

**System Commands**  
**Professor Gandham Phanikumar**  
**Metallurgical and Materials Engineering**  
**Indian Institute of Technology, Madras**  
**Shell Variables - Part 1**

(Refer Slide Time: 0:14)

## Shell variables



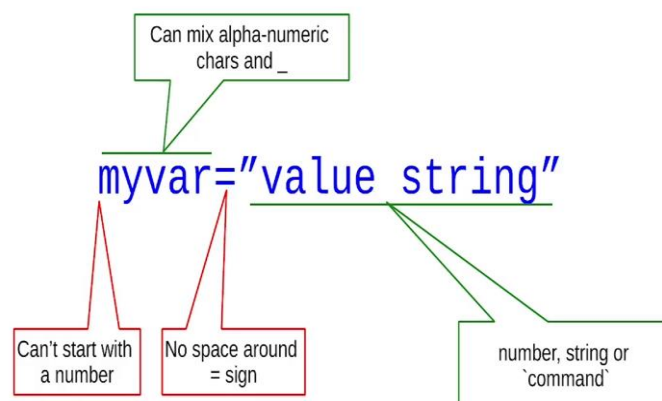
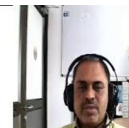
Creation, inspection, modification, lists...



In this session, we will be looking at Shell variables, various aspects of Shell variables such as the creation, the inspection of those variables, the values, modification of those variables and lists of those variables in two different types of arrays.

(Refer Slide Time: 0:33)

## Creating a variable



So, the way to create a variable is directly by assigning a value, and the variable should not contain a digit at the beginning of the name. It can contain alphanumeric characters as well as

an underscore. There should not be any space before or after the equal to sign. And if the value is a number, you can directly provide it on the command prompt. But if it has a space, or if it is a string value, it is safer to enclose it in quotes. You may also assign the output of a command as a value to a variable by using the command in the back quotes, we will see that in a demo.

(Refer Slide Time: 1:19)

## Exporting a variable



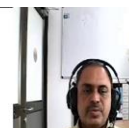
```
export myvar="value string"
or
myvar="value string"
export myvar
```



By exporting a variable, what we mean is making the value of that variable available to a shell that is spawned by the current shell. And you can do it directly by assigning a value to that variable and exporting it in one line or by creating the variable and then exporting that on the command prompt.

(Refer Slide Time: 1:42)

## Using variable values



```
echo $myvar
echo ${myvar}

echo "${myvar}_something"
```



And the variable value itself can be displayed by using the echo command. The dollar sign at the beginning of the variable name should not be forgotten. It is always safe to use the name of the variable within the braces or the flower brackets. As you will see there is a lot of manipulation that we could do to the value of the variable by inserting certain triggers are commands within the braces. So, it is always a good idea to use the braces while using the values of variables.

(Refer Slide Time: 2:18)

---

## Removing a variable



```
unset myvar
```

## Removing value of a variable

```
myvar=
```



---

To remove a variable it is quite simple, you could just unset the variable. You could also remove the value of the variable by placing nothing after the equal to sign. You could use the same command as what you would use for creating the variable where you write like my var is equal to and then a value. So, if you skip the value and press enter, then the value will be null. And that is also one way by which you remove the content of a variable.

(Refer Slide Time: 2:48)

## Test if a variable is set



```
[[ -v myvar ]];  
echo $?
```

Return codes:

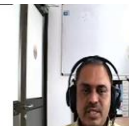
0 : success (variable myvar is set)  
1 : failure (variable myvar is not set)



If you want to test if a variable is available in the memory or not, you could use the test conditions syntax of the bash language and minus v option is to check whether the variable is set. You could check the status of the output of the test command to see if the variable was available or not. We have already looked at the error codes or written codes of various commands on bash prompt earlier.

(Refer Slide Time: 3:17)

## Test if a variable is *not* set



```
[[ -z ${myvar+x} ]];  
echo $?
```

Return codes:

0 : success (variable myvar is not set)  
1 : failure (variable myvar is set)

can be any string



You could also test if the variable is available in the memory or not in indirect manner by using the minus z option. And here you could actually see that the character x can be replaced with any other character and it would work opposite to the minus v option that we have seen just now.

(Refer Slide Time: 3:37)

## Substitute default value



If the variable `myvar` is not set, use "default" as its default value

```
echo ${myvar:-"default"}
```

if `myvar` is set:  
display its value  
else:  
display "default"

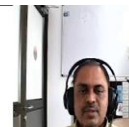
no spaces



If you are using a variable and if you would like to have a default value in situations where the variable is not set, then there is a syntax available within the braces you could have a colon and a minus symbol and then you can provide the value that should be used as a default in case the variable is not set. Remember, there should be no spaces before or after the colon and you are using the syntax within the braces. The logic is if the variable is set, then it will display the value and if it is not set, then it would display the value that is given in the command here which is default.

(Refer Slide Time: 4:17)

## Set default value



If the variable `myvar` is **not** set, then set "default" as its value

```
echo ${myvar:= "default"}
```

if `myvar` is set:  
display its value  
else:  
set "default" as its value  
display its new value

no spaces



If you wish you could set a default value to the variable if it was not set and the syntax is here colon equal to and the logic is that if the variable is set, then its own value will be used. But if

it is not set, then the variable will be set to the value that is given in this command like default and that variable and that value will then be used.

(Refer Slide Time: 4:40)

## Reset value if variable is set



If the variable `myvar` is set, then set "default" as its value

```
echo ${myvar:+ "default"}
```

if `myvar` is set:  
set "default" as its value  
display its new value  
else:  
display null

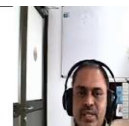
no spaces



You could use the reverse logic where if the variable is set, then you should display a particular value. Here in this case, we are saying default and if the variable is not set, then nothing will be done. The two logics that are there with the plus and minus signs, when combined can be used to have whichever logical loop that you would like to have in a shell script, as you will learn later.

(Refer Slide Time: 5:06)

## List of variable names



```
echo ${!H*}
```

List of names of shell  
Variables that start with H

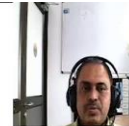


Very often, we are interested in knowing what are the variables that are actually stored in the memory while you are operating within the bash shell. So, you could actually print out the

names of the variables by using this command, echo dollar within the braces, exclamation mark, followed by the character with which you would search for the names of the variables, and then a pattern. So, H star would mean that all the variable names which starts with H will then be displayed on the screen. You could have any other pattern that is supported by the bash syntax.

(Refer Slide Time: 5:39)

## Length of string value



```
echo ${#myvar}
```

Display length of the string value  
of the variable `myvar`

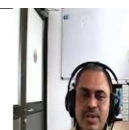
If `myvar` is not set, display 0



Now, the length of the string contained as a value of a variable can also be counted directly. And it could be done using the pound symbol or hash symbol in front of the variable within the braces. In case the variable is not set, then the value will be 0.

(Refer Slide Time: 5:55)

## Slice of string value



```
echo ${myvar:5:4}
```

Display 4 chars of the string value  
of the variable `myvar` skipping first  
5 chars

offset

slice length



You could extract a slice of a string value of a variable by providing what is the offset and the number of characters in the slice that you would like to print out by using the syntax where there are two colons separating the offset and the slice length. If the slice length is larger than the length of the string, then what is available in the string only will be displayed. The offset can also be negative, only thing is that you need to provide a space after the colon to avoid confusion with the previous usage of colon minus symbol. And you could also have the offset coming from the right-hand side of the string.


(Refer Slide Time: 6:35)

## Remove matching pattern

```
echo ${myvar#pattern}
echo ${myvar##pattern}
```

match once

match max possible



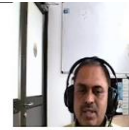
If you would like to match a pattern within the value of a shell variable and print whatever is not matching, then that can be done using the hash symbol. The pattern can be similar to the regular expressions that we will be learning later on. But it could just be a dot and a star so that you could look for filename extensions are parts of a particular file name.

If you use the pound symbol twice, it means that the pattern matching will be done as many times as possible within the value of the variable. And if it is used once it means that the pattern matching is done only once. So, whatever is matching the pattern will be removed and the rest of it will be displayed on the screen.



(Refer Slide Time: 7:20)

## Keep matching pattern



match once

```
echo ${myvar%pattern}  
echo ${myvar%%pattern}
```

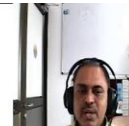
match  
max possible



Sometimes you want to actually display what is matching the pattern rather than remove it. So, that is accomplished by using the percentage symbol. So, a single percentage symbol is used to match only once and it is used twice to match it as many times as possible. So, whatever is the matching pattern will be displayed on the screen.

(Refer Slide Time: 7:40)

## Replace matching pattern



match once &  
replace with *string*

```
echo ${myvar/pattern/string}  
echo ${myvar//pattern/string}
```

match max possible  
& replace with *string*

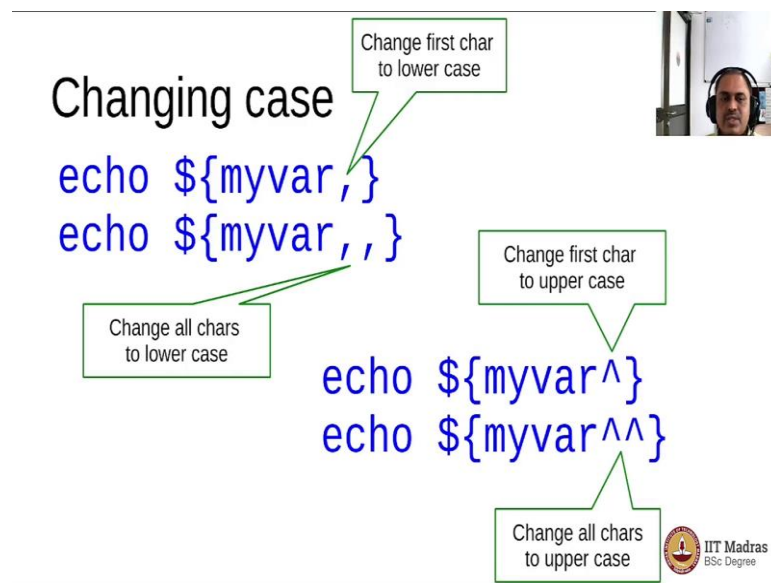


And sometimes you may want to match a pattern and replace that pattern with another string. So, it is like search and replace but performed on a single string, which is the value of a variable here in this case it is myvar. So, if you use a forward slash once, it means the matching is done only once. And if you use the forward slash twice, it means that the pattern

is matched as many times as possible within the string. And then all those patterns will be replaced with the string that is given for the replacement.

You could also do the pattern matching and replacing going by the location of the string. So, if you want to do the pattern matching only at the beginning of the string, you could use a pound symbol or a hash symbol after the forward slash. And if you would like to have the matching done only at the end of a string, then you could use the percentage symbol after the forward slash.

(Refer Slide Time: 8:34)



**Changing case**

```
echo ${myvar,}  
echo ${myvar,,}
```

Change first char to lower case

Change all chars to lower case

```
echo ${myvar^}  
echo ${myvar^^}
```

Change first char to upper case


Change all chars to upper case

IIT Madras  
BSc Degree

Now, you can display the value of a variable by converting the case of the characters within the string. And you could do it for the first character or for all the characters to the lowercase or uppercase. As you can see from here, the comma symbol indicates that it is the lowercase that has to be printed out. The caret symbol indicates that it is the uppercase that has to be printed out the original value of the variable is not changed, only the display will be modified as you have asked with these trigger commands within the braces.

(Refer Slide Time: 9:10)

## Restricting value types



```
declare -i myvar
```

Only integers assigned

```
declare -l myvar
```


Only lower case chars assigned

```
declare -u myvar
```

Only upper case chars assigned

```
declare -r myvar
```

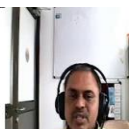
Variable is read only



You can restrict what type of values can be assigned to any variable. So, you can restrict it to be only integer type, or lowercase type or uppercase type or even read only by using the options like i, l, u and r. Remember that if you make a variable Read Only, then you cannot modify it after that. And if you want to do it, you may have to start the bash all over again so that it will forget that particular variable.

(Refer Slide Time: 9:41)

## Removing restrictions



```
declare +i myvar
```

integer restriction removed

```
declare +l myvar
```

lower case restriction removed

```
declare +u myvar
```

upper case restriction removed

```
declare +r myvar
```

Can't do once it is read only

If you want to remove the restrictions after having given those to a particular variable, you could do it by changing the option from minus to plus. So, plus i would actually remove the restrictions of integers to be assigned to the variable myvar. So, plus l is removing the restriction of the lowercase or the variable myvar.

Plus, u to remove the restriction on only uppercase to be assigned for the variable myvar and plus r cannot be done, because once you have already given a minus r attribute, then myvar is actually read only and you cannot change the attribute including the restriction of read only.

(Refer Slide Time: 10:18)

## Indexed arrays

```
declare -a arr
$arr[0]="value"
echo ${arr[0]}
echo ${#arr[@]}
echo ${!arr[@]}
echo ${arr[@]}
unset 'arr[2]'
arr+=("value")
```

declare `arr` as an indexed array

set value of element with index 0 in the array

Value of element with index 0 in the array

Number of elements in the array

Display all indices used

Delete element with Index 2 in the array

Display values of all elements of the array

Append an element with a value to the end of the array

Variables can also be stored as arrays within the shell environment, which is very powerful, because these operations are quite fast. So, you can declare a particular name as an array using the minus small a, if you wanted it as an index array. So, you could do various operations such as assigning the value to a particular element of the array.

And remember that unlike the arrays in C language or C++, the indices need not be contiguous, you could actually use any integer index for the indexed arrays. You can print the value of any specific element of the array by using the syntax of square brackets, which is quite common for you if you are familiar with the other programming languages.

By providing a hash in front of the array name, and an add symbol within the square brackets, you are conveying that you would like the command to be interpreted for all the elements that are in the array. And by hash you are asking to count. So, you would get the output as the number of elements in the array.

And the exclamation mark would indicate that you are interested in knowing what are the indices that are used by the array. And the add symbol tells that you would like to have the list of all the indices used in the array. And you could also list out all the variables the values of the variables stored in the array by simply using the syntax as given here, dollar and within

the braces array name and in the square brackets, you give an add symbol which is like a wild character to run through all the elements of the array.

And you can actually knock off any specific element in the array by providing its index value to the unset command. And you can also append an element to the array by using the syntax similar to the C language. So, plus is equal to would actually append the value at the end of the array. And when you are appending the index will be taken appropriately.

(Refer Slide Time: 12:27)

## Associative arrays

```
declare -A hash
$hash["a"]="value"
echo ${hash["a"]}
echo ${#hash[@]}
echo ${!hash[@]}
echo ${hash[@]}
unset 'hash["a"]'
```

declare hash as an associative array

set value of element with index "a" in the array

Value of element with index "a" in the array

Number of elements in the array

Display all Indices used

Delete element with Index "a" in the array

Display values of all elements of the array

Associative arrays or hashes are also available within the bash environment and these are very powerful because the index can be just anything, it can be an integer or a string. So, here is an example of a hash declared by using the option minus A, capital A and the index 'a' as a character is used to store an element with the value as given here.

And you could do most of the operations similar to the associative arrays except the appending operation, which is not possible because there is nothing called at the end of the array, because there is no sequence part of the indices that are used in a hash.

(Refer Slide Time: 13:10)



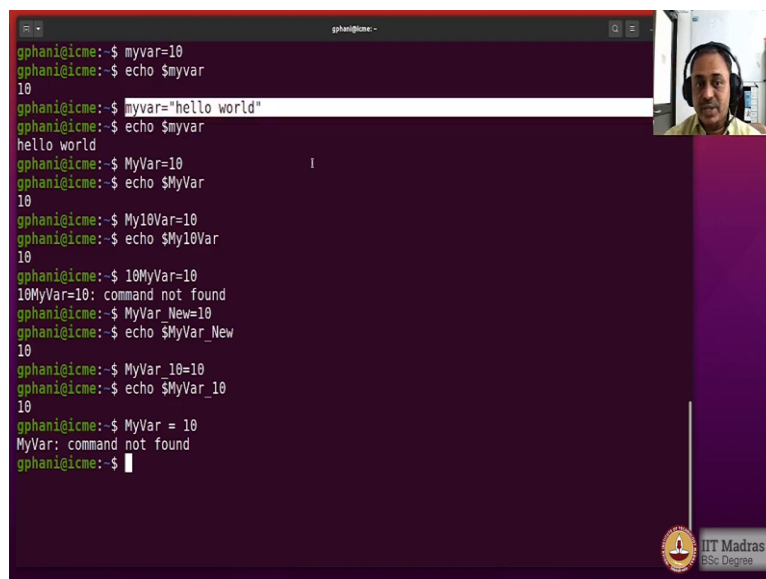
*Shell variable manipulations are fast !*



So, please remember that the shell variable manipulations are very fast. And therefore, if you are able to perform an operation using shell variable manipulations, then your script will be running very fast. So, mastering these options will be very valuable to you when you write your scripts.

This session is about shell variables, we will learn how to create shell variables, inspect them, edit them for display, also using command output to store as a value in a shell variable. We will also learn how to use an array of variables in the shell environment, let us get started with the demo.

(Refer Slide Time: 13:49)



```
gphani@icme:~$ myvar=10
gphani@icme:~$ echo $myvar
10
gphani@icme:~$ myvar="hello world"
gphani@icme:~$ echo $myvar
hello world
gphani@icme:~$ MyVar=10
gphani@icme:~$ echo $MyVar
10
gphani@icme:~$ My10Var=10
gphani@icme:~$ echo $My10Var
10
gphani@icme:~$ 10MyVar=10
10MyVar=10: command not found
gphani@icme:~$ MyVar_New=10
gphani@icme:~$ echo $MyVar_New
10
gphani@icme:~$ MyVar_10=10
gphani@icme:~$ echo $MyVar_10
10
gphani@icme:~$ MyVar = 10
MyVar: command not found
gphani@icme:~$
```



So, the way to create a shell variable is directly to use the name let us say I will create a variable called myvar and then equals and then we can give the value. It can be numerical, or a string. So, I will choose the value to be numerical, namely 10. To inspect the value of this variable, we would use the command echo and the dollar sign followed by the name of the variable.

We could actually overwrite the value stored using any other type. So, here I store a string in the variable myvar. And you could inspect the value of the variable myvar and you see that it has changed. Now, what kind of a restrictions are there in assigning values for variables in the shell environment?

It will be evident you could actually make the case for the shell variable names. So, you could have something like this myvar is equal to 10. So, you could mix the cases that is fine. You could also use digits within the name like this, you can see that you can use both alphabetical as well as numeric characters in giving the names. However, there is a restriction you cannot start the name of a variable using a digit.

So, try that out and you see that it is actually going into a little bit of a evaluation and then conses that it is actually a command and then therefore, such a command does not take this. So, it does not actually create a variable by the name 10Myvar. So, you cannot start the name of a variable with digits.

If you want you can actually use the underscore as a character to make your variables readable. So, here he would then see that, you can actually display the value of the variable Myvar underscore New, underscore is a poor man's way of representing a space for easy readability. You could also use numbers for that purpose and you could see that it is all working quite fine.

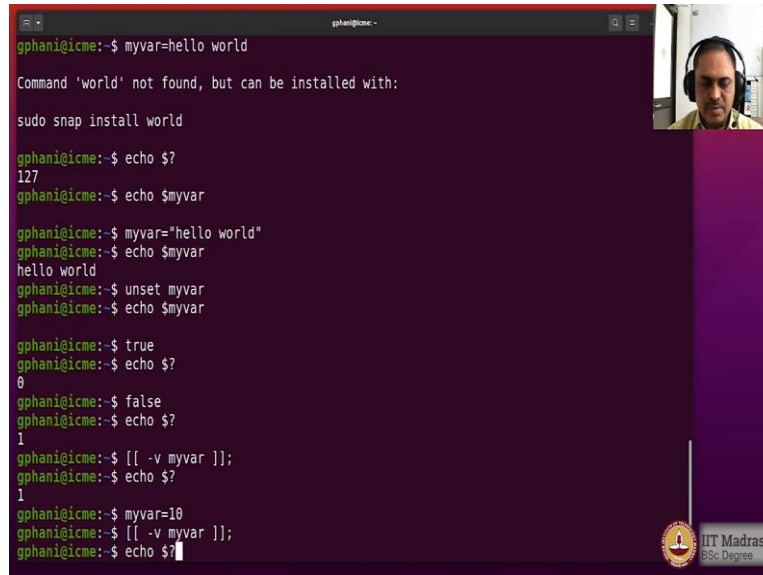
So, the lesson is that you could have the variables named either in small case or uppcase, you can also mix them up, you could use underscore as a part of the name, you could also use digits, but only thing is that you cannot start the name of a variable using a digit. Now, while assigning the value is when the variable also is getting created. But the equal to sign should not be surrounded by space for you to write in a legible fashion.

So, consider this for example. So, that will actually not work because Myvar is then interpreted as a command because there is a space after that. So, you should not have any space after the variable name, if you are going to assign a value and create that variable in the



process of writing that command. Variable that we have used here I have used a quotes now what happens if I do not use a quotes?

(Refer Slide Time: 17:32)



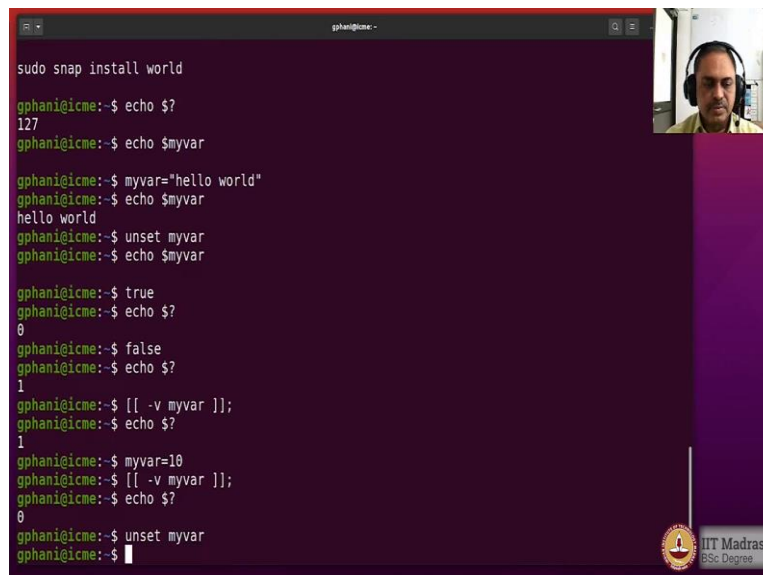
```
gphani@icme:~$ myvar=hello world
Command 'world' not found, but can be installed with:
sudo snap install world

gphani@icme:~$ echo $?
127
gphani@icme:~$ echo $myvar

gphani@icme:~$ myvar="hello world"
gphani@icme:~$ echo $myvar
hello world
gphani@icme:~$ unset myvar
gphani@icme:~$ echo $myvar

gphani@icme:~$ true
gphani@icme:~$ echo $?
0
gphani@icme:~$ false
gphani@icme:~$ echo $?
1
gphani@icme:~$ [[ -v myvar ]];
gphani@icme:~$ echo $?
1
gphani@icme:~$ myvar=10
gphani@icme:~$ [[ -v myvar ]];
gphani@icme:~$ echo $?
```

The terminal window shows a series of commands and their outputs. The first command is `myvar=hello world`, which results in a return code of 127. The second command is `echo $myvar`, which results in an empty output. The third command is `myvar="hello world"`, which results in a return code of 0. The fourth command is `echo $myvar`, which results in the output `hello world`. The fifth command is `unset myvar`, which results in a return code of 0. The sixth command is `echo $myvar`, which results in an empty output. The seventh command is `true`, which results in a return code of 0. The eighth command is `echo $?`, which results in the output `0`. The ninth command is `false`, which results in a return code of 1. The tenth command is `echo $?`, which results in the output `1`. The eleventh command is `[[ -v myvar ]]`, which results in a return code of 1. The twelfth command is `echo $?`, which results in the output `1`. The thirteenth command is `myvar=10`, which results in a return code of 0. The fourteenth command is `[[ -v myvar ]]`, which results in a return code of 0. The fifteenth command is `echo $?`, which results in the output `0`.



```
sudo snap install world

gphani@icme:~$ echo $?
127
gphani@icme:~$ echo $myvar

gphani@icme:~$ myvar="hello world"
gphani@icme:~$ echo $myvar
hello world
gphani@icme:~$ unset myvar
gphani@icme:~$ echo $myvar

gphani@icme:~$ true
gphani@icme:~$ echo $?
0
gphani@icme:~$ false
gphani@icme:~$ echo $?
1
gphani@icme:~$ [[ -v myvar ]];
gphani@icme:~$ echo $?
1
gphani@icme:~$ myvar=10
gphani@icme:~$ [[ -v myvar ]];
gphani@icme:~$ echo $?
```

The terminal window shows a series of commands and their outputs. The first command is `sudo snap install world`, which results in a return code of 0. The second command is `echo $?`, which results in the output `0`. The third command is `echo $myvar`, which results in an empty output. The fourth command is `myvar="hello world"`, which results in a return code of 0. The fifth command is `echo $myvar`, which results in the output `hello world`. The sixth command is `unset myvar`, which results in a return code of 0. The seventh command is `echo $myvar`, which results in an empty output. The eighth command is `true`, which results in a return code of 0. The ninth command is `echo $?`, which results in the output `0`. The tenth command is `false`, which results in a return code of 1. The eleventh command is `echo $?`, which results in the output `1`. The twelfth command is `[[ -v myvar ]]`, which results in a return code of 1. The thirteenth command is `echo $?`, which results in the output `1`. The fourteenth command is `myvar=10`, which results in a return code of 0. The fifteenth command is `[[ -v myvar ]]`, which results in a return code of 0. The sixteenth command is `echo $?`, which results in the output `0`.

Let us see what happens when we assign a value to a variable with a space in that particular string. So, here it has failed. So, how do I know that? I can inspect the return status and it says there is a problem there. So, 127 means that the return code says that the command was not found.

So, if I now try to inspect what would be the value of the variable, and you will see that there is a null which means that entire command was neglected. And if we did not end up in assigning value to the variable `myvar`. Now, we can accomplish that correctly by enclosing the space of the string not to be interpreted as a delimiter. So, the entire string has to be



enclosed in quotes. So, you could now see the value of the variable and it comes out quite fine.

Now, for some reason, you would like to remove this variable. So, you could use a command called unset. So, if you say unset myvar, then the variable is removed from the memory and you could just try it out. And you see that there is a null so which means that the variable is no longer there in the memory.

Now, you will know that the true and false used as 0 and 1 respectively, can also be used to check whether a particular variable is present in the system or not. So, let us try that out by using a construction which will be put to use in shell scripts shortly. But for now, we can actually already try it on the command line.

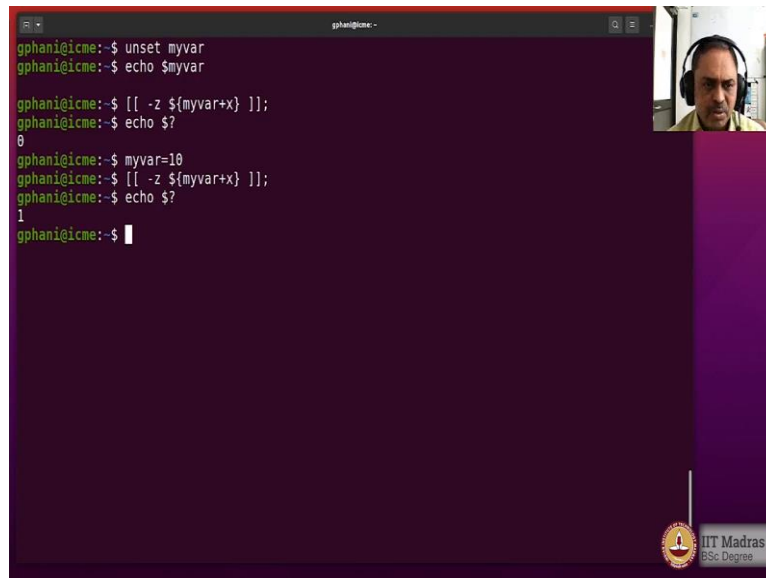
So, the syntax is as follows. Let us say I want to know whether there is any variable by name myvar. So, what I would do is, first let us check what is this command true and you see that it is always returning the success state. So, if I type echo dollar percent mark or return status, it says 0 which means it is true, is always returning a success state and if I run a command false, then you will see that the return code is 1.

So, we can now use this as a way to check whether the variable is present or not by doing a check, so, we will do that by using a particular construction. So, minus v and then the name of the variable. So, if you do this it would run and then the return code will tell whether that variable was present or not.

You see that the return value is 1 which means that the output of this particular construction that we have done the test condition gave false which means that myvar is not existing. Now, we do the same thing by creating the variable now myvar is equal to and let us say 10 and then run the same code which is basically to test for the variable being present or not.

And then echo dollar question mark and you see that the code has changed to true which means that minus v myvar is a way by which we can check whether the variable is present in the memory or not. You could also test it in a different manner that I will illustrate now, first I will remove the variable.

(Refer Slide Time: 21:21)



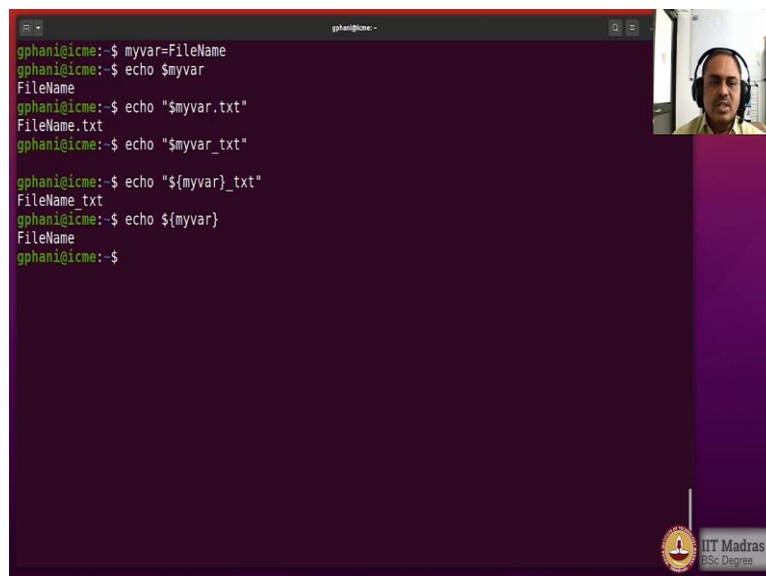
```
gphani@icme:~$ unset myvar
gphani@icme:~$ echo $myvar

gphani@icme:~$ [[ -z ${myvar+x} ]];
gphani@icme:~$ echo $?
0
gphani@icme:~$ myvar=10
gphani@icme:~$ [[ -z ${myvar+x} ]];
gphani@icme:~$ echo $?
1
gphani@icme:~$
```

The terminal window shows a series of commands to unset a variable and test its presence using the `-z` flag. The first test with `unset myvar` returns 0, indicating the variable is not set. The second test with `myvar=10` returns 1, indicating the variable is set. A small video inset of a person is visible in the top right corner.

So, there is another way by which we can test for the variable this is by using the minus z test. So, first I will remove the variable that I want to check first so that I am sure that such a thing does not exist. So, also verify it does not exist. So, now we do the test minus Z dollar the variable name variable name is myvar and then plus any string which will be used as a replacement in case the variable was not present okay. Now, you see that the test has resulted in 0. Now, the test has resistant false. This is also one more way by which you can go about checking whether the variable is present or not, but this is a little bit roundabout.

(Refer Slide Time: 22:24)



```
gphani@icme:~$ myvar=FileName
gphani@icme:~$ echo $myvar
FileName
gphani@icme:~$ echo "${myvar}.txt"
FileName.txt
gphani@icme:~$ echo "${myvar}_txt"
FileName_txt
gphani@icme:~$ echo "${myvar}_txt"
FileName_txt
gphani@icme:~$ echo ${myvar}
FileName
gphani@icme:~$
```

The terminal window shows a series of commands to assign a value to a variable and test its presence using different bracket notations. The variable `myvar` is assigned the value `FileName`. The tests show that `${myvar}.txt` and `${myvar}_txt` both result in `FileName.txt`, while `${myvar}` results in `FileName`. A small video inset of a person is visible in the top right corner.

Now, you have seen that I have used the braces or flower brackets to enclose the variable name. So, this is for a particular purpose I will illustrate that to you now. So, myvar is equal

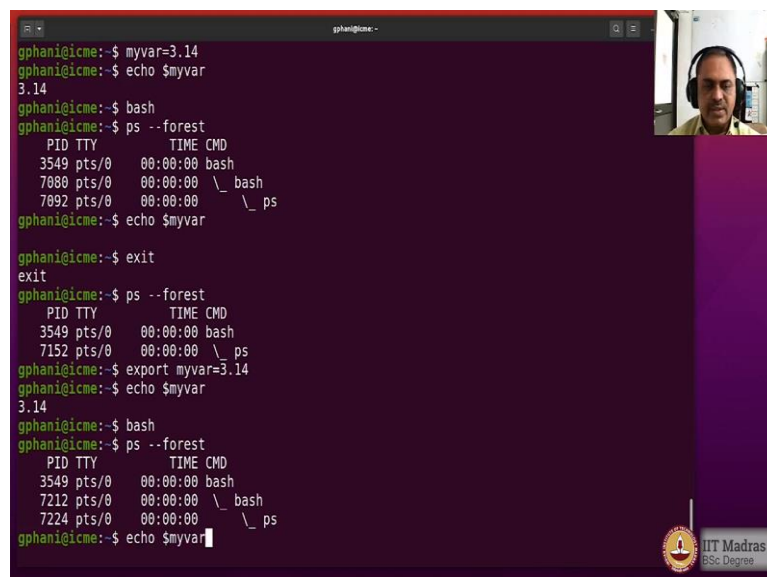
to let us say a file name so, I will have it as a string. Now, echo dollar myvar would display that on the screen. Now, I would like to display a file name with an extension as a part of some program.

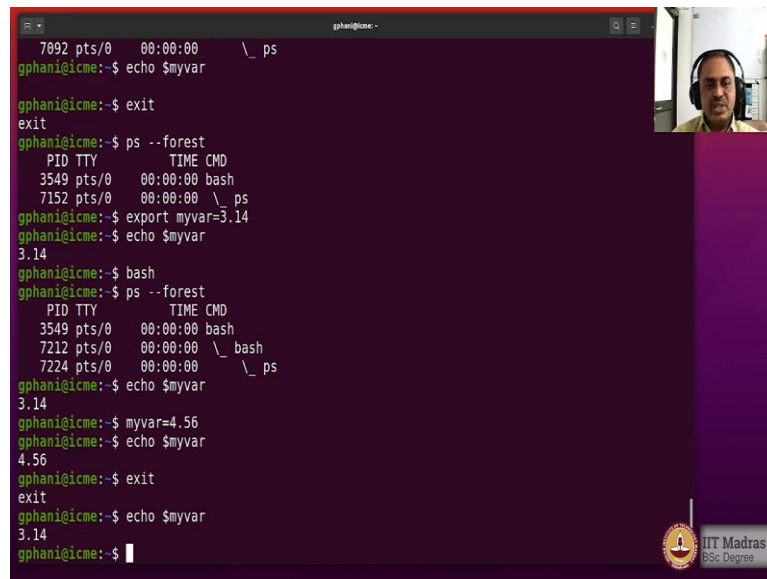
So, let us say I want to do it this manner. So, you see that the extension has come. So, dot txt has come because it has not been interpreted as a part of the variable name. Now, I would like to have an underscore there and you see what would happen. So, I was expecting now file name underscore txt to come and it does not come because now the interpretation is that myvar underscore txt is the entire name of the variable which actually does not exist.

So, how do we overcome, this is where the braces will come up to us to tell clearly what is the name of the variable and what is not by using those braces. So, what I would do now here is in this manner. So, now you see that it is becoming clear that myvar is the name of the variable and after that the rest of it is part of the string that I want to happen. And now you see that it behaves as we originally wanted.

So, here is an illustration to tell you that the flower brackets or braces are very useful in stating clearly what is the name of the variable and whatever comes after that will be used as part of the string. Now, this can also be used outside of the quotes. So, you could also do this way and that works quite fine as we have seen in the previous example.

(Refer Slide Time: 24:38)

A terminal window with a dark purple background and green text. The user 'gphani@icme' is running several commands. First, they set 'myvar=3.14' and then 'echo \$myvar' which outputs '3.14'. Next, they run 'bash' to start a subshell, then 'ps --forest' which shows a tree of processes: PID 3549 (bash) has child PID 7080 (bash), which has child PID 7092 (ps). Then they run 'echo \$myvar' which still outputs '3.14'. They then type 'exit' to leave the subshell. Back in the main shell, 'ps --forest' shows PID 3549 (bash) with child PID 7152 (ps). Then they run 'export myvar=3.14' and 'echo \$myvar' which outputs '3.14'. Finally, they run 'bash' again, then 'ps --forest' showing PID 3549 (bash) with child PID 7212 (bash), which has child PID 7224 (ps). The last command is 'echo \$myvar'. In the top right corner, there is a small video feed of a man wearing headphones. In the bottom right corner, there is a logo for 'IIT Madras BSc Degree'.

A terminal window titled 'gphani@gnome' with a dark purple background. The terminal shows a series of commands and their outputs. At the top, a 'ps' command shows the current process. Then, 'echo \$myvar' outputs '3.14'. The user enters 'exit', and the prompt changes to 'exit'. Then, 'ps --forest' shows a tree of processes, including 'bash' and '\\_ ps'. The user enters 'export myvar=3.14', and 'echo \$myvar' outputs '3.14'. The user enters 'bash', and the prompt changes to 'gphani@icme:~\$'. Then, 'ps --forest' shows the new process tree, including 'bash' and '\\_ bash'. The user enters 'echo \$myvar', and the output is '3.14'. Then, 'myvar=4.56' is entered, and 'echo \$myvar' outputs '4.56'. Finally, 'exit' is entered, and the prompt returns to 'gphani@icme:~\$'. The terminal also shows a small video feed of a person in the top right corner and an IIT Madras logo in the bottom right corner.

```
7092 pts/0    00:00:00  \_ ps
gphani@icme:~$ echo $myvar
3.14
gphani@icme:~$ exit
exit
gphani@icme:~$ ps --forest
PID TTY          TIME CMD
3549 pts/0    00:00:00  bash
7152 pts/0    00:00:00  \_ ps
gphani@icme:~$ export myvar=3.14
gphani@icme:~$ echo $myvar
3.14
gphani@icme:~$ bash
gphani@icme:~$ ps --forest
PID TTY          TIME CMD
3549 pts/0    00:00:00  bash
7212 pts/0    00:00:00  \_ bash
7224 pts/0    00:00:00  \_ ps
gphani@icme:~$ echo $myvar
3.14
gphani@icme:~$ myvar=4.56
gphani@icme:~$ echo $myvar
4.56
gphani@icme:~$ exit
exit
gphani@icme:~$ echo $myvar
3.14
gphani@icme:~$
```

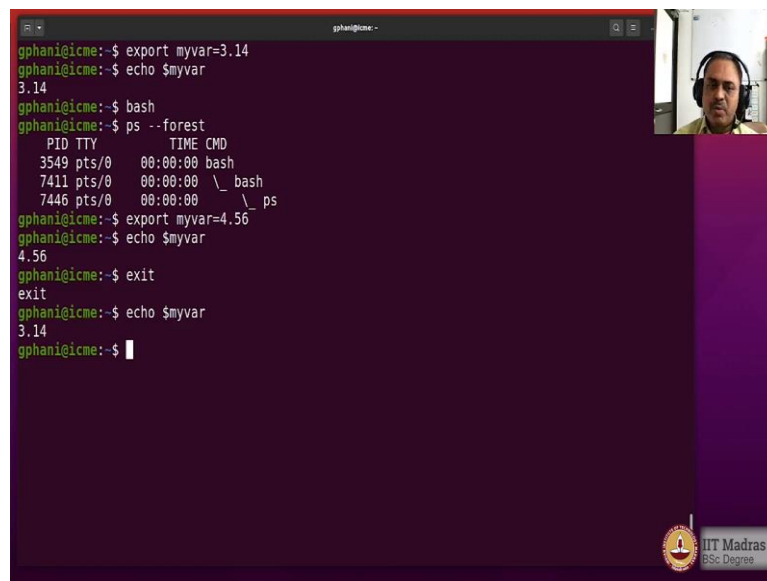
Now, let us see whether this variable that we have created, does it get passed on to the shell or any other program that has been launched within the shell? So, what I would do is myvar is equal to let us say 3.14. So, a variable name is myvar and it is holding a value 3.14. Now, what I will do is I will go down into one more level of bash and ps minus minus forest and you will see that we are actually one level below.

And here I would like to check whether this variable was present, and you see that it is not present. So, which means that the variable that is created in any shell is not automatically passed on to a sub shell. So, what do I do to make sure that it is passed on? So, for that, let me come back to the parent shell and confirm that we are in the parent shell. And what we do is we use a command export to ensure that this variable is then made available to all the children that are spawned by this particular shell.

Verify that variable is available. And now I launch one sub shell. And in that we verify that we are in the sub shell. And in that I check whether this particular variable is present. And you can see that it is now available. So, this shows that when you use export, then the variable is available to the child shell.

Now, what happens when we modify the variable within the child shell, so, that I would do by saying myvar is equal to 4.56 echo my var. Now, I come back to the parent shell and look at the variable. And you see that it remains same as what we originally have set, which means that when a variable is exported, then it is available for the child shell, but when the child shell modifies the value of the variable, that modification is not reflected in the value of the variable in the parent shell. So, it is only then modified within the local environment, we could also verify that by using the export within the child shell also.

(Refer Slide Time: 27:06)



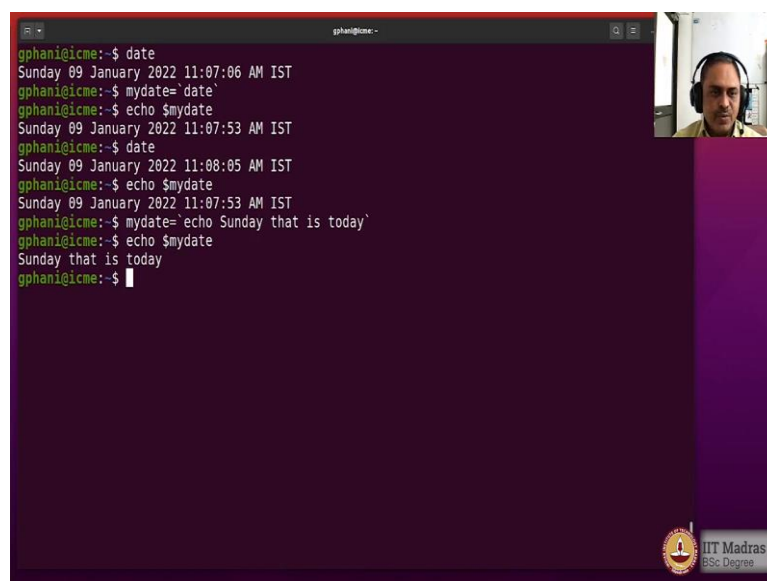
```
gphani@icme:~$ export myvar=3.14
gphani@icme:~$ echo $myvar
3.14
gphani@icme:~$ bash
gphani@icme:~$ ps -forest
PID TTY      TIME CMD
 3549 pts/0    00:00:00 bash
 7411 pts/0    00:00:00  \_ bash
 7446 pts/0    00:00:00      \_ ps
gphani@icme:~$ export myvar=4.56
gphani@icme:~$ echo $myvar
4.56
gphani@icme:~$ exit
exit
gphani@icme:~$ echo $myvar
3.14
gphani@icme:~$
```

The terminal window shows a user named gphani@icme. They first export a variable myvar to 3.14 and echo it. Then they spawn a new bash shell. In this child shell, they run ps -forest to show the process tree, then export myvar to 4.56 and echo it. Finally, they exit the child shell and echo myvar in the parent shell, which still shows 3.14. An IIT Madras BSc Degree logo is in the bottom right corner.

So, we do the same thing again myvar is equal to 3.14 and check that out bash and here again, what we would do is export myvar is equal to echo 4.56. And now I come back to the parent shell, echo myvar, and you see that even though you did export within the child shell, it has not changed the value of the variable in the parent shell.

So, which means that the parent shell is holding the value of the variable and though it is exported and getting modified by the child shell it is not (at) affecting the value within the parent shell. This is something that is very useful in some of the compiler flags that we may be using in compiling large packages when we use open source software.

(Refer Slide Time: 28:17)



```
gphani@icme:~$ date
Sunday 09 January 2022 11:07:06 AM IST
gphani@icme:~$ mydate='date'
gphani@icme:~$ echo $mydate
Sunday 09 January 2022 11:07:53 AM IST
gphani@icme:~$ date
Sunday 09 January 2022 11:08:05 AM IST
gphani@icme:~$ echo $mydate
Sunday 09 January 2022 11:07:53 AM IST
gphani@icme:~$ mydate='echo Sunday that is today'
gphani@icme:~$ echo $mydate
Sunday that is today
gphani@icme:~$
```

The terminal window shows a user named gphani@icme. They first echo the date. Then they create a variable mydate with the value 'date' and echo it. Then they spawn a new bash shell. In this child shell, they run date, then echo mydate, then modify mydate to 'echo Sunday that is today' and echo it. Finally, they exit the child shell and echo mydate in the parent shell, which still shows the original date. An IIT Madras BSc Degree logo is in the bottom right corner.

Now, we have seen that there are many commands that would give you output and we would like to have them stored as variables. So, that is something that is accomplished by using what is called the back quote, which is also the character that comes if you use the tilde key in the US keyboard layout. So, I will just to demonstrate that to you now.

So, the command `date` would give you the date output. So, if you have a variable created, `mydate` is equal to, now you see the quotation symbol that I have used here, it is a single code but a back quote, it is not to be confused with the single code which is straight up are the double code which is appearing with two dashes. So, it is a single quote, but appearing below the tilde symbol on the keyboard. So, it is called the back quote.

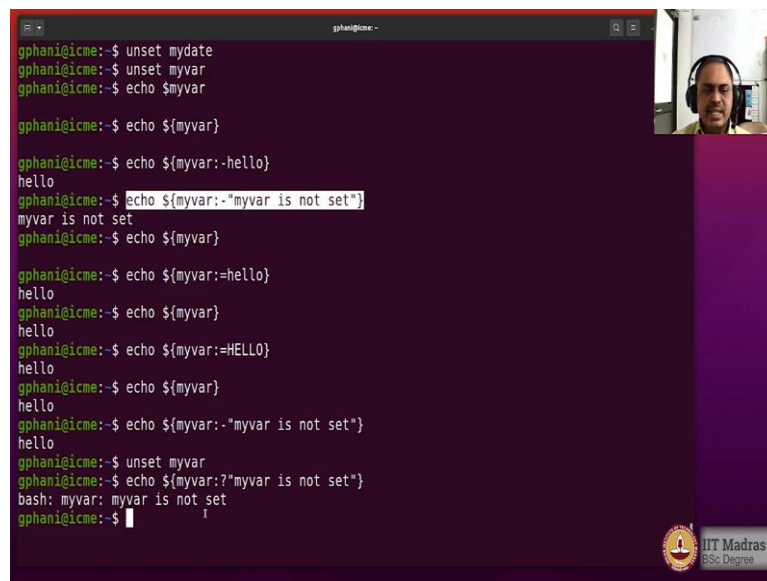
So, the back quote is used for a very specific purpose, the output of the command is then used as a value for the particular variable which you are assigning. So in this case, `mydate` is the name of the variable and the value of that variable will be the output given by the `date` command. So, let us just check that out.

So, you see that the variable `my date` has a string which is basically the `date` command output. Now, the `date` command output has changed now because the time is now 11:08. But, if you `echo` them, it is still showing 11:07 which means if it is not dynamic, it is only inserted as a value at the instance that the command is run and after that it remains a string alone. So, it is not a dynamically updated one.

You could also combine the `echo` command along with this particular back quote. So, let us do that here. So, you see that the `mydate` variable is now containing the string, which is the output of the command `echo`, which I have written here. So, whatever is the command that will be executed and that would then be used as the value of the variable.

Now, this is quite useful because we could then now store the string and manipulate it using the means that are available for manipulation of value of a variable within the shell environment because now that you have what a string then you can manipulate it the way you like. So, now, let us see what are the kinds of manipulations that are available for variables within the BASH environment.

(Refer Slide Time: 31:06)



```
gphani@icme:~$ unset mydate
gphani@icme:~$ unset myvar
gphani@icme:~$ echo $myvar

gphani@icme:~$ echo ${myvar}

gphani@icme:~$ echo ${myvar:-hello}
hello
gphani@icme:~$ echo ${myvar:-"myvar is not set"}
myvar is not set
gphani@icme:~$ echo ${myvar}

gphani@icme:~$ echo ${myvar:=hello}
hello
gphani@icme:~$ echo ${myvar}
hello
gphani@icme:~$ echo ${myvar:=HELLO}
hello
gphani@icme:~$ echo ${myvar}
hello
gphani@icme:~$ echo ${myvar:-"myvar is not set"}
hello
gphani@icme:~$ unset myvar
gphani@icme:~$ echo ${myvar:? "myvar is not set"}
bash: myvar: myvar is not set
gphani@icme:~$
```

So, I am just cleaning up the memory and I want to now show you how we go about manipulating the variables within the shell environment. So, let us say that we have a command where we would like to use a value if the particular variable does not exist, for some reason it was not set, but you still would like to go ahead and have that as a part of your script.

So, that can be accomplished by certain construction within the braces and that is the beginning of various manipulations that are possible. So, we will go step by step. So, first is that if there is any variable you want to print its output, you would put it as dollar myvar and you could actually have the braces to indicate that the variable is now, you can now start becoming comfortable by using the flower brackets for the variable name because there are a lot of characters that we will be inserting within the power brackets because they are all what will actually manipulate the value.

So, the very first manipulation is the variable is not actually available, but we would like to have the echo command display something which is like a default value in case the variable is not available. So, this can be achieved as follows. So, a colon is very popularly used to perform some operations, so we have to get used to that. So, after colon a bunch of characters will tell what operation we are doing.

So, there is a minus that is there so, which is sort of indicating that if the value is not present, then what should be the value that should be displayed. So, the display value let us say is hello. Now, you see that you have got some output on the screen and output is the default string that has been mentioned within the braces, you could actually have any other string

also, and you could also use the quotation symbols to contain it. So, you can say that myvar is not set.

So, now you can see that the echo command is a little bit more useful. If you have done only this, then the output is null and it is not clear whether the variable was present or not, or was it actually set with a blank character. So, that was not clear. But, if you see the way we have written here up, then it is a bit more useful because there is a particular string that is displayed as a message on the screen in case the variable was not set.

So, this is a way by which you can give default values to variables in the process of using them in case you are not sure whether they are set or not. Now, here you see that even though we have displayed a string, it is not yet in the variable because as you can see from this command, that it is still empty, which means that to the myvar is not yet set, we have only displayed a particular string, but it is not set.

But, if you want to actually set the value if it was not set already, then we could do this way colon equals and hello. And now you see that the echo command is giving you the default value, but at the same time the variable also has been set. So, earlier it was not set, but now it is said because the colon equal to is indicating that if the variable myvar is not set, then set it to the value that is specified after the equal to symbol which is basically a string hello.

So, this is a very useful way by which we can actually ensure the default behavior to be what we want by specifying those default values for the variables. Now, if I run the code again what would happen; you see that the variable myvar is present and therefore, there is nothing that is going to happen to it and it is retaining the real value and you see that the capital HELLO has not come into the picture at all because it is only when the variable myvar is not set that any action is going to be performed after the colon symbol what is indicated there.

So, we will also do that in this command, for example here this command we will actually try that out here, so, you see that the message is not displayed. The value of the variable is displayed because it is set. So, only if it is unset, then the action after the colon will start taking into effect.

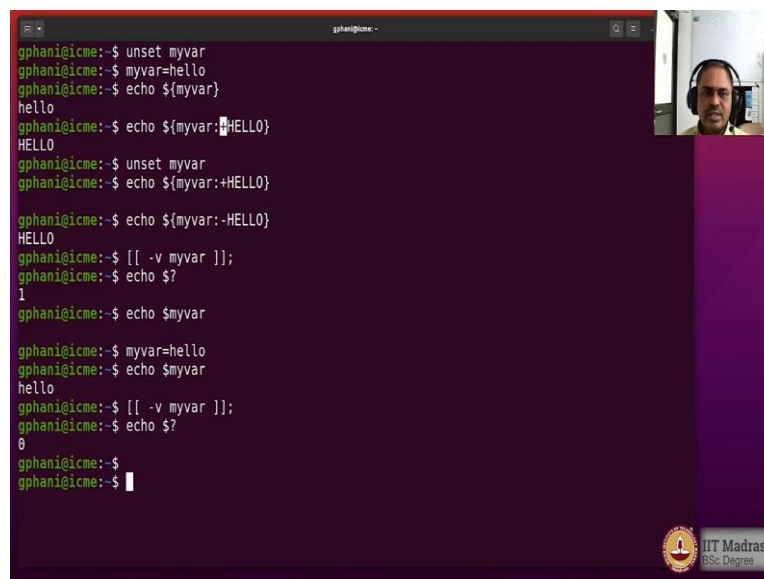
Now, you can also display a message on the screen if the variable was not set in the following manner. So, for that I would actually remove this variable echo dollar myvar colon question mark and then a string. So, you see that there is a display that is done and it is also giving you



a little bit more information. So, where it is actually telling about the particular variable and giving you a display of the debug message that you can put.

So, this is some way by which you are alerting that particular variable is not present. In what way is it different from this command if you see is that in this command the echo variable is behaving as if the value of that particular variable is what is in the string. But here, it is more clear that it is actually a message.

(Refer Slide Time: 37:13)



```
gphani@icme:~$ unset myvar
gphani@icme:~$ myvar=hello
gphani@icme:~$ echo ${myvar}
hello
gphani@icme:~$ echo ${myvar:+HELLO}
HELLO
gphani@icme:~$ unset myvar
gphani@icme:~$ echo ${myvar:-HELLO}
HELLO
gphani@icme:~$ [[ -v myvar ]];
gphani@icme:~$ echo $?
1
gphani@icme:~$ echo $myvar
hello
gphani@icme:~$ myvar=hello
gphani@icme:~$ echo $myvar
hello
gphani@icme:~$ [[ -v myvar ]];
gphani@icme:~$ echo $?
0
gphani@icme:~$
gphani@icme:~$
```

Now, there is a opposite action of the minus symbol which is basically to set value if it was actually present. So, let us try that out here. So, the variable is present. So, now colon plus and if it is present, then I want to do some particular action, so that I would have it as let us say capital HELLO and you see that the variable is displaying a particular string, which is basically if it is present.

Now, I would actually unset and then I try this command again and it is not working which means that only if myvar is present, then the action that is following with the plus sign will be done. So, that particular string will be displayed. But if it was not present, then nothing will be displayed. So, which is actually opposite of the minus symbol.

So, I just showed the same thing here, put a minus sign there, and you see that if the myvar is not present, then the string that is following the minus sign should be displayed. And that is what is being done. So, you can see that both the options are available the option to display if the variable is present option to display if the variable is absent, both are available and now it is upto your imagination to put it to a particular logical sequence of operations, when you

write your shell scripts by looking at the values of the particular variable, whether they are set or not.

Inspecting whether the values are set or not is also possible as I have illustrated earlier where we use the construction like this minus v and you can now see that the output would tell you whether it is present or not. So, one is for false so myvar is not available. And you see that it was not available and myvar is equal to hello, it is present now. So, you see that it is present and you could then also see that the status has changed to true.

So, you can always inspect it as a separate activity. But if you are, if you want to inspect the presence or absence of a particular variable and follow it up with a particular string that has to be displayed or used as the value of the string, then echo has a construction by which you have this possibility. So, the dollar brace construction after the colon is something that is very powerful.