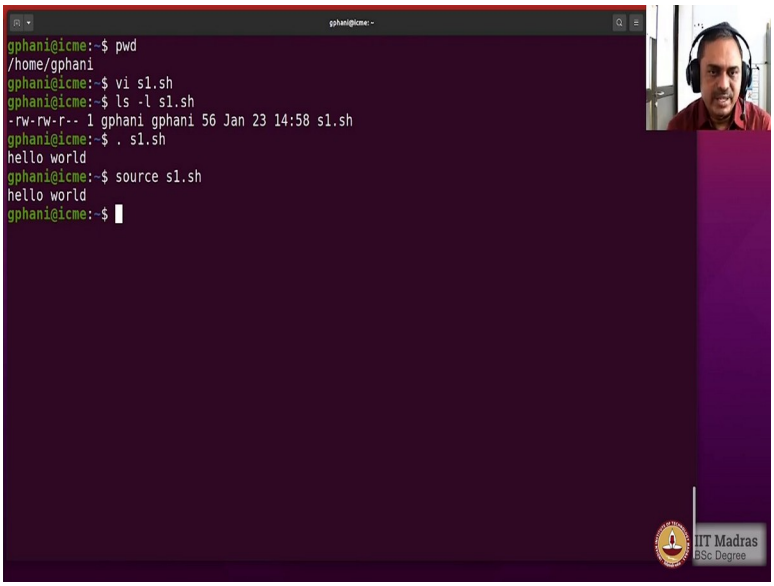
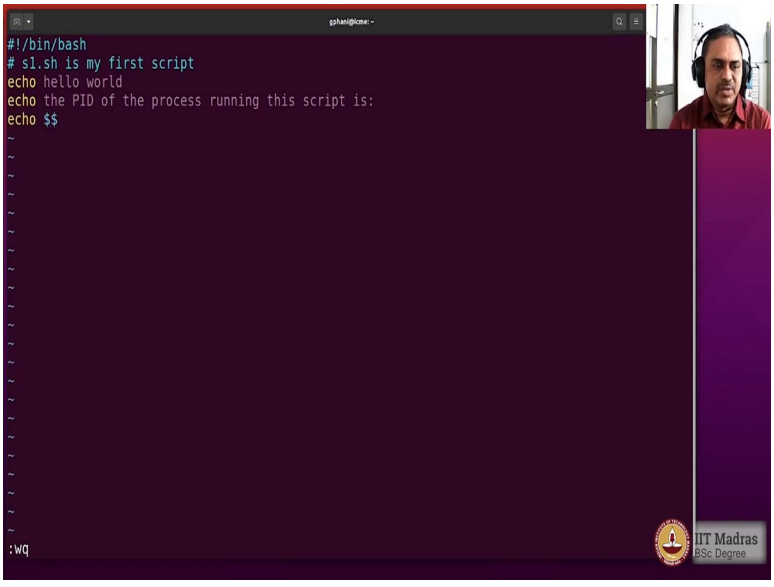
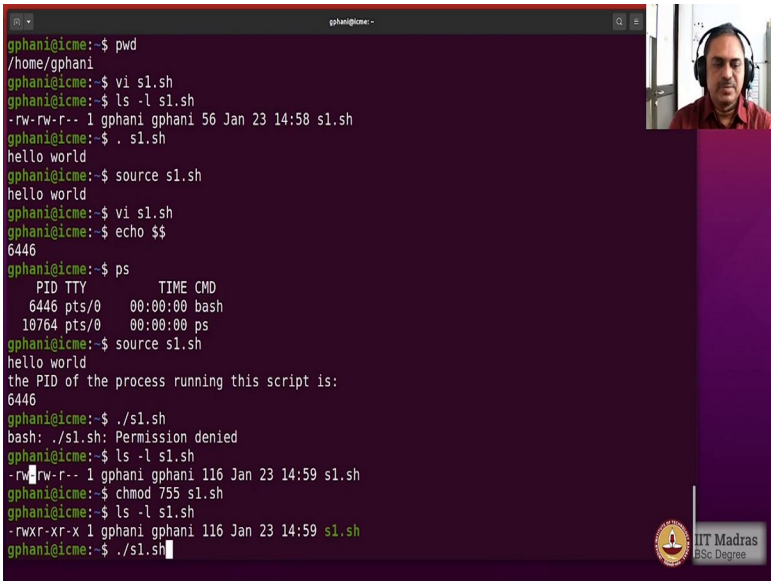
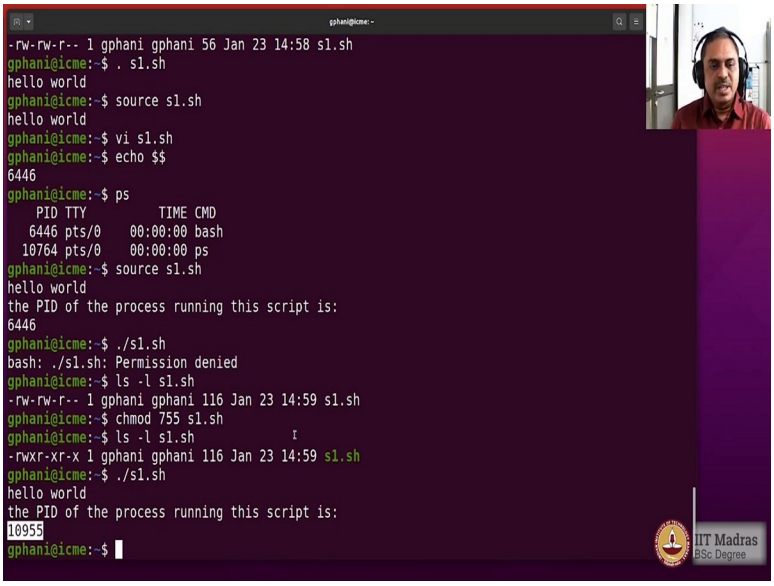
System Commands
Professor. Gandham Phanikumar
Metallurgical and Material Engineering
Indian Institute of Technology, Madras
Bash Script – Part 01

(Refer Slide Time: 0:14)









So, let us get started with creating our very first shell script. So, I would use vi editor to create the scripts, but you are welcome to use any other editor with which you are comfortable. But to try it out by using a command line editor so that you can open a terminal and within the terminal, you edit the script and also try it out how does it function. So, where am I right now I am in my home directory. So, let me start writing some scripts within this directory itself.

So, I start opening a file using the vi Editor. So, I would say my script 1, So, s1 and usually most of the shell scripts should have some ending so, that immediately we can recognize what type of file it is. So, sh is an ending for bash scripts. So, we will have that as our file extension. And I

can start typing the by first entering in the insert mode, I press I to go to the insert mode. And then, as the custom I would also mentioned that it is a shell script. So, I would write as bin bash.

And then I can start writing some commands here. So, I can add some comments for my own references. So, we can say s1.sh is my first script. And I will just simply like to echo something so I would say, hello world, and then I am done with that. So, escape, colon, and then wq. I am done with that now. So, let us check what does this contain.

So, it is a very small script, and you see that the default file permissions that such that there is no executable permission. And we can already try it out by sourcing it. So, we will source it by using the dot command dot s1 dot sh. So, what it will do is the shell would read the commands listed in the particular script and then execute them.

So, you see that it has gone through executed the command which says echo Hello world, and I mentioned to you that source is a nickname for a dot. So, you could also write source started dot sh and then the same effect will be seen. Now, we will go on to check whether they are actually running in the same shell or a different shell when we run as an executable. So, for that, we need to edit it and so, let us go and try that out.

So, I would say the PID of the process of running this script is and then I would like to mention the PID there. So, now, we are actually checking the value inbuilt value of this dollar dollar variable, which is the process ID of the script which is running. Now, we would actually so, first let us just check what will be the PID of the particular bash shell. So, it says 6446. So, I will type ps and you can see that the bash shell that we are currently running has a PID of 6446.

Now, when I source the s1 dot sh, it shows you the same PID number, which means that when you source a script, then the lines that are contained in the script are executed within the same shell as what we are logged into and running the script from. Now, we would like to execute this and if you try to do that by giving the path so, current directory slash s1 dot sh is the correct way to launch it.

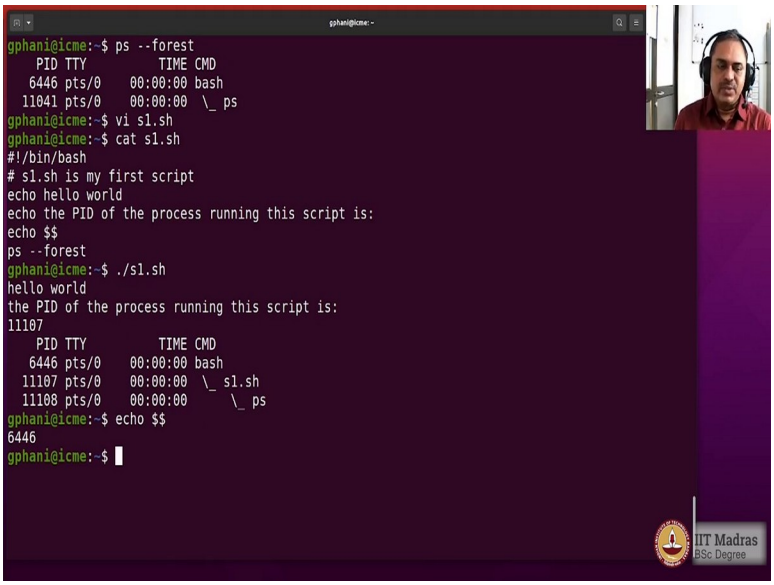
But when you do that you are expected to see an error, the error is that permission denied. Is it because we are not the owners. That is not true, you can see that we are the owners, but there is no executable permission and that is why the error has come. So, we already know how to give ourselves the executable permission. So, I am giving 755 so, that is full permissions for mine, the

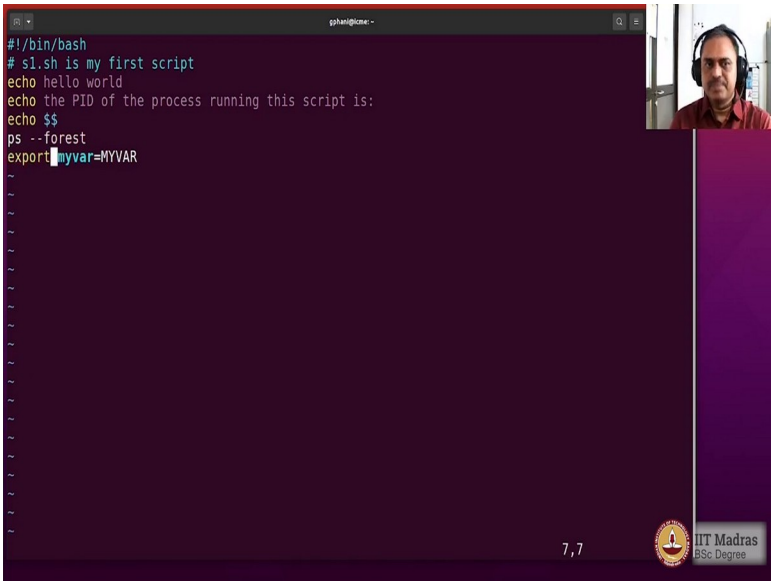
owner and then read and the executable permissions for the group and for others and on the file name s1 dot sh. So, then we go and see the permission. So, there is an executable permission available. And therefore, we now should be able to execute this using the command s1 dot sh.

And when you run you see that the output is very similar the hello world has been output correctly. The second line of code is also output correctly, but the PID number is different. So, which means that when you did this, what happened is that when you typed dot slash s1 dot sh at the bash has spawned another child shell in which this script has been executed. And the child is shell has the PID number 10955.

Now, we can actually confirm this by typing commands which show us the process ID properly. So, let us do that now.

(Refer Slide Time: 5:23)



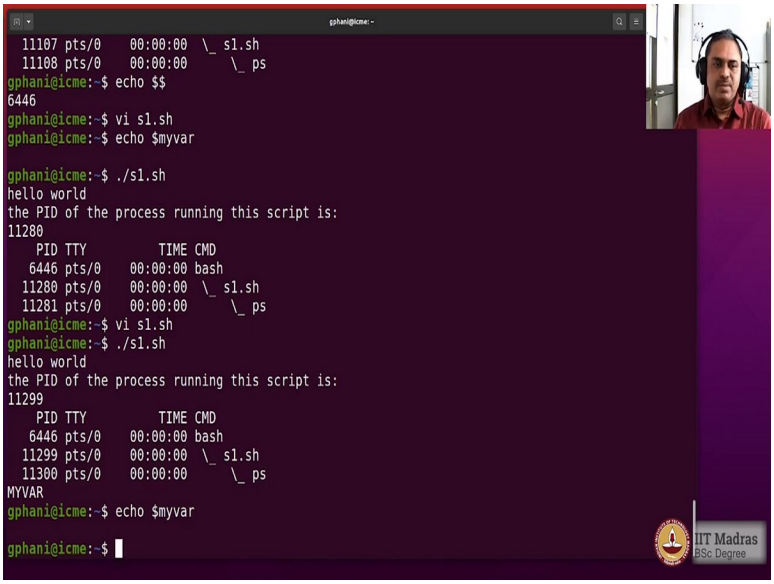


So, you know the outcome of the PS minus minus forest, it will tell you what are the processes that are running and which one is launched by which other process et cetera. So, let us try the same command from the s1 also. So, I just clicked there like that. So, cat s1 dot sh so, that it is there in front of you what it is doing, and we now execute it and you can see that 11107 is the process ID of s1dot sh and 6446 is the parent process ID and then it is a parent.

So, the shell that we are currently logged into, so, you can see that, when you execute a script using dot by s1 dot should, then a child shell is created and therefore, it is actually running as a separate process. And then once it is completed, the control is returned back to parent shell. So, if you happen to make any changes to the variables or environment within that particular shell, then those will not be effective now.

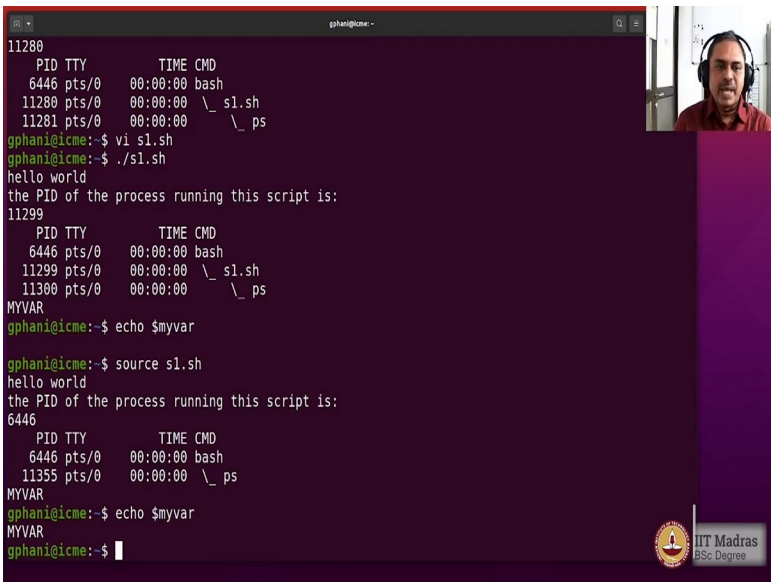
So, let us just try that out. So, I would create a variable here. So, myvar is equal to and let us say give some string and I would like to export so, that the variable is actually available with any other shell after it.

(Refer Slide Time: 6:56)



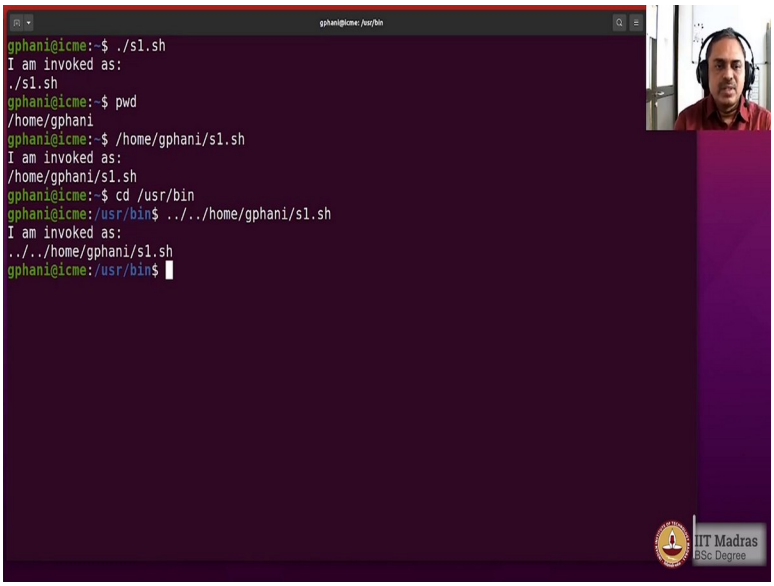
But we will actually see that, if you see myvar is empty, I type s1 dot sh it is executed. And now, I execute by actually displaying the content of that shell. So, myvar is actually available, then I will also now try if it is available within my parent shell. And you see that it is not available, which means that any environment changes that were done in a script that is executed will not be available to the parent shell, but if it was sourced it will be available. So, let us just check that out.

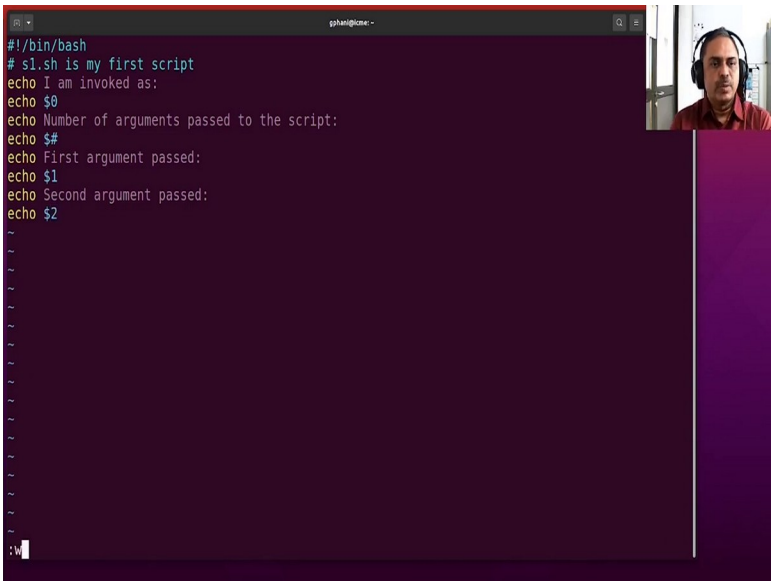
(Refer Slide Time: 7:33)

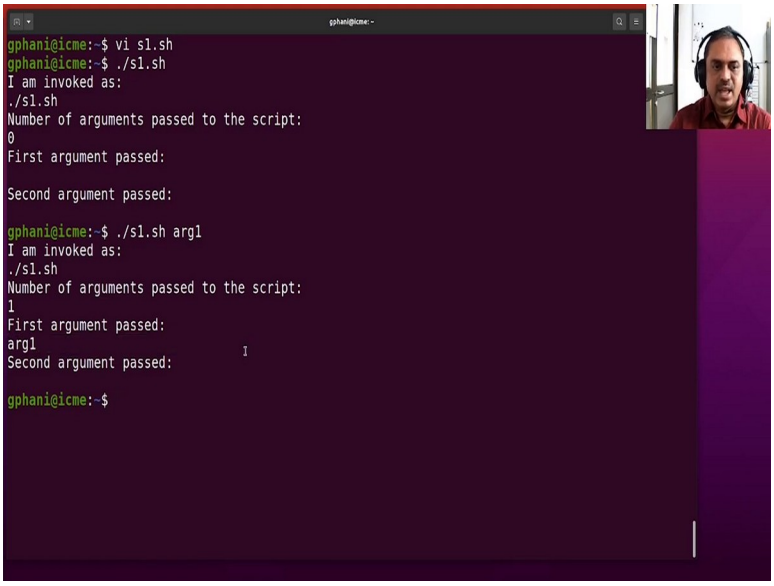


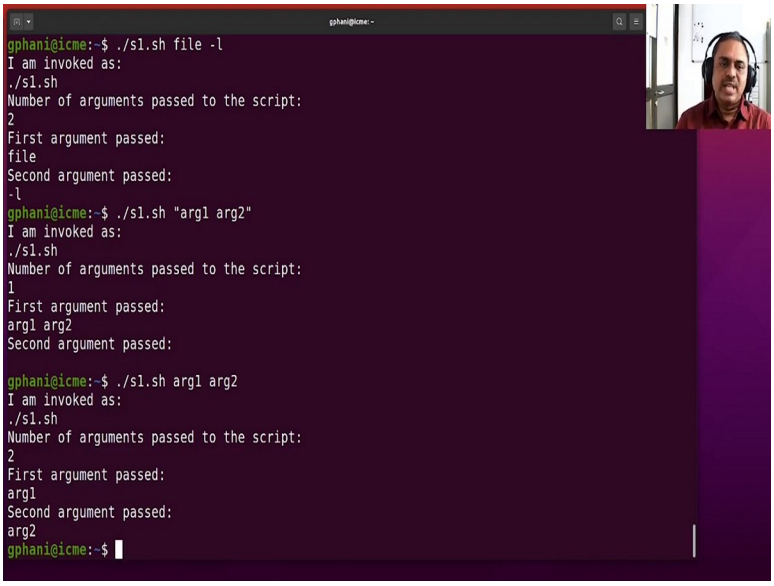
Now, source s1 dot sh and you see that it is actually doing the same execution except that the process tree you can see that it is not a sub shell, but it is actually directly in the same parent shell and therefore, if you now see whether the variable is available, then sure enough it is available. So, this is very important to understand that the way you run a particular script, is it sourcing or is it executing. And you must be very clear which one of the two types of behaviors is what is required for you.

(Refer Slide Time: 8:09)









Now, let us go ahead and try certain other features which tell us about the inbuilt variables that are available with the let us say the dollar star or dollar dollar 1 level and so, on. So, let us try that out. So, I will actually have the comments to illustrate what is being displayed. So, I would say I am invoked as and then we would have dollar 0. So, dollar 0 is supposed to be the name of the file with which we have invoked this particular script.

So, let us go ahead and run it and you see that it says the way we have invoked now where are we are in the home directory. So, we can also run the script using the full absolute path. And you see that when you run it as an absolute path, the way it was invoked is also changing. And in

fact, you can go elsewhere and then try it out also you can go to let us say user bin and then from there I would use a relative path.

So, when I go this way, I have gone up to the root and then I come down and then I can run the script and you see that whichever way you have invoked, that particular string is being displayed. So, it is one way by which everything the script you will know whether you have been invoked with a relative path or an absolute path or without any path information also. Now let us look at certain other variables. So, I just keep this output there itself for the name of the script.

So, we will also go ahead and checkout how many arguments have been passed. And if the first argument is available, let us go ahead and print it. And I can say, here echo number of arguments passed to the script. And here I would say echo first argument passed, is that, and then I will use escape y enter to copy the two lines and then print them here. And then replace cw to replace the word second argument.

And here I would use r to replace one single character 2 and therefore, we have edited the lines by copy pasting vi editor. So, let us go ahead and try this out. Now, I do not pass any arguments. So, it says there are 0 arguments passed, and there is no error when you want to try printing out the first 10 Second arguments, they are not available so it is null, and now that is all there is no error out there.

Now, we can give the verse as follows. So, we can say argument one and you will see that the number of arguments passed is one and it says that the first argument is arg 1 and the second argument is null. So, now I put arg 2 and you see the second argument is coming out quite nicely. So, when we actually pass on, let us say minus l file, then how the shell knows the arguments that we are passing on is actually through these variables.

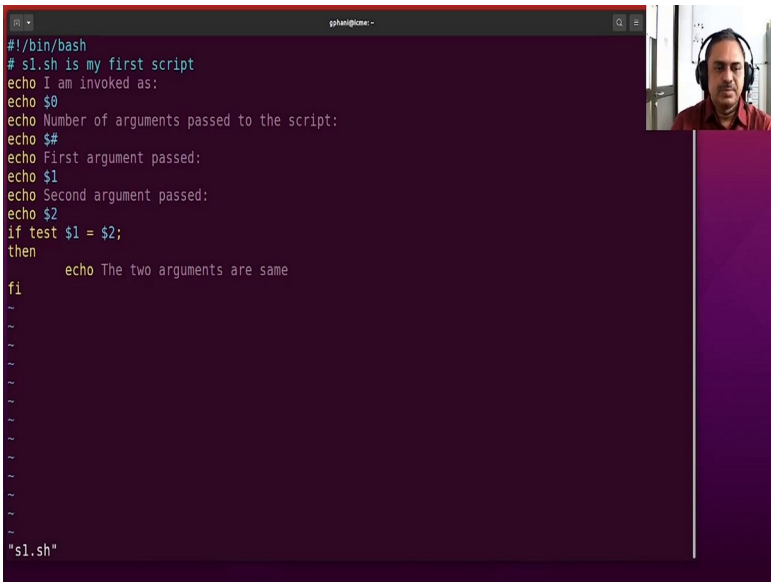
So, it knows that minus l is one of the arguments and file is another argument and the sequence with which they have come is also known. Because we know the positional placement of the arguments on the command line. So, to know that we can actually write it in a different manner. And you see that the first argument now is file and second argument is minus l. So, the shell actually knows what are the arguments you have passed and in which sequence.

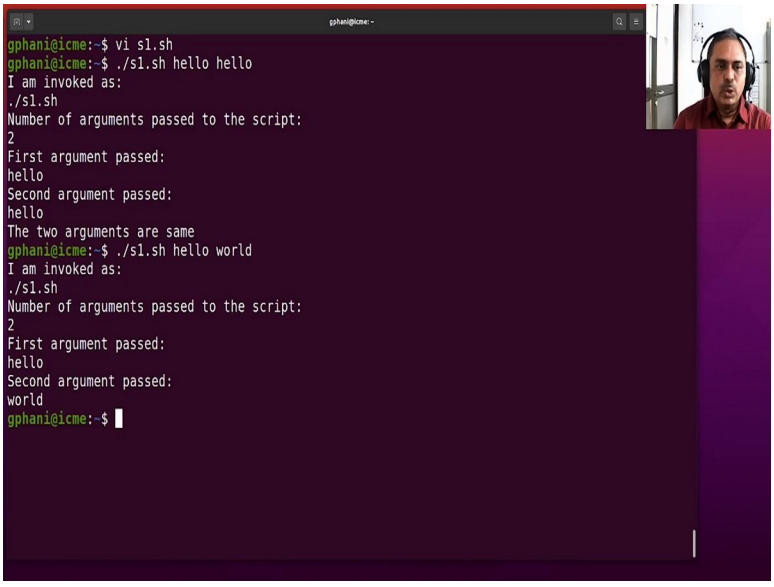
And therefore, it can take certain action within the script to handle those options accordingly. Now, if you want to pass arguments, where the blank space is not to be interpreted, you could

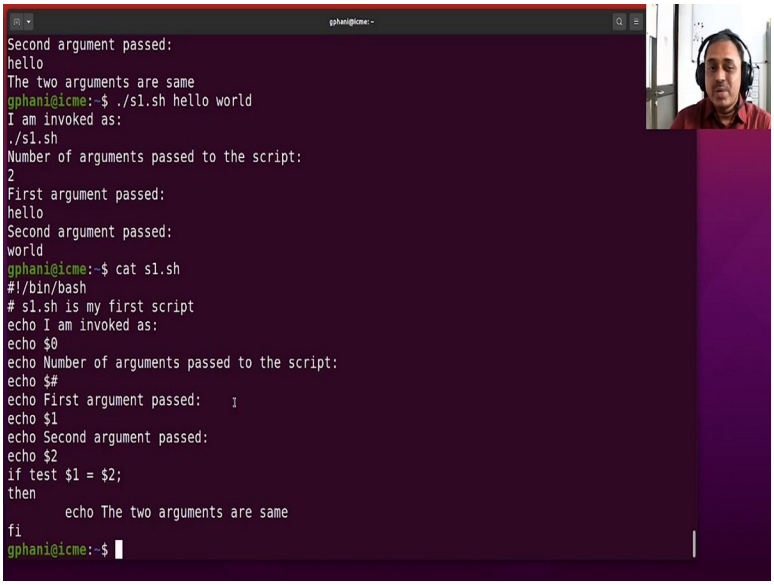
also do that by using the string possibility. Now, you will see that arg 1 arg 2 is actually passed as a single string as the first argument and then the second argument is null. So, you could actually compare that with the behavior here arg 1 and arg2 are are two different arguments, the first and second arguments namely.

So, therefore, you can see that the way you have to work out is by ensuring that the space has to be recognized as a default field separator now, we can also override that.

(Refer Slide Time: 12:47)





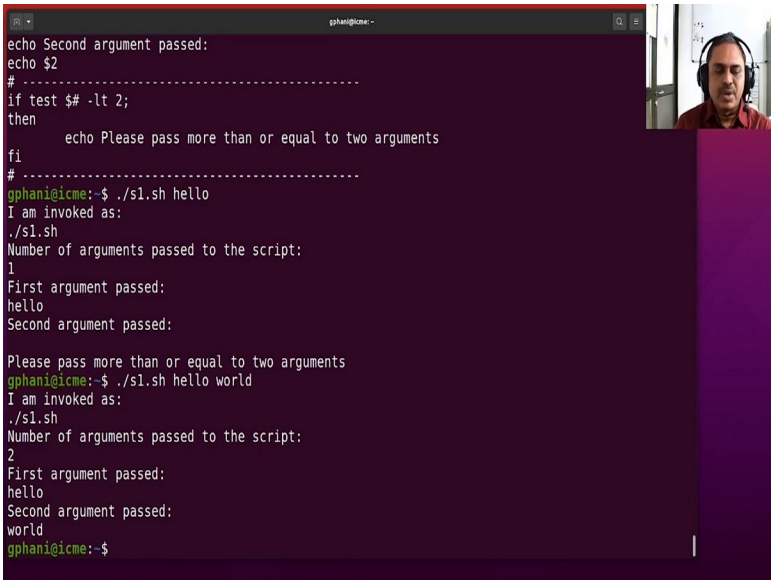


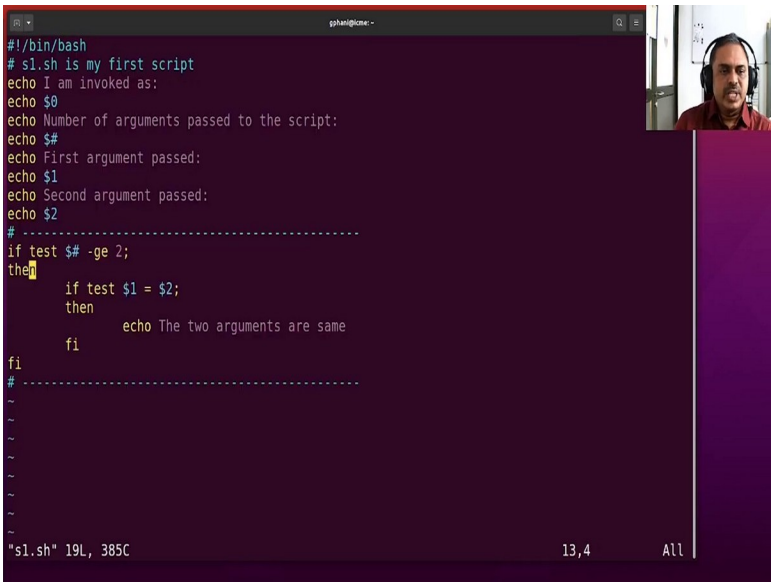
Now, let us start looking at certain loops also. So, we have already learned some of those in the lecture. So, let us go ahead and try that out now. So, we will test whether the first and second arguments are same. So, if test then dollar 1 is the first argument is it same as dollar 2. And if it is same, then we would like to say the two arguments are same. If they are same, it will tell that they are say, but otherwise, nothing else.

So, let us just try this out. And hello, hello. And you can see that it has recognized the two arguments and it also telling that they are same. Now, let us say I type hello, and world. So, the second argument is not the same as first, and therefore, that particular loop has been ignored, because we are not entering the loop. And I will just show you. So, this part, then, and then for this part, we are not entering.

Because this particular test condition has been false because the two arguments are not same. And therefore, nothing is happening with respect to the execution. So, after the echo of dollar to nothing much has happened. So, this is how you can actually check now you can also check how many arguments have been passed.

(Refer Slide Time: 14:24)



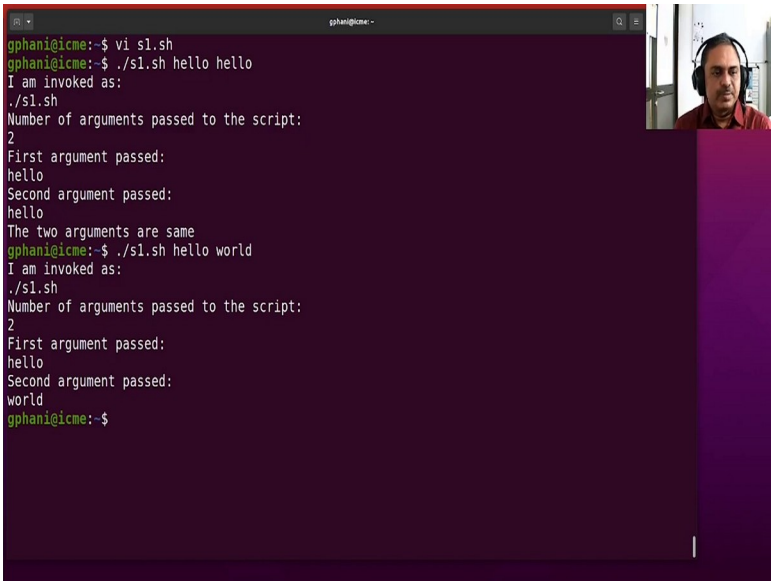


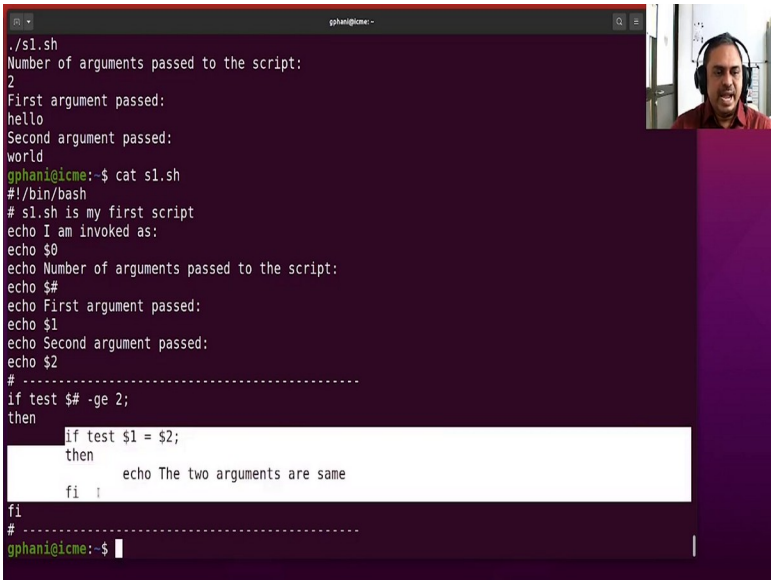
So, here is the way I have modified the script a little bit, where we are past testing for the number of arguments, and if it is less than two, then I want to give an error. So, let us just check that out how it works. And you see that the second argument is not there. So, there is an error that says please pass more than two or equal to two arguments. And if I actually pass on the argument, if I pass on two arguments, then the error is not displayed in the both arguments are not displayed.

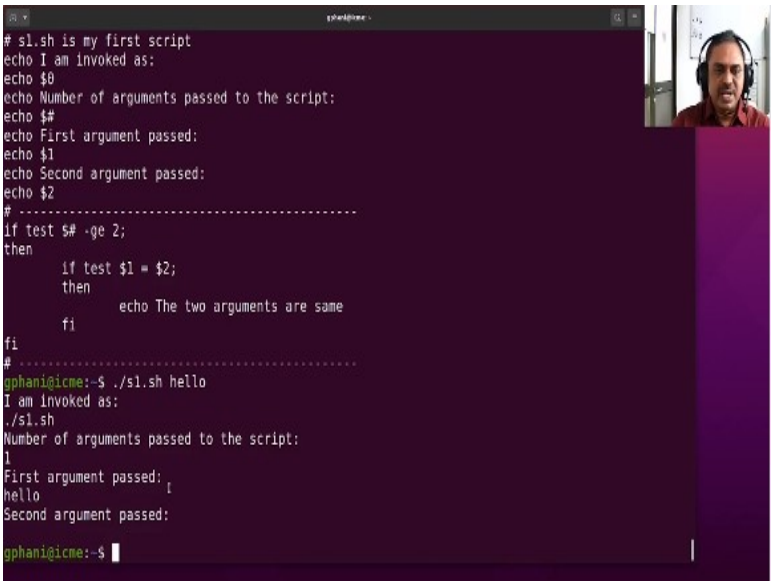
So, you see how we have edited the particular script where we have made the operator usage minus ge that is greater than or equals 2. So, only when the number of arguments is greater than

or equal to 2, then we would like to compare the first and second arguments. So, otherwise we would not like to compare and so, let us see how this works.

(Refer Slide Time: 15:11)



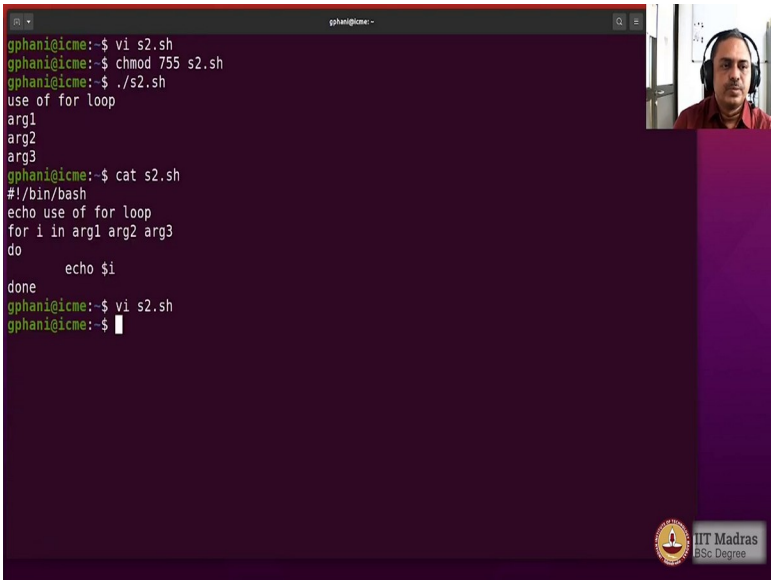


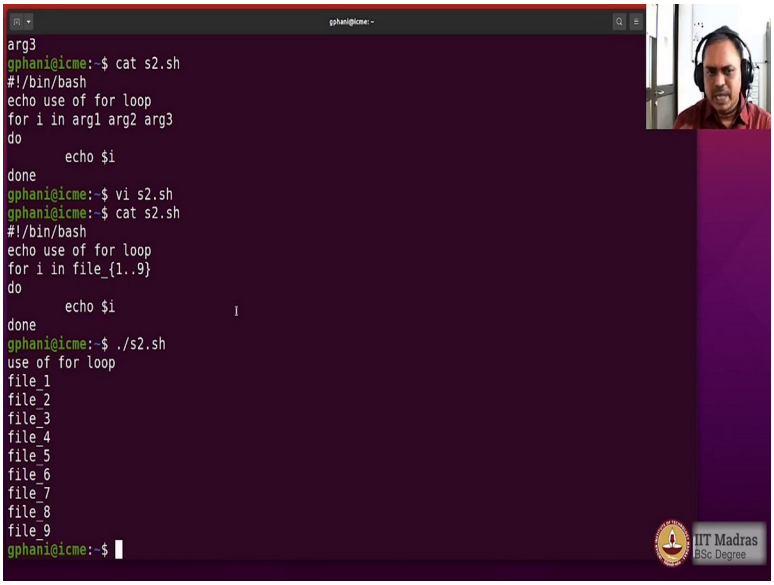


So, you will see that the two arguments are taken and compared and the output that is the two arguments are same is working out quite well. And let us say we give different words then that particular check is not passed and therefore, we are not having a statement coming from the nested loop. So, let me just show the script here. So, you see that we are entering this part here only when the arguments are same.

And we are not entering the loop at all if the arguments are less than two. So, let us try that out. So, we are just not even entering the loop if the arguments are less than two. So, that way the script is slightly better to handle the situations where there are two arguments or more and the trend trying to compare them et cetera.

(Refer Slide Time: 16:11)





Now, let us try to write another script as s2 dot sh in which we would like to explore some other kind of loops. So, bin bash, and here is where I would like to just simply try to use a for loop. So, echo use of for loop and what I would do is for then i in there I need to give a list. So, we can say arg 1, arg 2, arg 3, then do echo dollar i done. So, this is a very simple kind of a script. So, let us just try this out chmod 755 s2 dot sh.

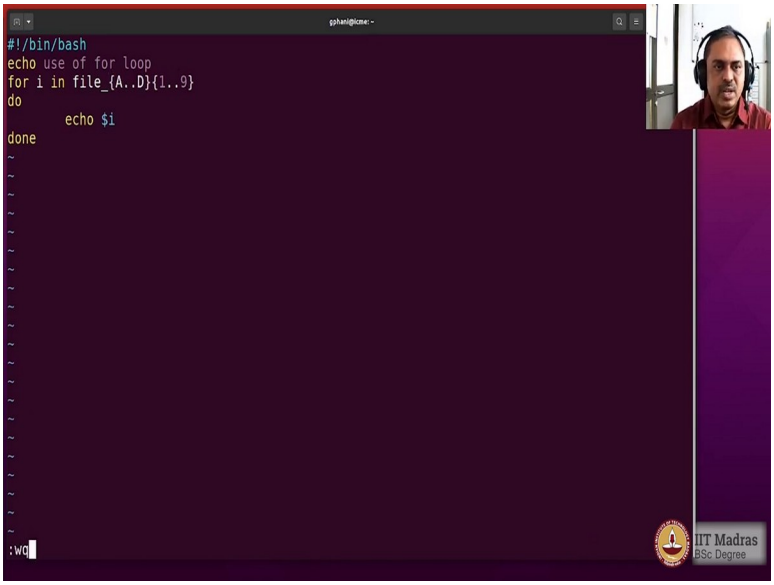
Now, when we run you see that all the three arguments have been printed one after other, because the loop here is working to just do only echo and each of these are treated as separate values in a list of three elements separated by a space and therefore, i is assigned the values that are in this list and then that particular value is printed out. Now this list can be provided not just by having spaces and typed in.

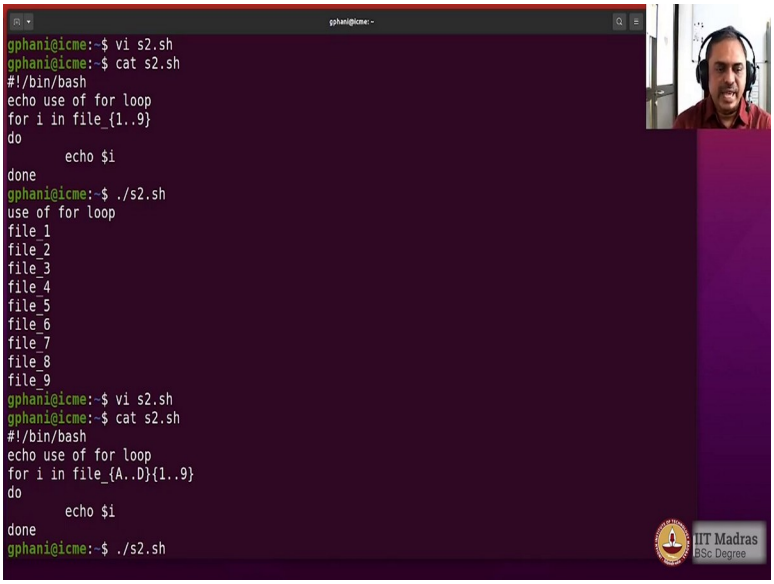
But it can also come from any variable or output of a comment also let us try that out. So, we have already the shell expansion possibilities. So, using the double dots. So, let us try that out for example, file underscore let us say and we know that the braces can be used to have the expansions. So, I would have one then double dot 9. So, cat s2 dot sh. You see that what has happened here.

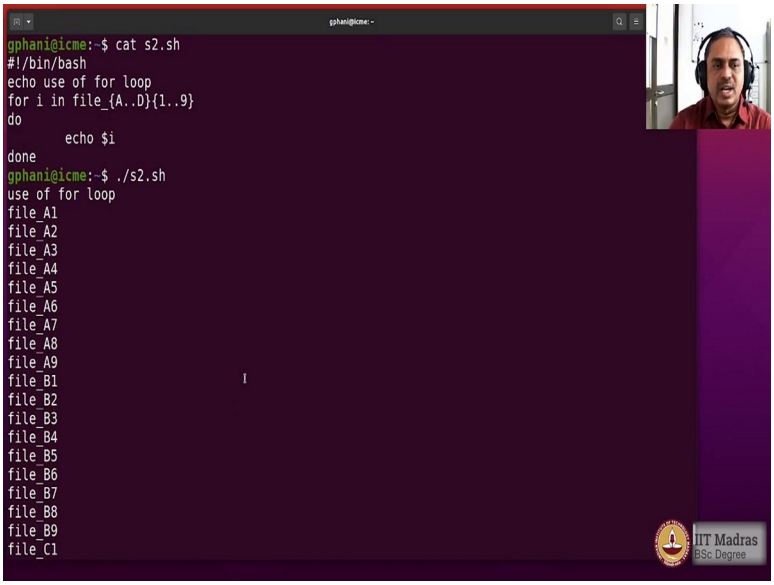
So, underscore and then the brace expansion has been used, So, that you go from 1 to 9. So, file 1 up to File 9. So, we have actually started using the brace expansion to automatically create a list and then from that list every element is assigned as a value to the variable i and then that

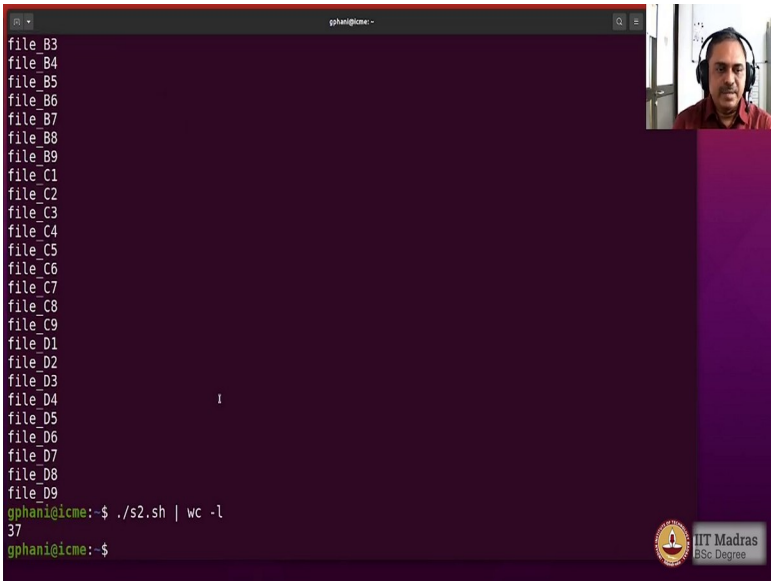
variable is being printed onto the screen. Now, you could also combine this with other kinds of expansions that we are familiar.

(Refer Slide Time: 18:38)



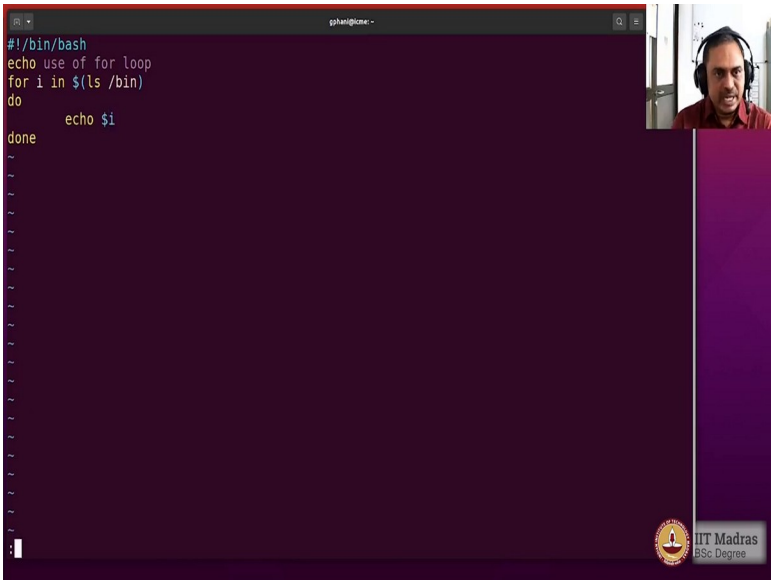


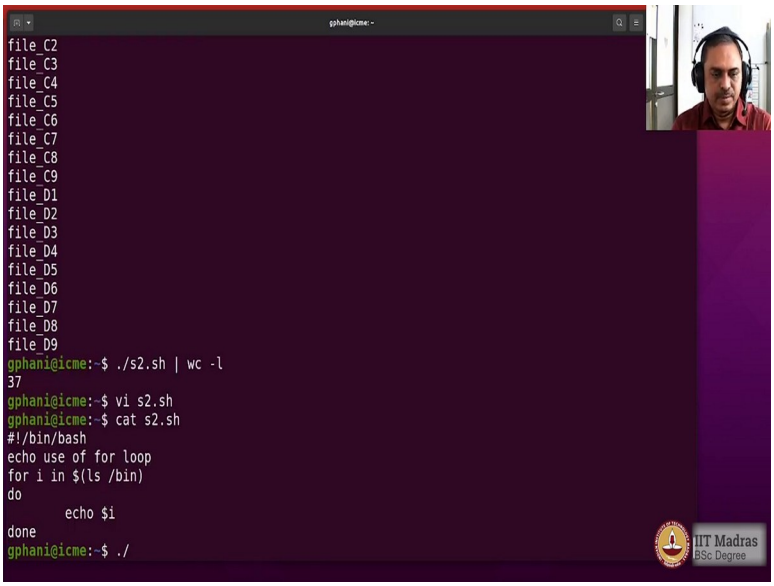


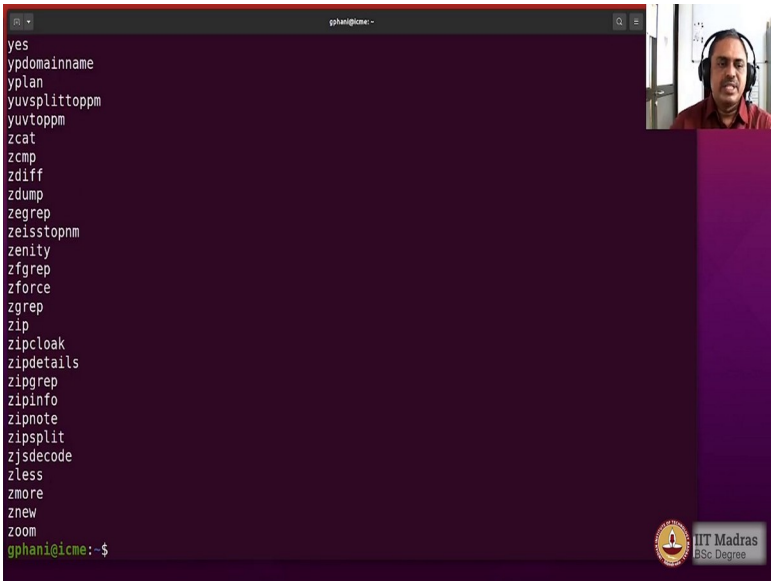


So, you would see that we are actually having two expansions and this means that from A to D and 1 to 9. So, that is 4 into 9. So, 36 lines have to come and let us see if that is the case. So, you see that from A1 to A9 and then B1 to B9 and so, on up to D9 has come. How many lines have come? So, 37 That is 37 because this is the original line so, one plus 36. So, you can see that the brace expansions can be used quite well along with the other features of the shell.

(Refer Slide Time: 19:28)

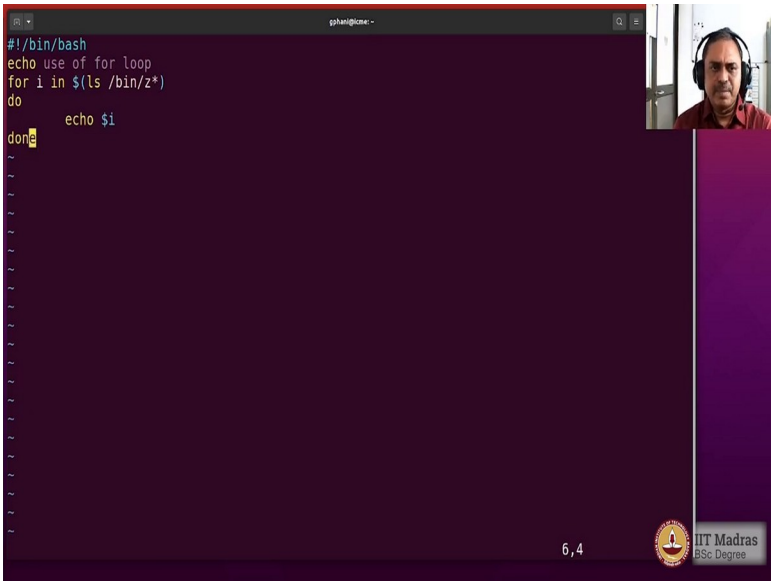


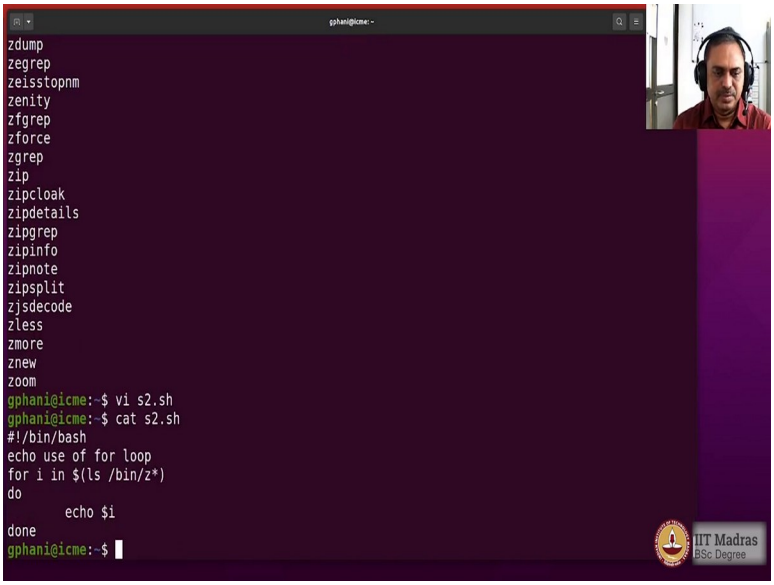


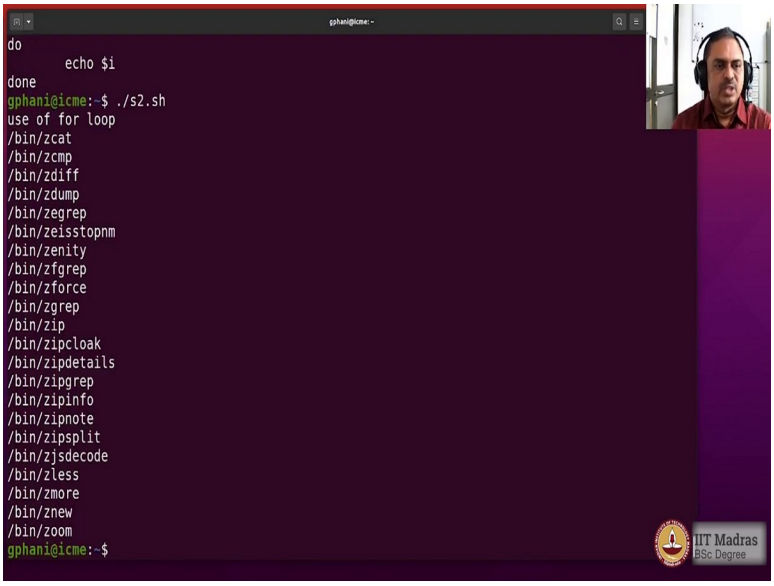


Now, you could also do it with respect to the command output. So, ls is a command for which we have the output that we know. So, what we would do is using the expression like this ls slash bin. Now, every file that is located in the slash bin directory will be available as an element of the array and then that will be assigned to i and it will be printed onto the screen. So, let us try that out. Now. And you see that the list of output coming from the slash bin directory is being shot.

(Refer Slide Time: 20:13)



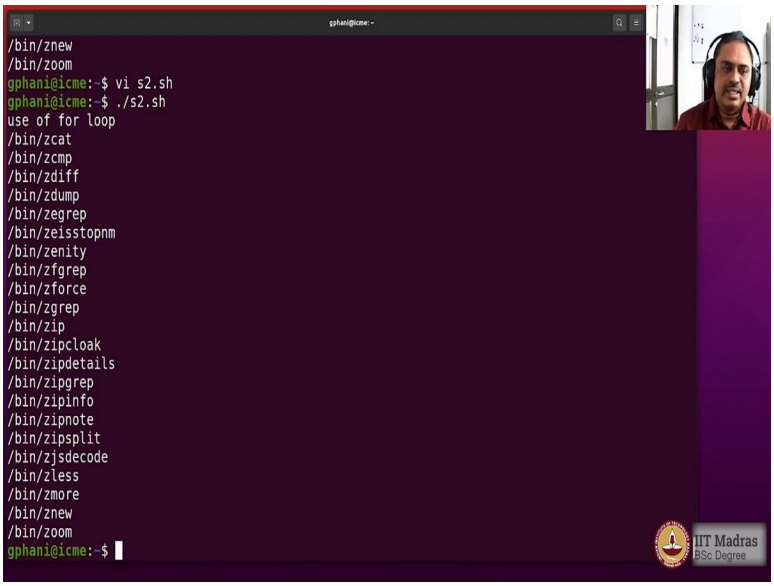




And we can now go and edit this command ls slash bin in a way that would help us to reduce that out. So, I put slash z star. So, that means that I want only those files in the bin directory which are starting with a z character. So, you can see that all the files in the bin directory starting with Zed are listed here. Because in our script, we are saying ls slash bin slash z star. So, this command has been executed the output which is what we have seen here, available as a array to in this line.

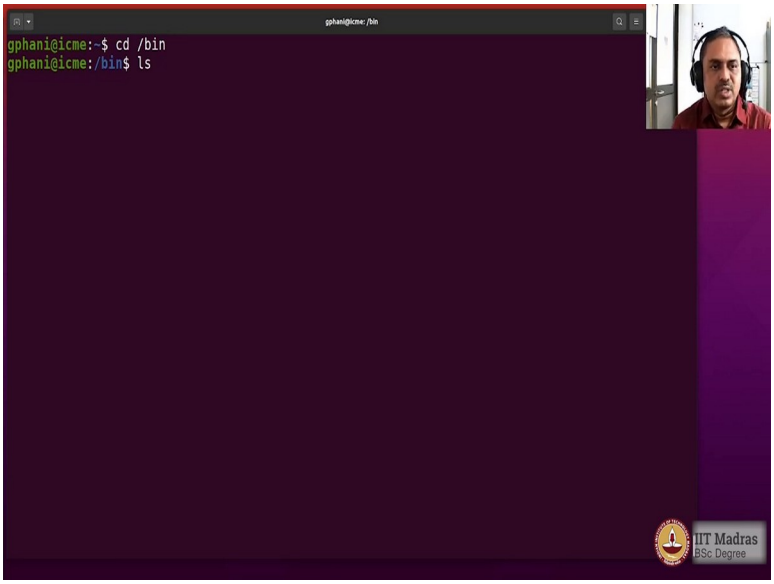
And therefore, i is assigned a value for each of these elements in the array, and then that is getting printed onto the screen using the echo command.

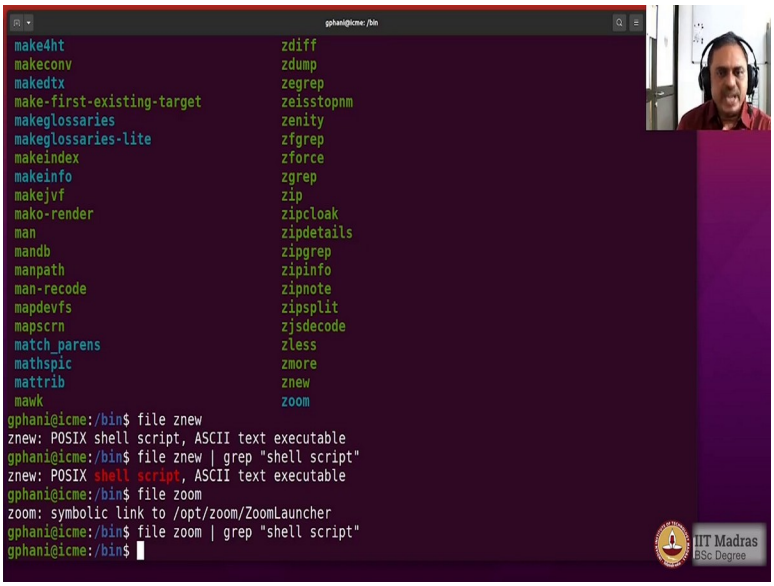
(Refer Slide Time: 21:01)



And this particular feature command substitution can also be achieved using the back ticks. Which I am just showing you now see that the output would be identical. So, either back ticks or dollar followed by a single parenthesis can be used for comments substitution, as we have illustrated right away.

(Refer Slide Time: 21:22)





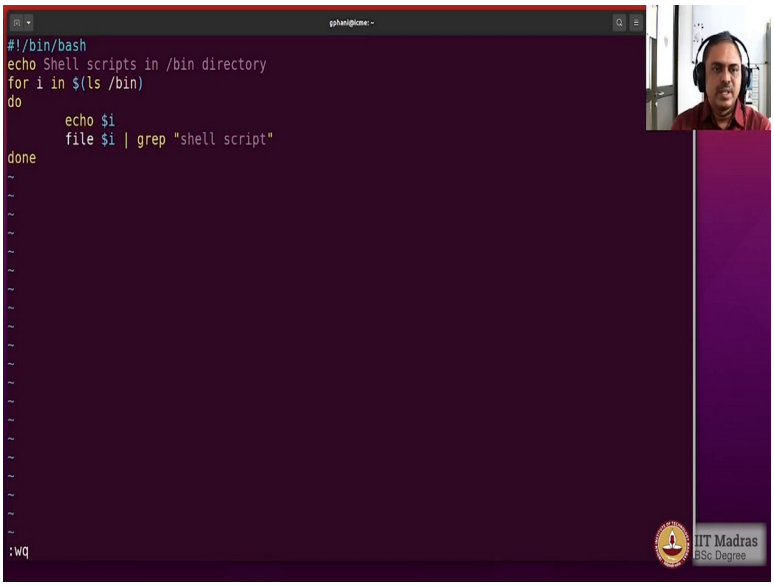
Now, one way by which you can learn about cell scripts is to actually look at some shell scripts that are already within the Linux environment. So, how do we find that out? So, let us go to the bin directory. And there are many, many files here. So, one of those commands, which we know will help us to understand whether the particular file is a script or a binary is the command file. So, file and then z new.

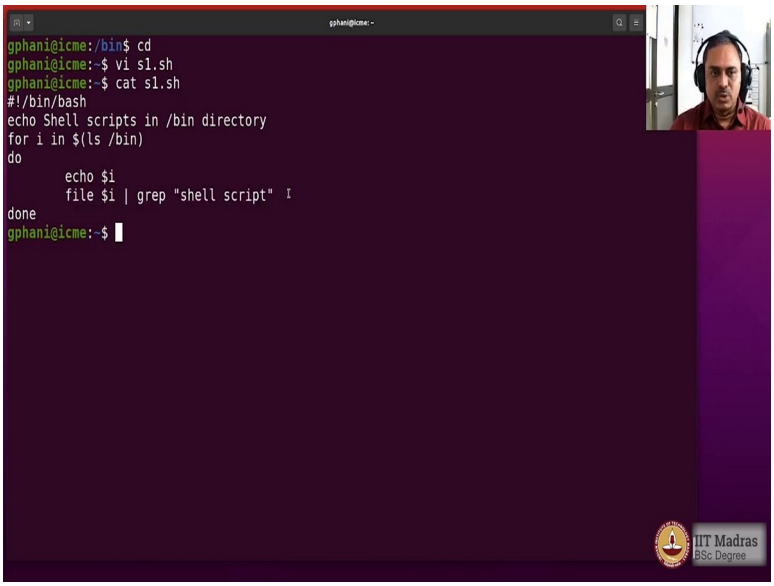
So, you will see that it says there is a shell script. Now, what we would do is pipe it to grep which we have already seen in the recent class to look for some pattern. So, here we will save shell script is a pattern. So, I would like for that and if it was available, it will actually print it

out. Now, let us say I use it for something else file zoom, and it says is a symbolic link. So, the pattern shell script is not appearing in the output of the command file.

And therefore, if I were to pass it on with this, then what happens is that there is no output so, null. So, which means that I can actually use the combination of a file command on every file name, pipe it to grep and ask whether it is a shell script to understand whether it is a shell script or not. So, let us find out how many files in the slash bin directory are actually shell scripts.

(Refer Slide Time: 22:53)



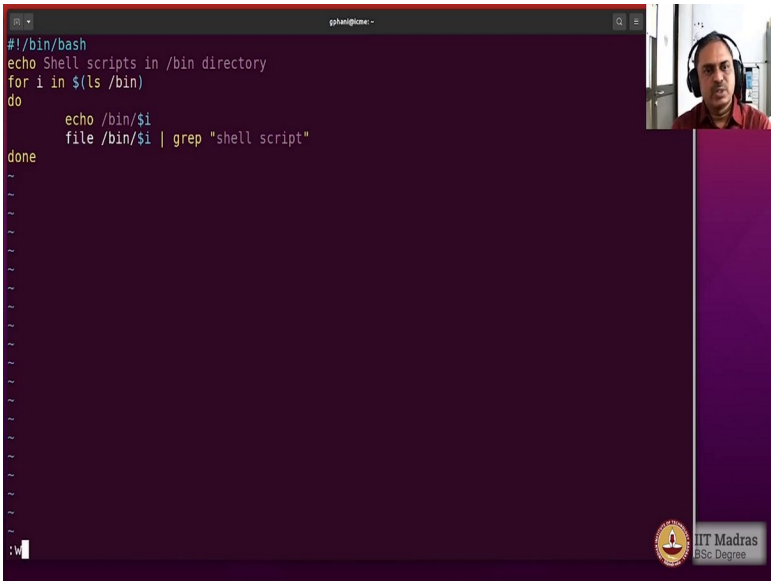


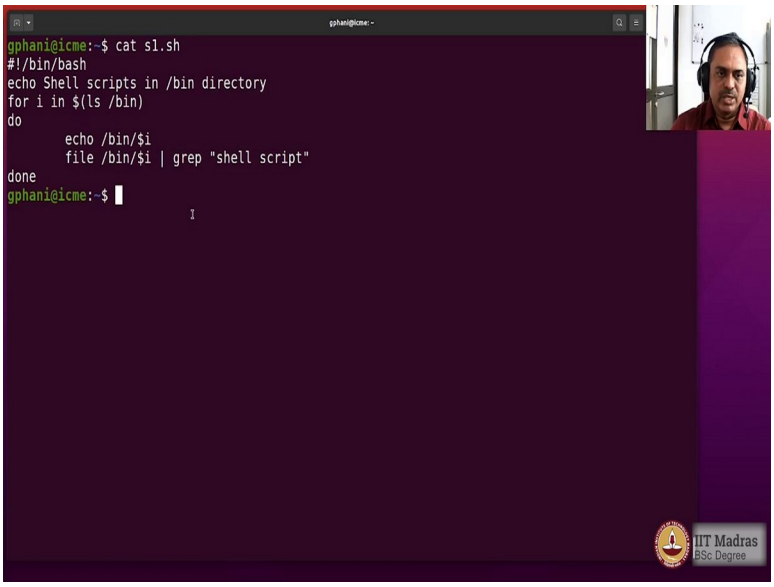
So, for that, we will go ahead and write a shell script echo shell scripts in slash bin directory. Now, we need to run through all the files that are in slash bin. So, for that what we do is for them

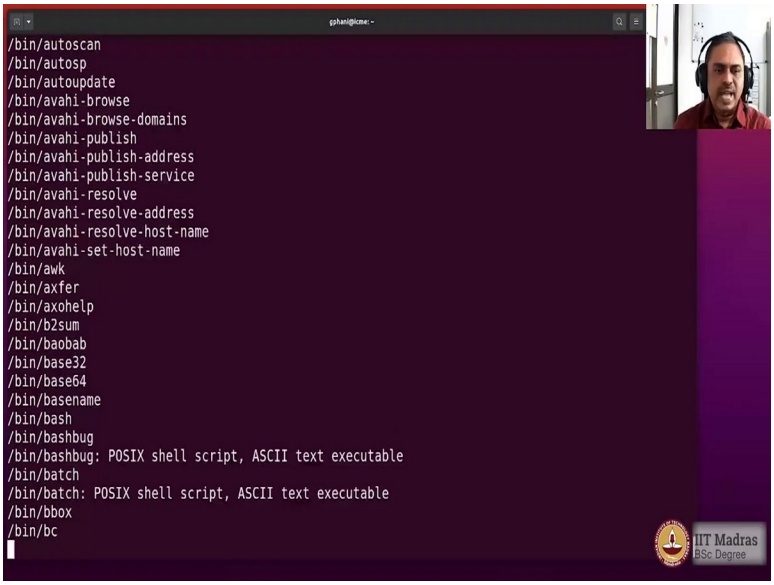
the file is let us say i in and then the list of files in the bin directory I need. So, I would actually use it like this. So, this is a loop and what do we need to do? So, first of all, let us just echo the file name to check whether the list of files are coming properly.

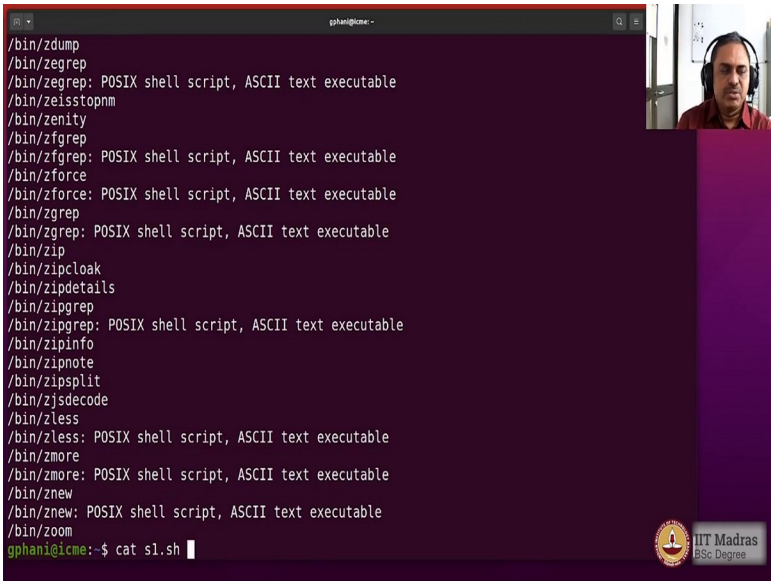
And we would also then type it out the command that we were doing earlier file dollar I grep shell script. So, let us try this out cat s1 dot sh to just to see what is it that we have written. So, for every i in the list of file names in the slash bin directory, what you have to do is print the file name and then get the output of the command file on that file name and pipe it to grep to check whether the output contains the phrase shell script and if it contains then print that out.

(Refer Slide Time: 24:34)





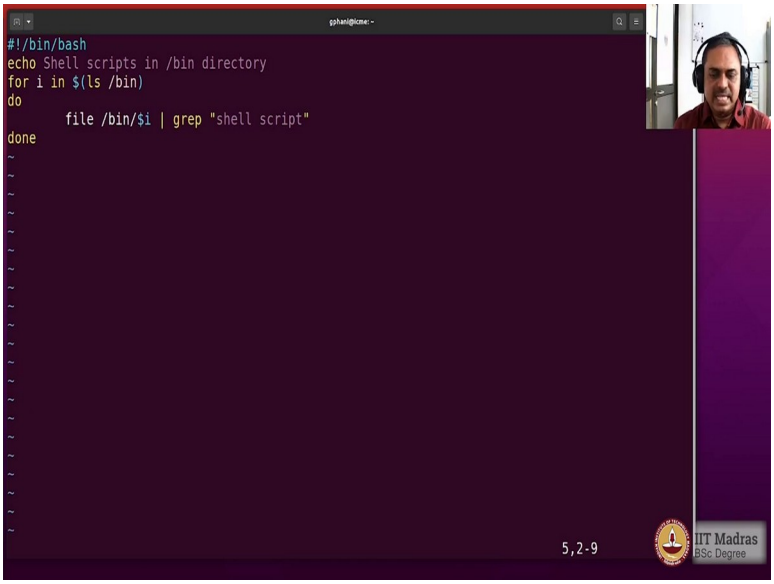


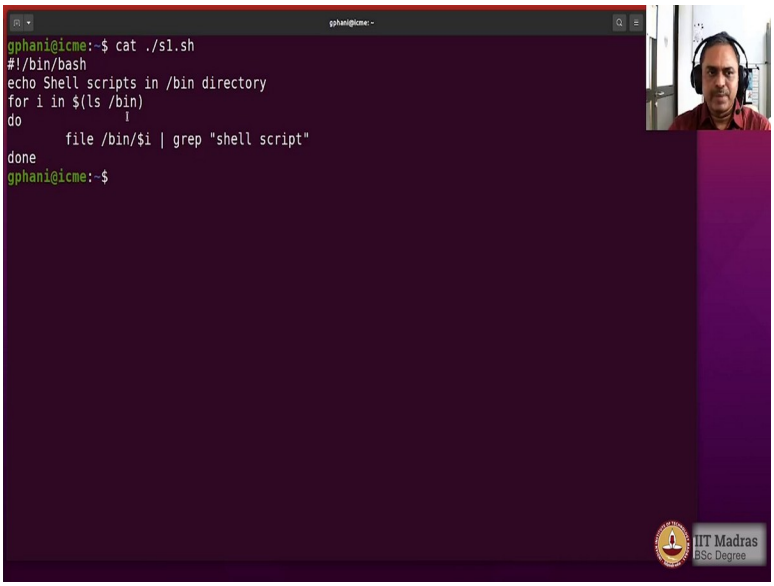


So, because the file name i is coming to the slash bin directory, you need to put slash bin in front of the dollar i so, that the file name is given with the full path and therefore the file command would then work correctly. So, let us go ahead and try this out. So, here is what the script is run. So, slash bin slash file name. And that is a file that is being processed with the file command.

And then, we are grepping the phrase shell script within the output of the file command on that particular file. And if it is successful, then there will be a line that is displayed. Otherwise, only the file name is displayed. So, let us try this out. And you see that the file names are all getting processed, and some of the files are getting the output, which we are expecting. So, let us that finish. So, you could see that there are a lot of files that are actually scripts.

(Refer Slide Time: 26:00)



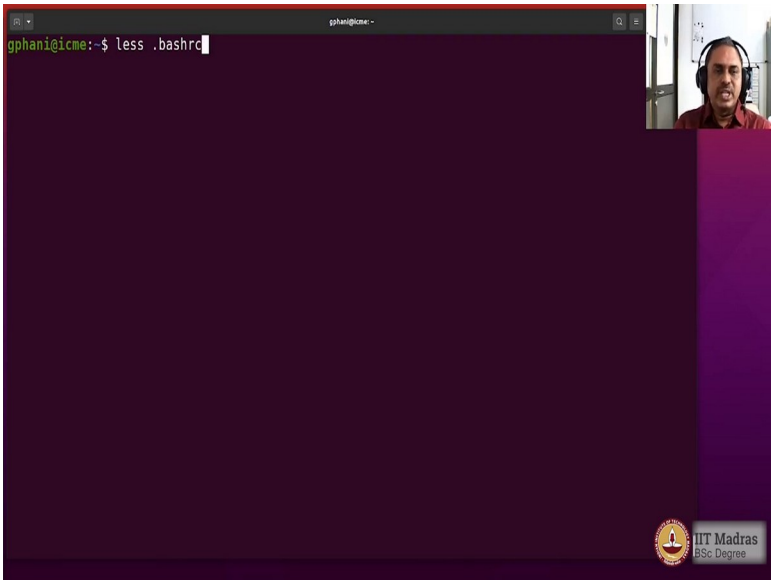


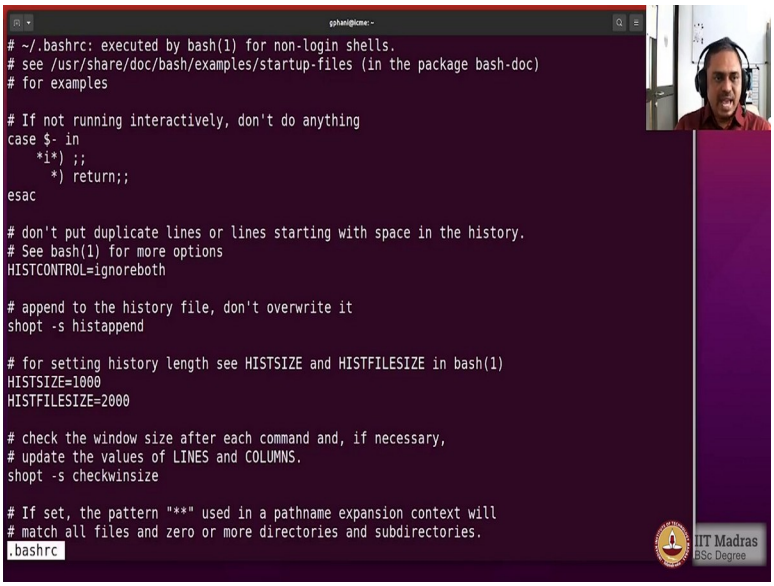
And let us go ahead and limit the output by not echoing the file name and directly listing only what are actually shell scripts. So, now the output will be a little bit less. So, let us go ahead and try this out. Cat s1 dot sh. So, you can see that what I am doing is I am not echoing the file name I am only displaying the text, if it was a shell script, otherwise, do not do anything. So, you will see that now the output will be much less.

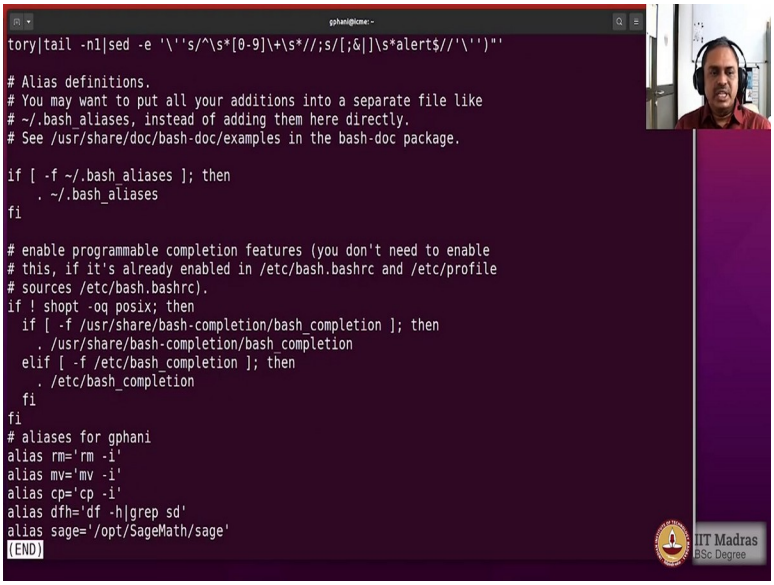
And only the shell script names are being displayed on the screen. Now how many lines came you could also analyze that and you could pipe it to a file and then find out how many scripts are there and then go on to read each of them to understand some of these features that very used to by the Linux operating system. You could read these shell scripts to explore various features of the bash scripts as they have been used by the Linux opera system itself.

And do not use superuser permissions to modify any of them because you are not supposed to touch them unless you know what you are doing. You are welcome, of course, to copy them to your home directory and modify them to see how do they work. But do read the main page for that particular command to see actually what are they supposed to do.

(Refer Slide Time: 27:22)







Now, I would also show you that we have some shell scripts that are processed automatically sourced automatically when we log in to the bash shell. So, here is one such example bash rc in my home directory. And this actually has various commands that are being run when I start the bash shell and it has loops it has the if loop case. It has aliases and so, on. So, you could actually read these kinds of scripts to actually see what are the features that are used in the bash scripting language, and then go on to explore some of those in your own scripts.