now let us look at it with an example, we'll take the same example what we had done last time, we'll take a minor change the code and then we'll see how this trap signal can be enable as well as disable.

(Refer Slide Time: 02:52)



So let us go to this program called trap infinite.sh.

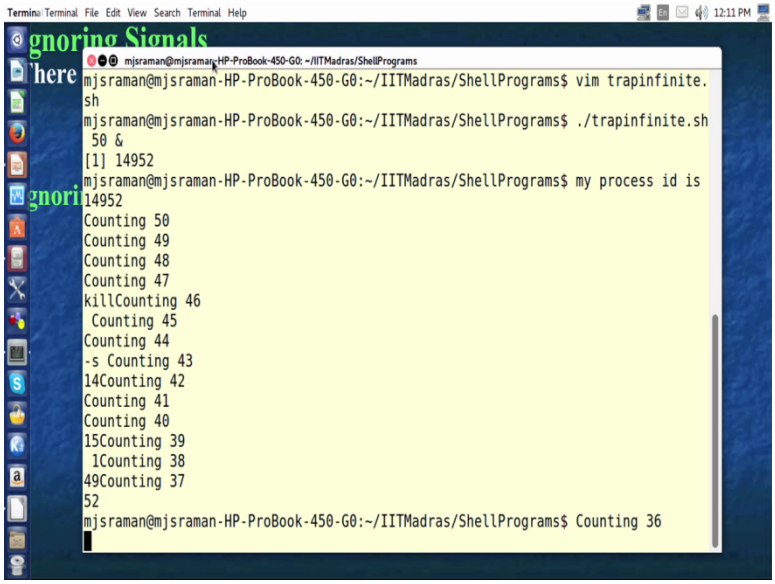
(Refer Slide Time: 02:58)



So this program is the same if you see I am not going to repeat this again the only change that I have done to this program is the following line, see I've added an if statement which tells you that when we are counting down in this program until you count your count is greater than 26 if I sent signal number 15 you just have to ignore it, so if you look at line number 23, I am just saying that if I send signal number 15 then use this trap command to ignore these signals.

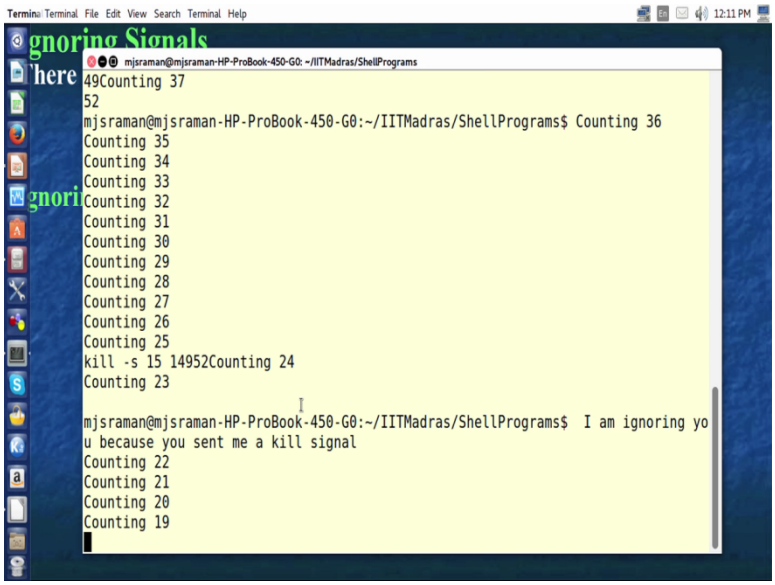
So what is going to happen is until the count is 26 if you sent the kill space minus s then fifteen and then this process id the kill command, the kill I mean signal fifteen this process will simply not bother about it, then after the number is less than 25 what happens is that here in this case, now whenever I type 15 ok signal number 15 it will now called this process, so this is going to plenty I am ignoring you because you send me a kill signal ok, so something like this I'll do, but in this case let us also add the return statements so we know this guy is getting ignore.

(Refer Slide Time: 04:20)



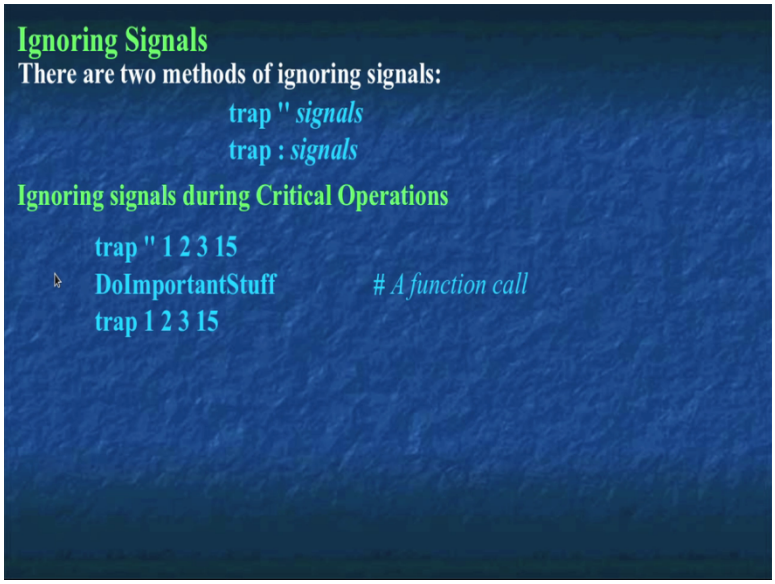
So now let us run this program ok. So trap infinite.sh and then we have to give a number, so let me give a number like say 50 and then I'll run it in the background, now I got the process id 14952, so I'll now try to kill minus s 14 15 14952 ok so nothing is happening, because until if you remember until 25 this guy will ignore whatever you send.

(Refer Slide Time: 04:58)



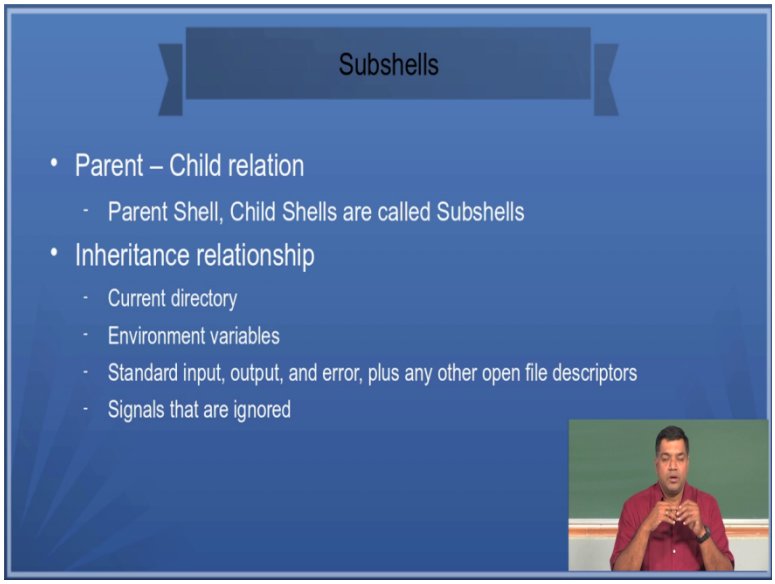
Now let the count be less than 25, so now it has become 25, so now I'll try to send the signal. So now it says I am ignoring you because you send me a kill signals.

(Refer Slide Time: 05:29)



Now let's move on to another facility of shell scripting, so what we'll do is

(Refer Slide Time: 05:38)

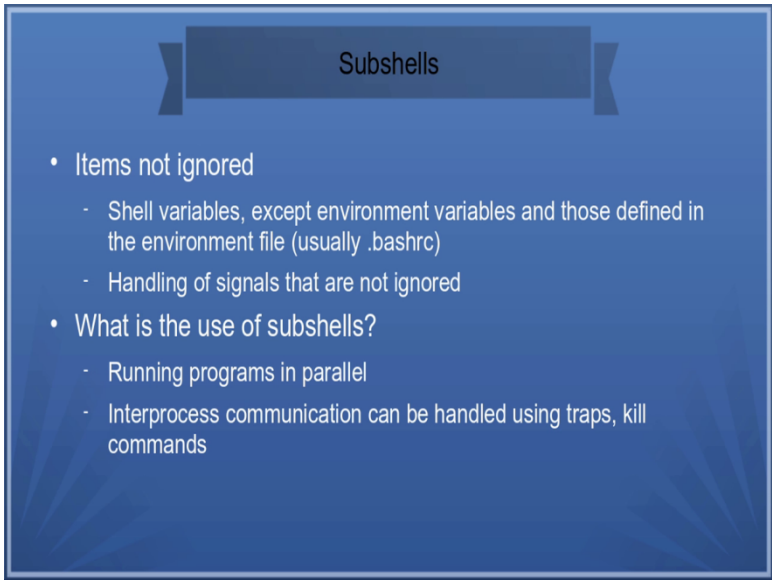


We'll now talk about sub shells, So essentially until now we've been handling one shell ok, now the question is how do I create other shell within a shell, so in this case what it is called as a sub

shell ok, so a sub shell actually has a parent child relationship with its parent shell, ok? And a parent can create many sub shells inside itself and there is an inheritance friend's relationship that comes between the parent shell and the sub shell that you are creating. So in a inheritance relationship what are all the the the elements that are carried over from the parent to the child, so in our shells and sub shells the inheritance relationship carries over one the current directory where the program is getting executed.

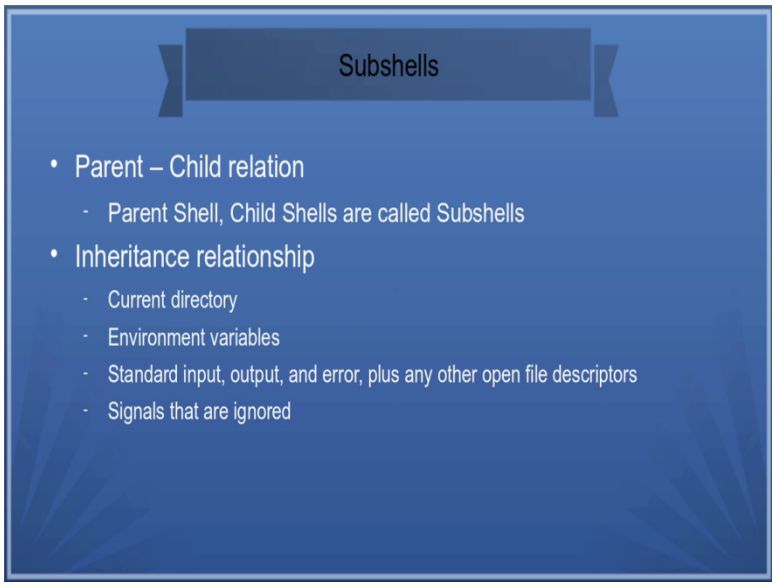
And then all the environmental variables that was supplied by the parent to the current sub shell then you have any opened file scriptures either than standard input standard output and errors they are by default they are just passed on to the child and one of the most important things is the signals that are ignored are also caused on to the child, ok so in the previous session, the session before we just found out how to handle the signals and even the current sessions few minutes back we discussed how signals can be ignored.

(Refer Slide Time: 07:18)



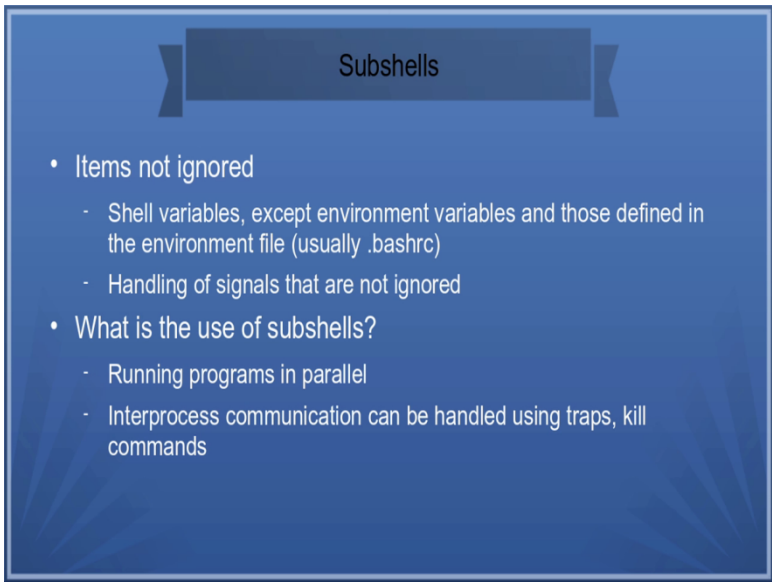
Now the items that are not ignored so let us try to find out the items that are not ignored ok the first item that are not ignored are the shell variables ok so except environmental variables other shell variables are not ignored ok and so the I handling of signals that are not ignored so you'll have to find out

(Refer Slide Time: 07:39)



What is all the inheritance that gets and that are not got.

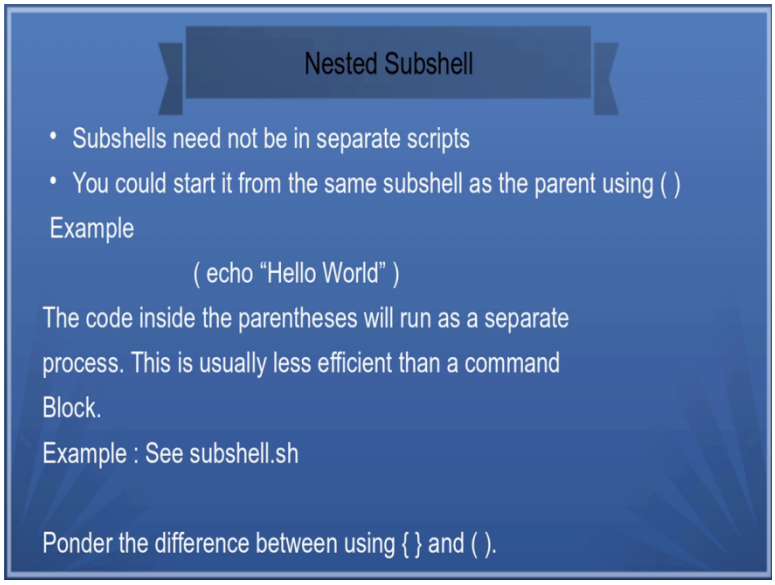
(Refer Slide Time: 07:43)



So into handling of signals that are not ignored are also not got ok. Now what is the use of a sub shells? Ok what are the advantages of using a sub shell is that you can run programs in parallel. So what I can do is I can I can go on executing a shell script and then I can (())(07:58) some sub shells and then they can be doing their job and then I can also have some sorts of inter process

communication that can be handle between so I think we discussed about it a lot in previous section, ok?

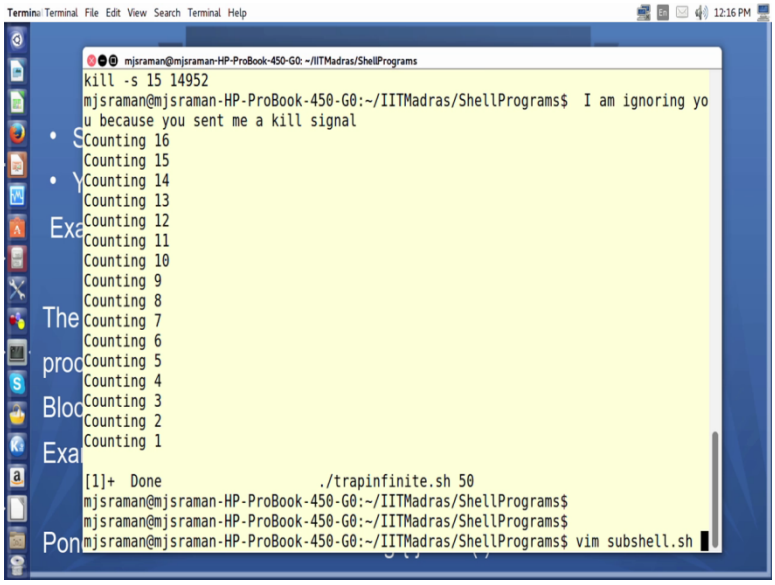
(Refer Slide Time: 08:11)



So one of the things that that that we need to understand about a sub shells is that you need not write a separate script for sub shell, so actually if you want to create a sub shell within a parent shell you essentially what you have to do is you've to put a brace like this a curved brace. So here is an example so when I say echo hello world what it actually does is that this hello world program actually runs within a shell, within the from the parent shell,

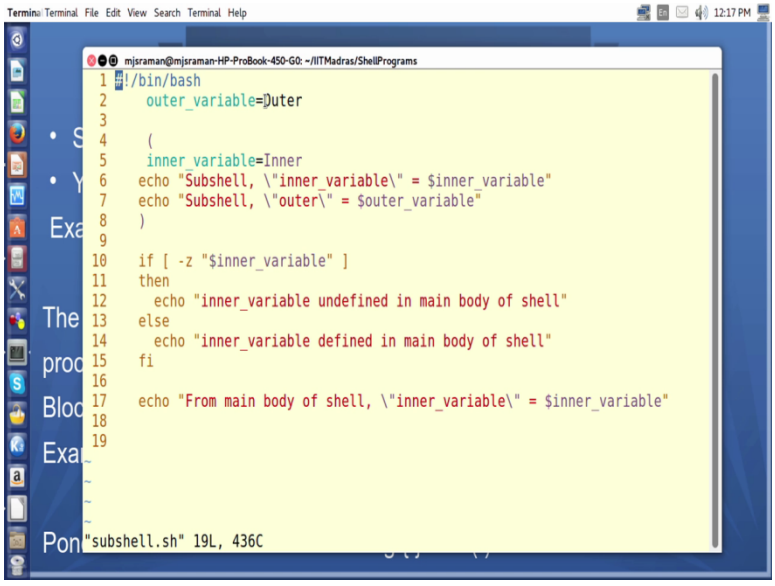
So so the code inside the parent is this will run as a separate process. This is usually less efficient than a command block ok, so if you remember the command block we had used the symbol ok so the one that is at the bottom the curly braces so that what was is used as a command block, ok and for a sub shell we use this, in a command block the the process runs in the id of the parent shell itself but in a sub shell it actually creates a new process ok, so let's take a look at the example of sub shell.sh and then see how we can do it ok.

(Refer Slide Time: 09:19)



So here is sub shell.sh.

(Refer Slide Time: 09:22)



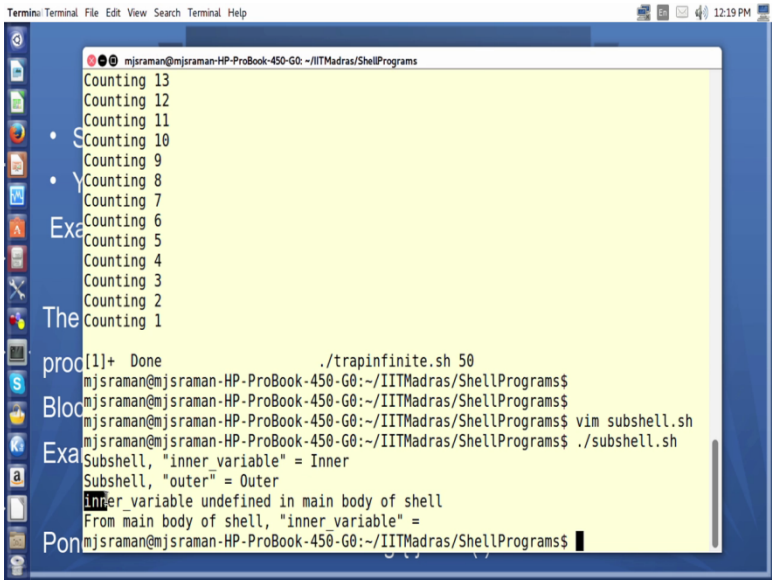
So let's try to understand this program first ok. So wontedly I have not formatted the program so I just wanted to show you that ok you can write a nice looking shell program as well as a program that looks like this, ok which looks slightly difficult to understand because the indentation is bad and things like that, ok. So let us try to understand this program so usual line

number 1 tells you that you've to use the bash shell and then we come to line number 2 where you declare a variable called an outer variable and then initialize it to the value called outer, ok I think this is straight forward. Now look at these lines 4 5 6 7 and 8, what these lines tells you is that I have to create an inner sub shell so this is what it says by putting this, so inner sub shell and then inside this inner sub shell I just declare a variable called inner variable and I put the value of inner to it and then what I do is I echo inside the sub shell.

I put this inner variable and then find out the value of inner variable so if dollar inner variable should actually print inner Inner and then the cap the, I am also trying to print the value of the outer variable to find out whether the outer variable is actually passed because this guy is going to be executed in a separate process id I just want to see whether this value is getting passed to the sub shell ok and then the parent shell goes on executing this if it finds out that the inner variable is null then it says that inner variable is undefined ok.

In the main body of the shell and it otherwise it says inner variable is defined as the main body of the shell because we just have to find out whether this inner variable is getting exported to the out parent ok, so we saw that inheritance whatever is the parent goes to the child but whether the one from the child that goes to the parent so we need to find out right, and then so after this I say from main body or shell I am trying to print the inner variable, so let us see what happens because of the usage of this sub shell.

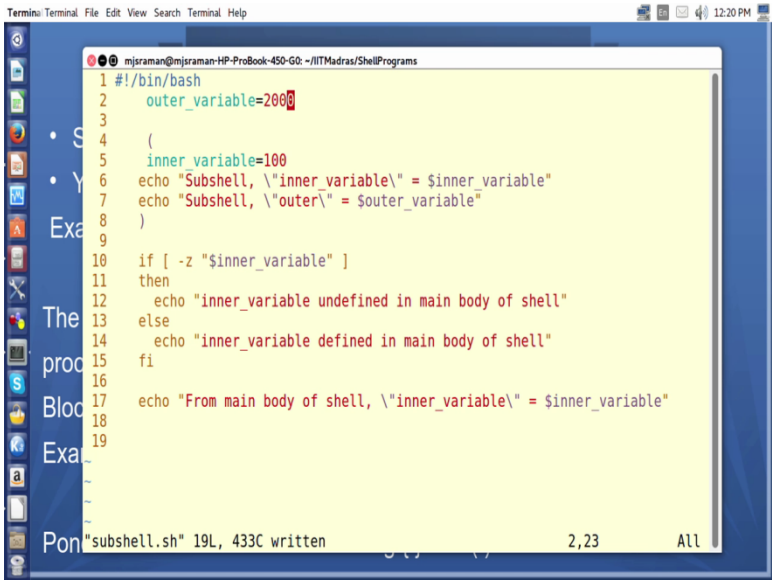
(Refer Slide Time: 11:40)



So let's run this program and try to understand, so what I am trying to do is I am trying to execute this program called subshell.sh. So if you look at this ok the results is very surprising, but as we had discussed for us it is not surprising so what it says is that if you, if the sub shell ok is says the inner variable is actually defined so I am getting inner.

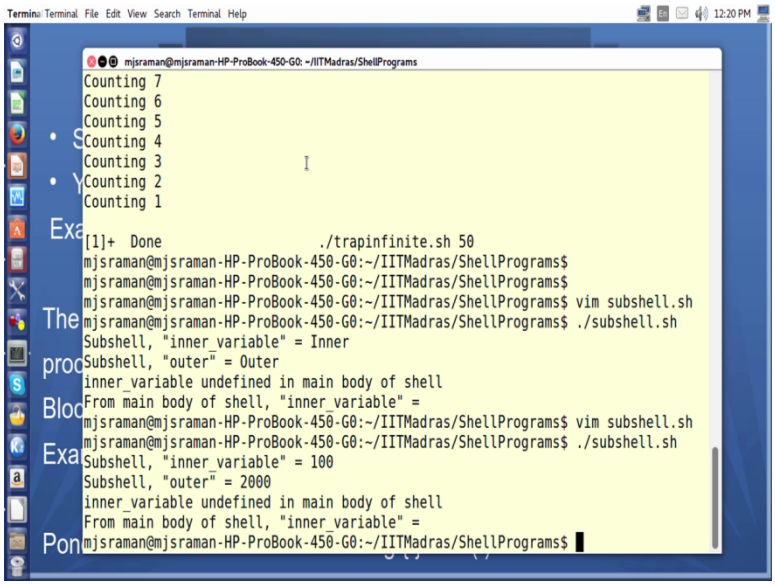
Similarly because the parent is calling the sub shell, it is passing its variable to the child, therefore the sub shell outer variable is also defined ok. Now we are looking at the execution of the parent, the parent says that the inner variable is an undefined the main body of the shell, but in the main body of the shell, there is no variable called inner variable and the value is undefined. So if you look at the program again you'll understand what is happening.

(Refer Slide Time: 12:31)



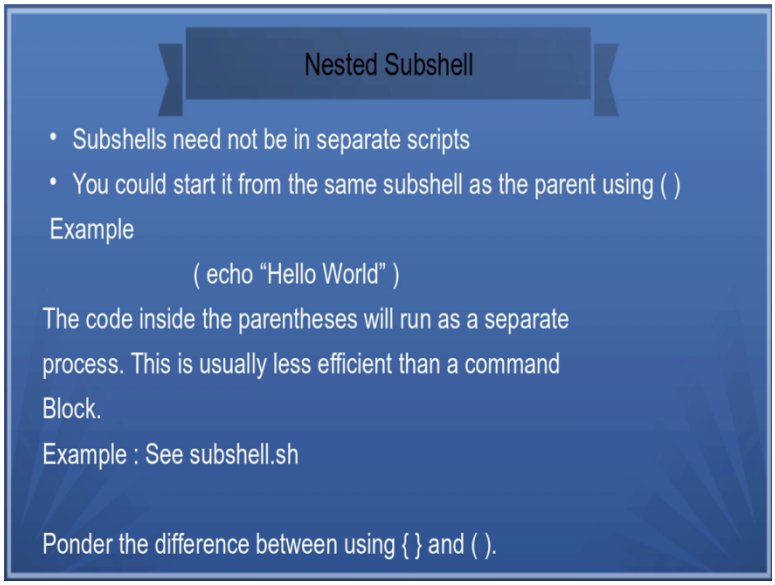
So if you look at this we see that this inner variable so let me instead of saying inner variable outer why don't you substitute some values and see, so I'll say I'll give a value of 100 to this ok and let me give a value of, I'll give a value of 100 to this then the outer variable I'll give a value of 2000 so you can see that the value of 2000 gets passed to the child, but the value of 100 does not get passed to the parent. So if you run this program again.

(Refer Slide Time: 13:05)



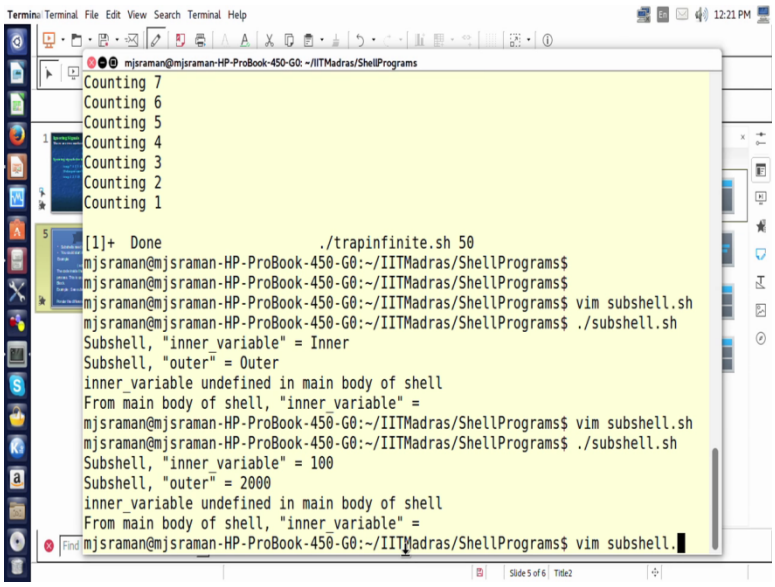
So if you look at this the sub shell gets a value of 100 as well as 2000 that is the parent shell does not get's the value of of 100, ok so in this way we are able to see that a sub shell gets created and we can use sub shells.

(Refer Slide Time: 13:25)



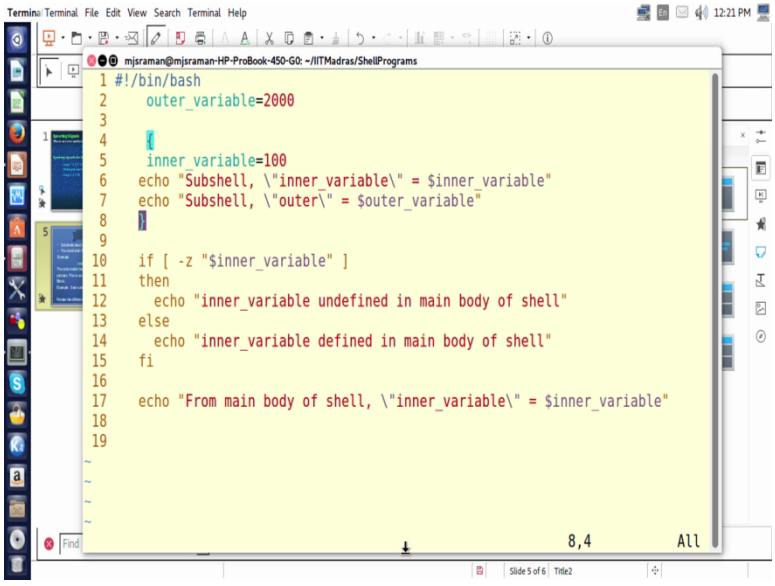
Now as I told you I leave it as an exercise for you to identify the difference between using this parenthesis and using this brace. So what you do is you spend about two minutes stop the video, spend about two minutes and try to understand what is going to be the effect and probably after you switch on we'll again see what happens if we replace this brace with this brace in the previous program. So please pause for about two minutes, try to understand what we'll be the result, then we'll run the program again.

(Refer Slide Time: 14:02)



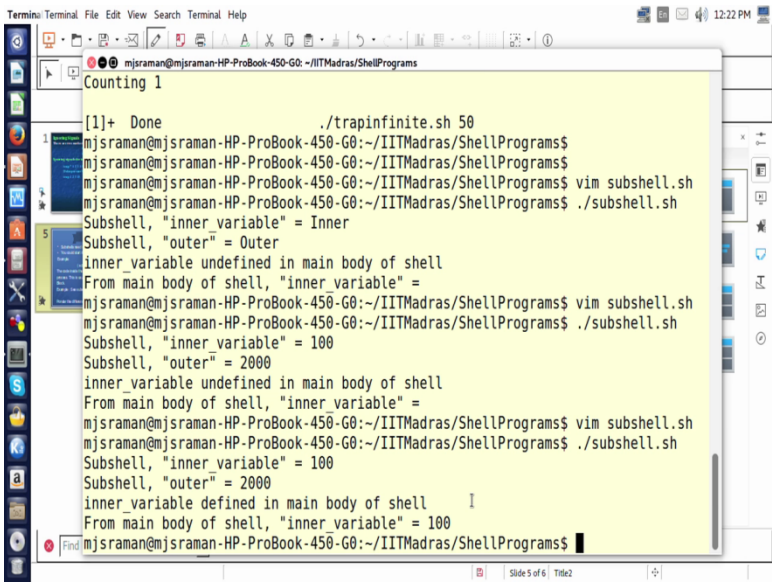
Hi, hope you did that so let us try to make the small change to that program ok.

(Refer Slide Time: 14:07)



Whereby I replace this brace with this brace what is happening is so I am just change the braces like this,

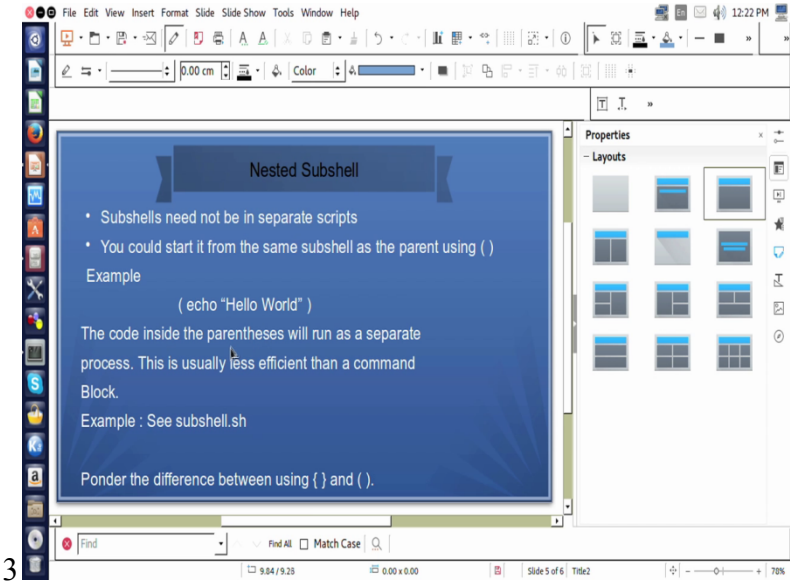
(Refer Slide Time: 14:16)



I am running this program, now if you look at this the difference between the previous execution and the current execution, since I put a curly brace and then once I put a curly brace the whole thing becomes a block ok. So it's becomes a block of code and since it is only a block of code, I am able to get the value of the variable ok the inner variable also inside the code because essentially it's only a single process that is running and it's a single shell program.

Whereas when I just change it to the curved braces we actually see that a new process is created and because of which the variable is not getting exported and the parent program is not able to able to see the value of the inner variable, so this is the major difference between using block so that's exactly what we've also defined here,

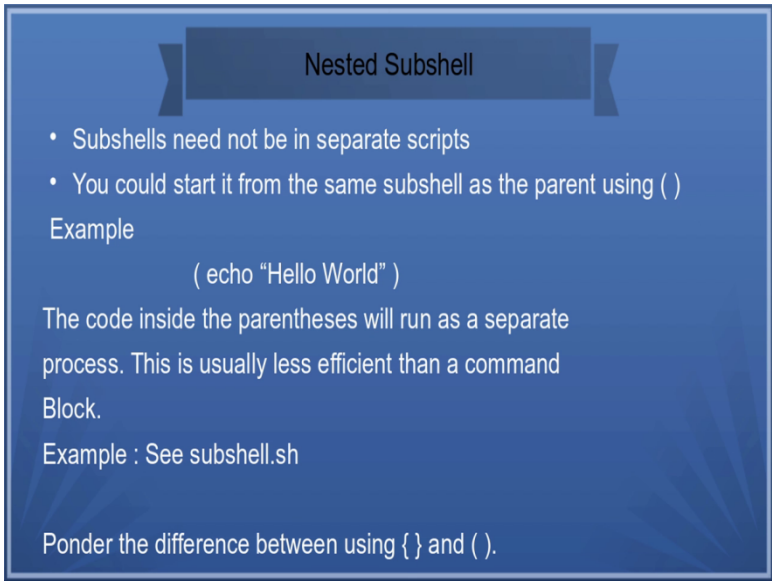
(Refer Slide Time: 15:11)



3

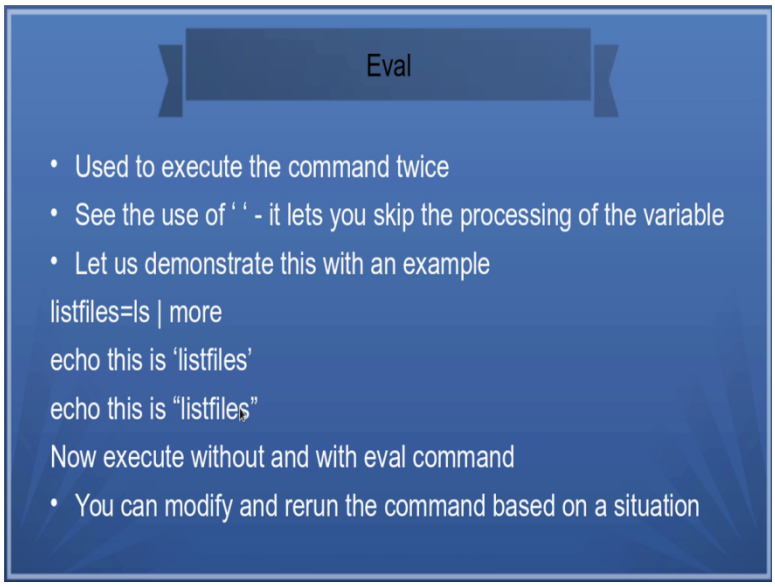
So we've just explained it like that stating that so if you look at this, the code inside the parenthesis will run as a separate process ok.

(Refer Slide Time: 15:17)



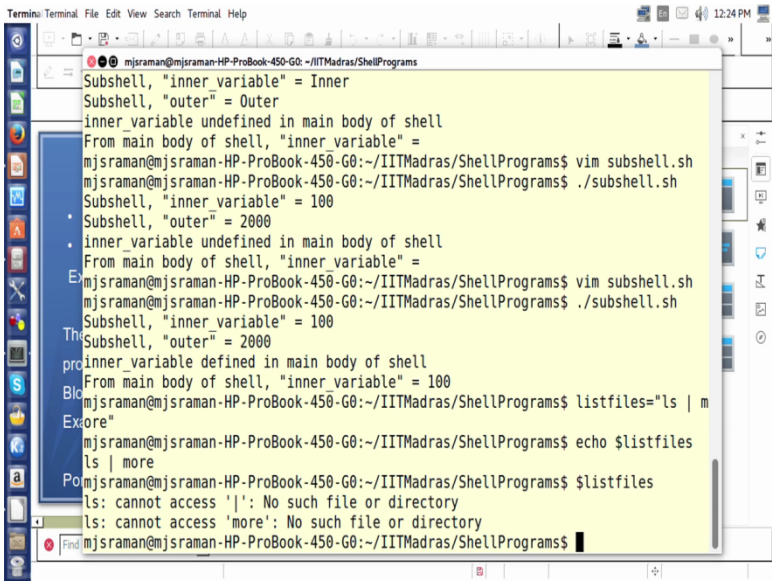
Whereas if you put this kind of curly brace then it becomes a command block and obviously since you've to spont a new process ok it's slightly less efficient than the command block.

(Refer Slide Time: 15:37)



So what we will do is we will now go forward to a new command that we can use this is slightly different from what we've been using in the previous stage ok so Eval is a command that lets you executes the same command the command twice, ok so why do we need this we already know that you can skipped expansion of variables or commands just by putting this single code ok,

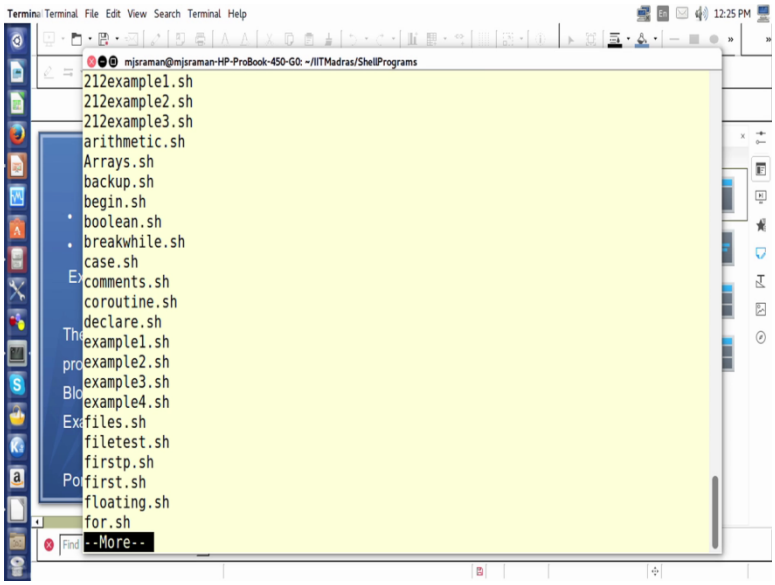
(Refer Slide Time: 16:11)



Let us demonstrate that with an example, ok so I will go ahead and say list files is equal to let me put it as ls and then I'll say more ok, and I say echo dollar list files. So it essentially tells you that echo dollar list files now suppose I just want to run this, let's see what happens if you want to run this ok.

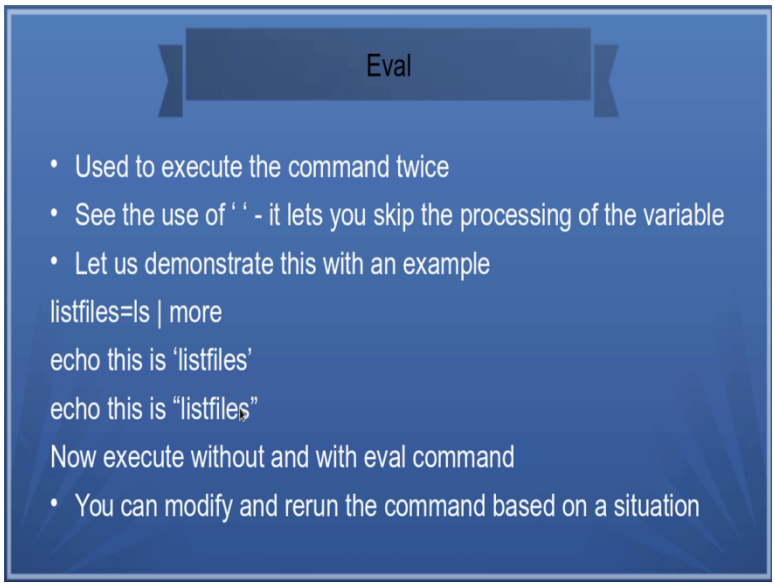
So I say dollar list files ok so what happens is that this variable gets expanded as ls ok command and then what happens with ls command is that the ls command expects some kind of a argument ok but the argument that you are passing is a pipe symbol ok and then it has got another argument called more, so because of which the command does not know what to do, so the ls says that ok there is no such directory called pipe and then there is no such directory called more.

(Refer Slide Time: 17:22)



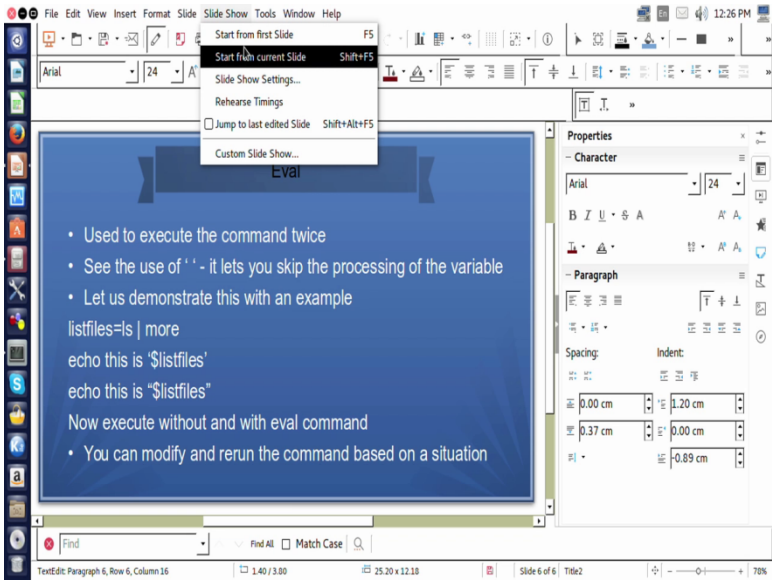
So what we can do now is that now suppose I want to run this ls command what I will do is I'll just say Eval dollar list files. So if you look at this when I say Eval dollar list files

(Refer Slide Time: 17:38)



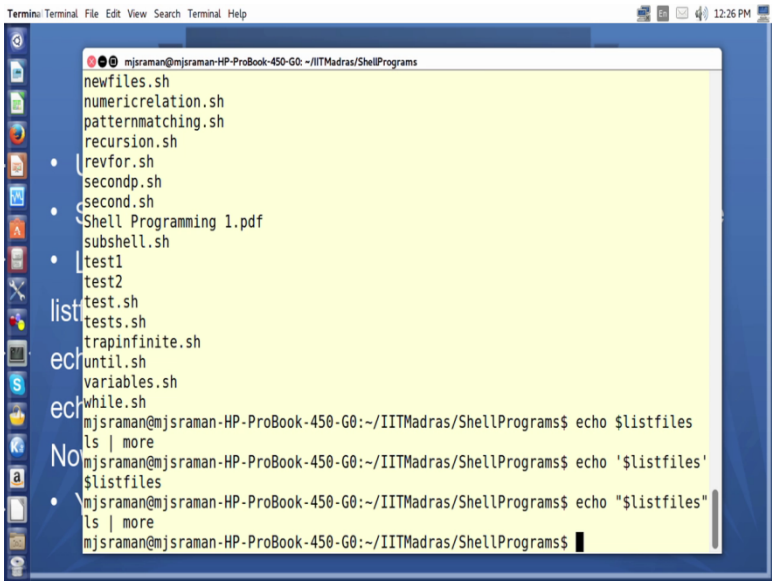
So what exactly is happening is that, If I say this list file so if I say list files is equal to ls more and when I say echo this is list files with a single code. So what happens is that it does not expand so actually it should be a dollar.

(Refer Slide Time: 17:58)



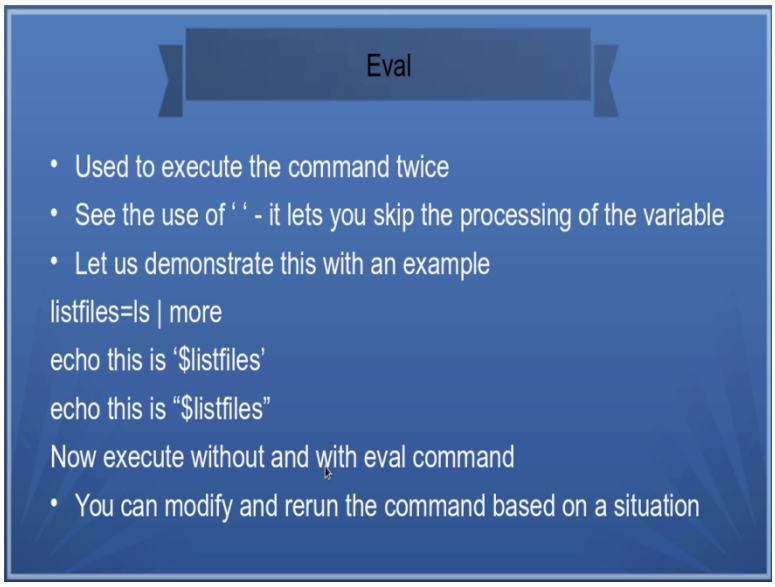
so let me now Make a so the there is an error here, so what I should do is I should say dollar so when I say dollar and then execute this, let's try to execute this in the previous slide ok so what we will try to do is we'll try to execute this command.

(Refer Slide Time: 18:13)



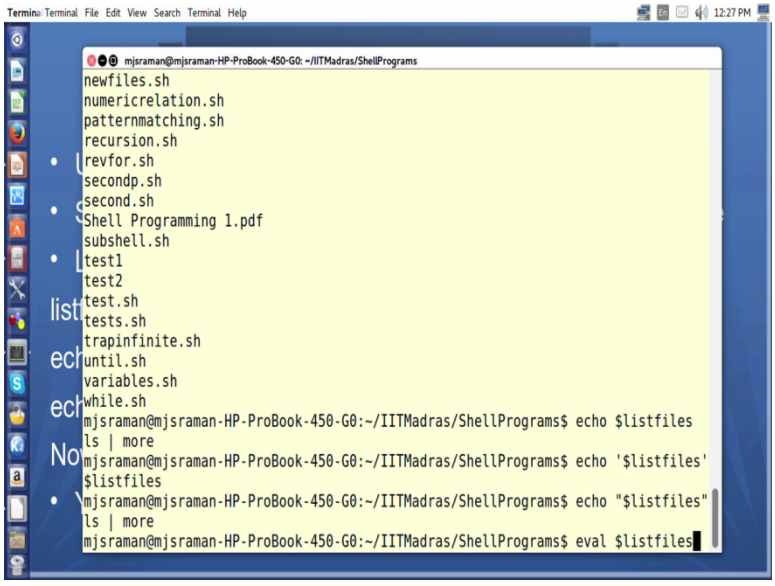
So I say echo and I put dollar list files ok so as I told you it just echoes ls and then space pipe symbols space more. Now if I put it within single code, if you look at this then the expansion is also ignored ok whereas if I put within double codes, then the value gets expanded ok, so if I put within double codes the value gets expanded so essentially if you put something within single code it tells you that ignore that ok, whatever within the single code so don't expand or do anything here. Now this Eval command exactly does the opposite it actually executes it twice ok?

(Refer Slide Time: 19:04)



So what had happened with this Eval command is that.

(Refer Slide Time: 19:08)



So if you remember we put an Eval dollar list files, so what happen was that at the first step of evaluation the dollar list files was expanded, then after the dollar list files was expanded then the command itself was run so you are trying to execute this program twice, the first level of execution happens then this dollar list files itself is expanded and the second level of executions

when the command that is nothing but ls pipe it to more itself again gets run so the program gets run two times, now why is this feature provided ok, so this can be used to write some sort of a programs that modify themselves ok after the first execution so there are situations where you've to modify certain requirements during the execution of the program say I have two choices of execution and I want to execute one choice after looking at what has happened as the result ok of the previous execution so in research circumstance we can use this Eval command.