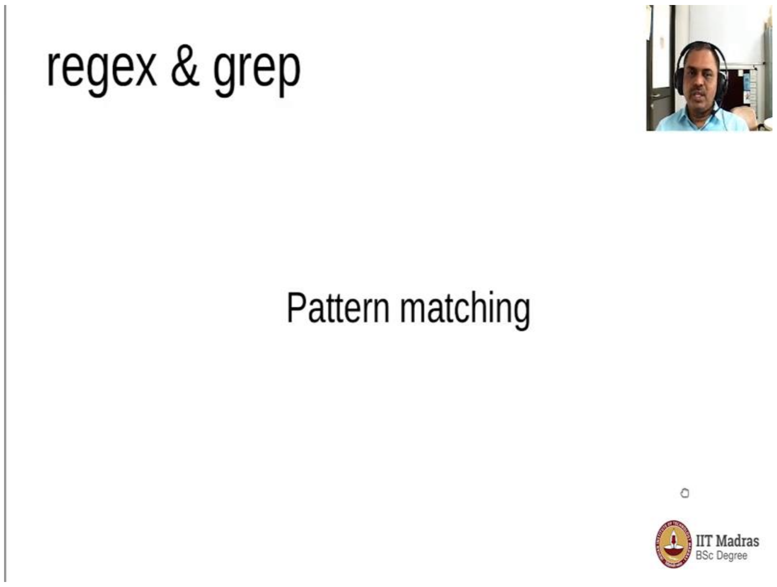
System Commands

Professor Gandham Phanikumar,
Metallurgical and Materials Engineering
Indian Institute of Technology Madras

Pattern Matching - Part 01

(Refer Slide Time: 00:14)



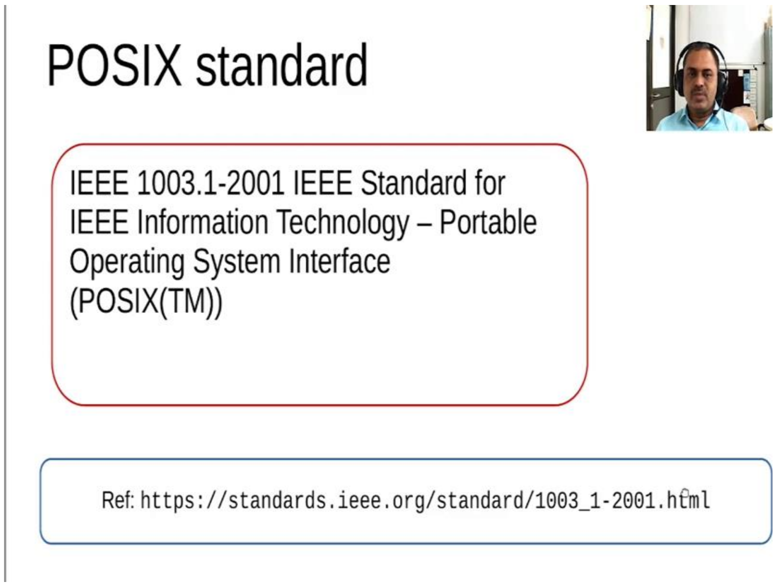
regex & grep

Pattern matching

IIT Madras
BSc Degree

Welcome to the session on regular expressions and the grep command. Pattern matching is what we would be discussing in this session. We will go through detailed demonstrations for you to understand the grep command thoroughly. Once you understand the concept of regular expressions, you can use the grep command to its full extent, once you understand the regular expressions, you can make the most out of the grep command and use the command line in a very powerful manner.

(Refer Slide Time: 00:48)



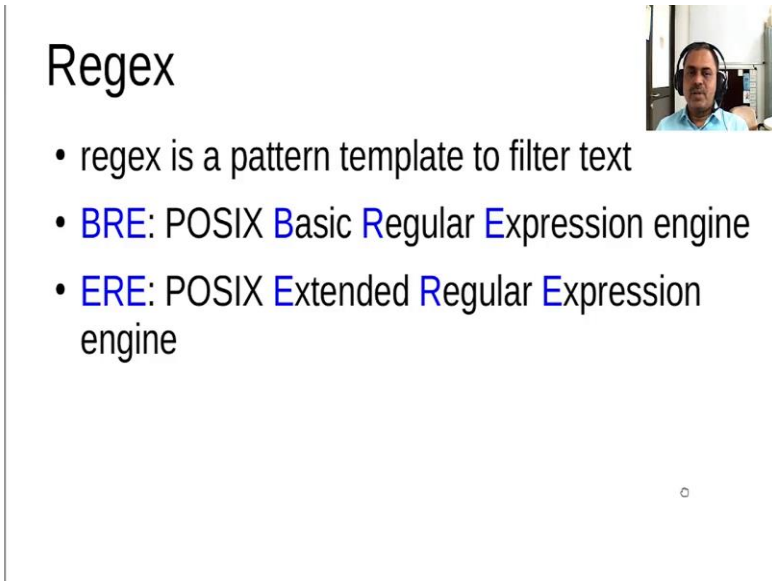
POSIX standard

IEEE 1003.1-2001 IEEE Standard for IEEE Information Technology – Portable Operating System Interface (POSIX(TM))

Ref: https://standards.ieee.org/standard/1003_1-2001.html

We will be discussing the grep command using the regular expressions that conform to the POSIX standard. There are many engines in different languages and tools that implement regular expressions with various options, but we would stick to the IEEE 1003.1 hyphen 2001, POSIX standard which defines the regular expressions to be of two different types, the one the basic and the extended, and we will limit our discussions to these two varieties.

(Refer Slide Time: 01:26)



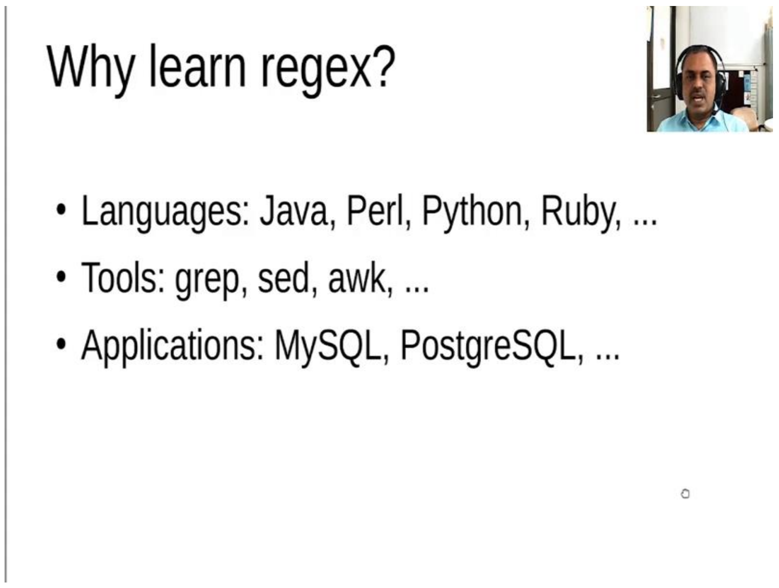
Regex

- regex is a pattern template to filter text
- BRE: POSIX Basic Regular Expression engine
- ERE: POSIX Extended Regular Expression engine

So, what is regular expression? It is basically a pattern template to filter text. So, as you have seen in the command line, some of the commands will give a lot of output and we would like to filter and extract certain features from that output. So, you would pipe the output of the command to grep command and then use a regular expression to pick the pattern that you are interested in. So, this is the basic utility that we would be focusing on.

However, you could also combine the grep command with any other command to extract the output in a format that you are interested in. And there are two types of regular expression engines, the BRE, which corresponds to the POSIX basic regular expression engine and the ERE which corresponds to POSIX extended regular expression engine. And the options for these two engines are slightly different and there are some common options as well. So we will look at them and use the grep command to use either BRE or ERE depending upon our requirement.

(Refer Slide Time: 02:31)



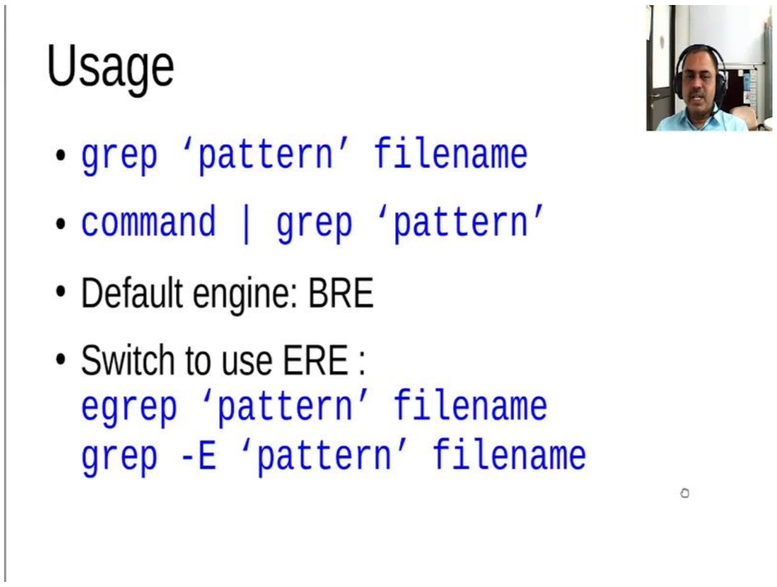
Why learn regex?

- Languages: Java, Perl, Python, Ruby, ...
- Tools: grep, sed, awk, ...
- Applications: MySQL, PostgreSQL, ...

So, why do we need to learn regular expression at all? In many languages like Java, Perl, Python, Ruby and so on, in many of the scripting languages you would require to process the input from the user or perform some string operations. So, regular expressions are what are used to perform these operations without being tied down to the actual value of the string so that the program is generic in nature. So, if you learn regular expressions, you can make the most out of the string manipulations in multiple languages.

And there are many tools in Linux, such as the grep, the sed and the awk, which actually allow you to make use of those language features to a larger extent, if you have understood regular expressions. There are also other independent applications such as the databases such as MySQL or PostgreSQL, which actually allow you to also take utility of the regular expressions, because they are used to process strings in those languages as well.

(Refer Slide Time: 03:41)



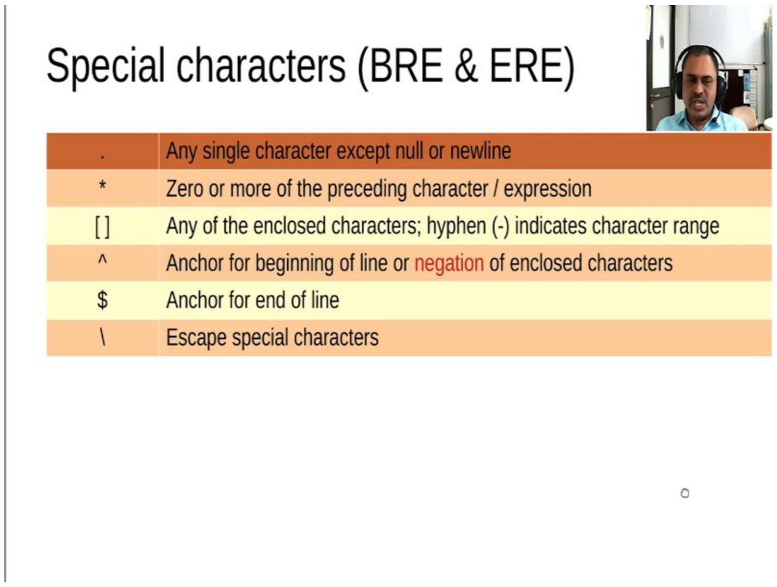
Usage

- `grep 'pattern' filename`
- `command | grep 'pattern'`
- Default engine: BRE
- Switch to use ERE :
  ```
  egrep 'pattern' filename
  grep -E 'pattern' filename
  ```

A typical usage of the grep command is basically to supply a file name so that each line in the file is processed using the grep command using the pattern that is provided as an option to the grep command. You could also launch the grep command to take input from a pipe. So, you could have a command the output can be piped to grep and then you provide the pattern and every line in the output would be processed by the grep command using the pattern that you have supplied.

Now, remember that the grep command operates line after line, which is a very common feature among many utilities on Linux. Now, by default, if you launch the command grep, it would use the basic regular expression engine. But if you want to use the extended set, then either you can use the command E grep or you could also give the option minus capital E to the grep command and then the pattern will be interpreted within the options that are available and extended under the extended regular expression engine.

(Refer Slide Time: 04:54)



Special characters (BRE & ERE)

.	Any single character except null or newline
*	Zero or more of the preceding character / expression
[ ]	Any of the enclosed characters; hyphen (-) indicates character range
^	Anchor for beginning of line or negation of enclosed characters
$	Anchor for end of line
\	Escape special characters

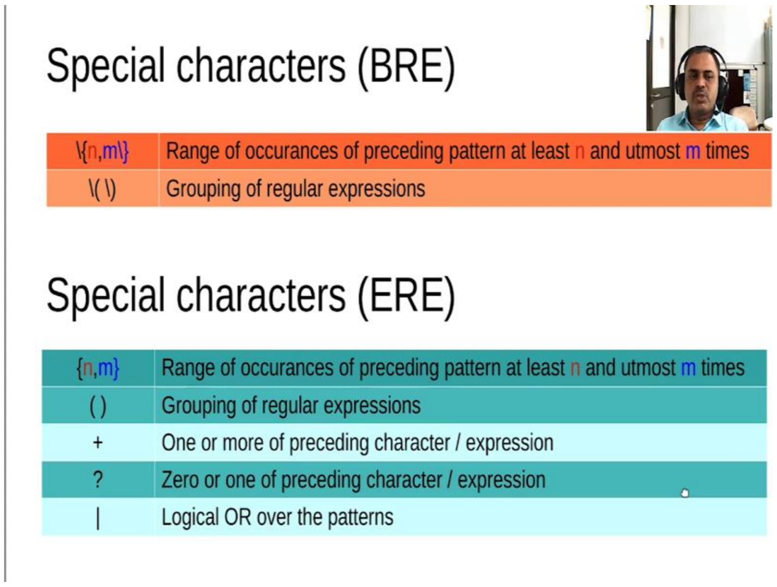
Now, there are some special characters with their meaning which we must understand. So, these are not necessarily the same as what are used by the shell to interpret when you write a command. And the way you can escape the special character interpretation by the shell and allow only the grep to interpret it as a part of the pattern is to enclose the pattern in single quotes and that would actually be very safe option. So, there are some grep commands where these single quotes are not necessary, but we will make it a habit so that we do not make a mistake later on where there is a conflict between the interpretation of a special character between the grep command and the shell which is passing on the options to the grep command.

The dot is used to match any single character except the null character which comes at the end of a string or the newline character which comes at the end of a line. Star is usually used in shell commands to interpret as many options for the files as possible. But in the case of grep, it is to match either 0 or more of the preceding character. So, if you have a character followed by star, then that character preceding the star would be matched with either 0 or more occurrences. And this can also be used along with an expression. So, any sub-expression, any sub-regular expression can be used with the star operator.

The square brackets are used to give a list of characters any one of them can match the string. And if you have a hyphen between two characters in the square brackets, it indicates a range. So the collating sequence of characters which is nc in the case of most of the installations, then you will actually see that A to Z or capital A to capital Z, or 0 to 1 can be used as ranges within this square bracket. You could also use a reduced range if it is required for you.

The hat character is used to indicate as an anchor for the beginning of the line. That means that the pattern which follows after the anchor will be matched only if it is occurring at the beginning of the line. And if the hat character occurs within the square brackets, it indicates that the list of characters in the square brackets should not be matched. It is like an opposite effect of the range that is provided in the square brackets. The dollar is an anchor to indicate that the matching of the button should be at the end of the line. And the backslash as usually also used in the shell, it is to escape special characters and you can escape the backslash itself by providing it twice.

(Refer Slide Time: 07:46)



Special characters (BRE)

\{n,m\}	Range of occurances of preceding pattern at least n and utmost m times
\(\)	Grouping of regular expressions

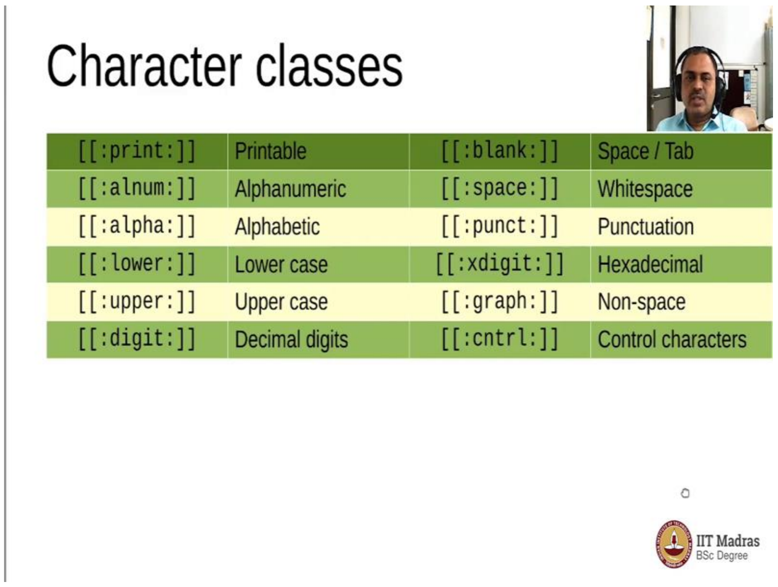
Special characters (ERE)

{n,m}	Range of occurances of preceding pattern at least n and utmost m times
( )	Grouping of regular expressions
+	One or more of preceding character / expression
?	Zero or one of preceding character / expression
|	Logical OR over the patterns

Now, there are ranges of occurrences for the patterns that you can also indicate. That is there is a pattern which you would like to have it either 2 times or not more than 4 times by giving 2 comma 4 as options within the braces. And in the case of a BRE, the braces should be preceded by the backslashes. And in the case of the extended regular expression sector, the braces need not be preceded with a backslash. Similarly, parenthesis can be used to group regular expressions. By grouping what we mean is that it can be followed with either the range of occurrences or the star operator, so that you can actually have a more complicated matching pattern.

Now, the parentheses do not require a preceding backslash in the case of ERE, but the function is the same. And there are also more options in ERE. The plus sign would indicate that the preceding character should be matched either once or more. And the question mark sign is to indicate the preceding character should be matched either 0 times or once. And the pipe character is used within the parentheses to indicate a logical OR operator over the patterns that are provided, but patterns need to be of the same length. And the plus question mark can be used also with sub-expressions and not necessarily only a single character.

(Refer Slide Time: 09:15)



Character classes

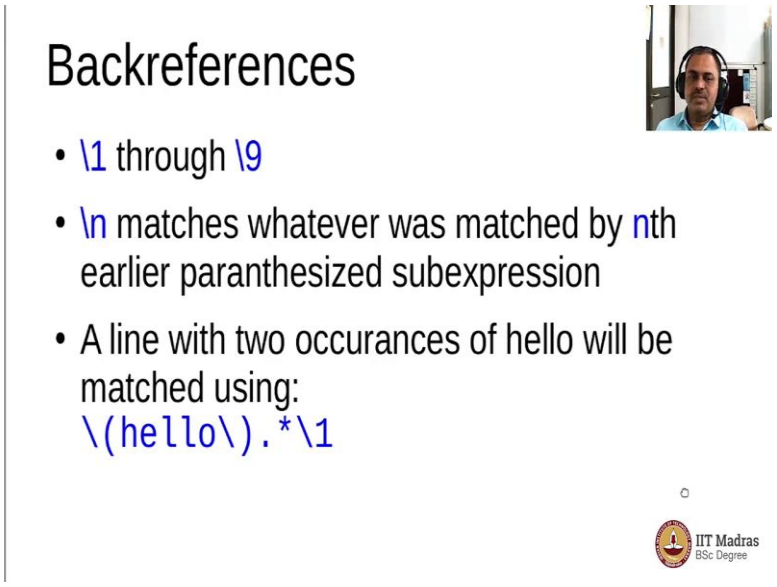
[[:print:]]	Printable	[[:blank:]]	Space / Tab
[[:alnum:]]	Alphanumeric	[[:space:]]	Whitespace
[[:alpha:]]	Alphabetic	[[:punct:]]	Punctuation
[[:lower:]]	Lower case	[[:xdigit:]]	Hexadecimal
[[:upper:]]	Upper case	[[:graph:]]	Non-space
[[:digit:]]	Decimal digits	[[:cntrl:]]	Control characters

IIT Madras
BSc Degree

To make life easy, there are certain character classes that are defined so that you could match those character classes without having to specify the range of those characters. So, any printable character can be matched by using colon print colon as the option for the character range. All the character classes have to be provided as options within the pattern using the double pair of square brackets. The first pair of square brackets is for the character enclosure and the second pair is to indicate the specific character class.

So, which means that you could also insert other characters that has to go along with the character classes by giving, let us say, an underscore or something else between the two closing or opening square brackets and there are many character classes that are defined with a wide variety that could be of use for you, the alphanumeric or alphabetic characters either lowercase or uppercase, decimal digits, character classes that match a space or a tab or a whitespace or a punctuation mark and so on. So, it is up to us to imagine how we can put these to good use so that the script that we write is as concise as possible.

(Refer Slide Time: 10:38)



Backreferences

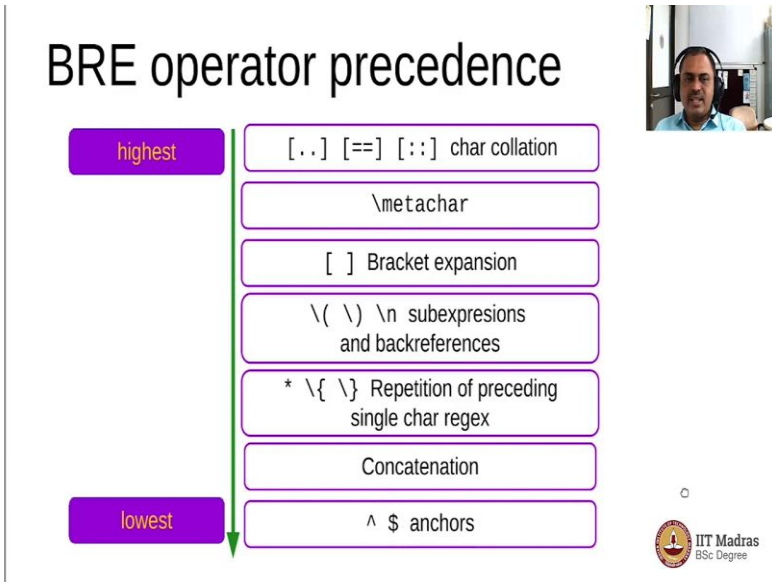
- \1 through \9
- \n matches whatever was matched by nth earlier paranthesized subexpression
- A line with two occurances of hello will be matched using:
  `\(hello\).*\1`

IIT Madras
BSc Degree

Now, when we have any grouping that is done using the parenthesis then such a match would also be possible to be referred and the back references are available in 9 different ways. So, there are 9 variables; slash backslash 1 all the way up to backslash 9. And for example here it is given hello is enclosed within the parenthesis for the BRE and you will see that the pattern would match hello occurring twice because whatever is matched within the parenthesis is then going to be matched again using backslash 1 and between them there is a dot star which indicates that it can have any number of characters between these two occurrences of hello.

So, in other words, this pattern would match a line that would contain two occurrences of hello in between them any arbitrary number of characters can come. So, this is a very useful feature also for replacement of a matched pattern which we will learn as an application of grep later on when we write the shell scripts, but for now it will be used to repeat whatever was matched within the same line.

(Refer Slide Time: 11:48)



BRE operator precedence

highest

[..] [==] [::] char collation

\metachar

[ ] Bracket expansion

\(\) \n subexpresions and backreferences

* \{ \} Repetition of preceding single char regex

Concatenation

^ $ anchors

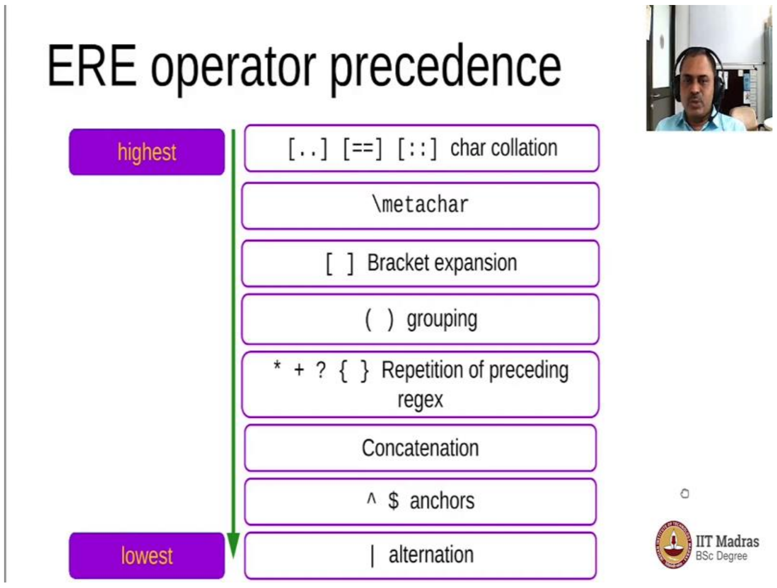
lowest

IIT Madras
BSc Degree

Now, there is an operator sequence just like any other arithmetic operation sequences in languages, programming languages. So, the highest preference is used for character collation, and then the meta characters which are escaped using the backslash and then comes the bracket expansion and after that the sub-expressions which are enclosed using the parenthesis and then the back references used for those patterns which are matched by this parenthesis.

And then comes the star character to indicate 0 or more matches of the preceding character or preceding sub-expression, and then the repetition of this particular sub-expression using the braces which are indicated with a backslash for BRE and without the backslash for ERE, and then comes a concatenation, and in the end, the lowest preference would be for the anchors which is for the beginning or the end of the line.

(Refer Slide Time: 12:44)

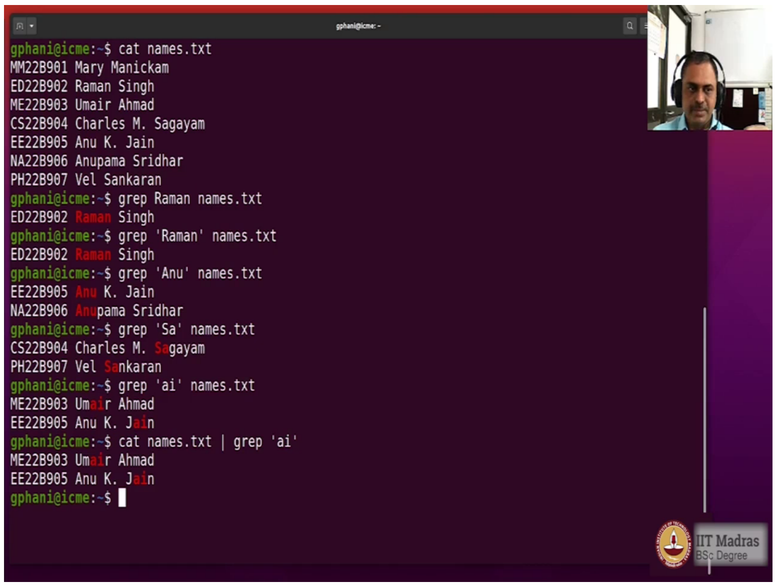


ERE operator precedence

highest	[..] [==] [::] char collation
	\metachar
	[ ] Bracket expansion
	( ) grouping
	* + ? { } Repetition of preceding regex
	Concatenation
	^ $ anchors
lowest	| alternation

The operator precedence is very similar for the extended regular expressions also, except for that you have the bracket expansion and grouping are done without the preceding backslash. And in addition, there is also an alternation operator pipe that is available which is having the lowest preference. So, now, let us go ahead and try these various options using the terminal and understand how the regular expressions help us in making use of the grep command to its fullest extent.

(Refer Slide Time: 13:19)



```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ grep Raman names.txt
ED22B902 Raman Singh
gphani@icme:~$ grep 'Raman' names.txt
ED22B902 Raman Singh
gphani@icme:~$ grep 'Anu' names.txt
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
gphani@icme:~$ grep 'Sa' names.txt
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
gphani@icme:~$ grep 'ai' names.txt
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
gphani@icme:~$ cat names.txt | grep 'ai'
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
gphani@icme:~$
```

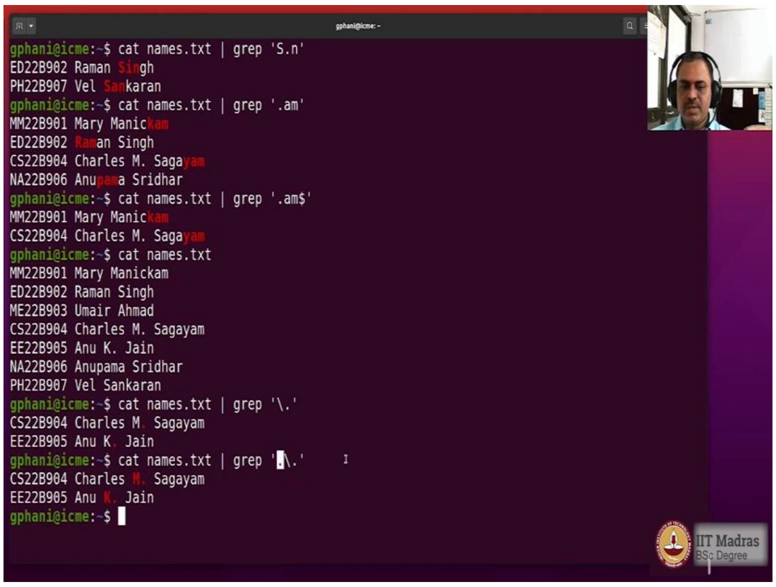
Let us get started with the grep command. So, to make our work simpler, I have created a text file in which there are some lines of text which you would like to filter. So, here is the name, list that I have created, it is the hypothetical list of roll numbers and the names of students and the pattern of the roll number is similar to what is followed in IIT Madras for the undergraduate and postgraduate students and the names are given with different formats with and without the initials in middle and let us see how we can pick various lines from this particular file by looking at patterns.

So, if you were to look for a particular string that could not be quite easy, you could do it either using the grep command. So, grep Raman names dot text would pick only the line that would have the string that is provided Raman which is matching, and you could also provide this particular string that has to be matched within single quotes which is actually a good habit to follow. Now, what has been matched would also be shown in a different color on the terminal if you have enabled the terminal colors.

You could now also look for any other pattern. Let us say you look for the pattern Anu and you would see that it would match two lines one where Anu is occurring as a single word and where Anu is occurring as a part of a word Anupama. So, let us look at this usage. You could also reduce the number of characters and you would see that if you were to look at Sa, capital S and small a, then it would match two lines where Sa is occurring at the beginning of those words because S is given as a capital letter, and you provide ai as the string and you would see that it would match to other lines where ai is occurring as part of those words.

Now, this kind of a string matching is the very first usage of grep and it is quite convenient. You could do it also using a cat command where the output of the cat command can be passed on to grep using a pipe, which we have seen in the past. Let us do that here. So, cat names dot txt pipe grep and then ai, and then after that, you do not have to provide any other option because the input is not in a file, but it is coming from the pipe already. So, you could see that that would work quite nicely. Now, this feature can also be used to process the output coming from commands so that you can pick up lines containing a particular string.

(Refer Slide Time: 16:07)



```
gphani@icme:~$ cat names.txt | grep 'S.n'
ED22B902 Raman Singh
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '.am'
MM22B901 Mary Manickam
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
NA22B906 Anupama Sridhar
gphani@icme:~$ cat names.txt | grep '.am$'
MM22B901 Mary Manickam
CS22B904 Charles M. Sagayam
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '\.'
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
gphani@icme:~$ cat names.txt | grep '.\.'
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
gphani@icme:~$
```

Now, let us look at the usage of dot in the grep command. So, we would have cat names dot text grep and within the single quotes I would give the pattern. So, I would reuse this particular string again and again. Let us say I gave a capital S dot n, which means that I would like to match a string which contains capital S after that any other character followed by n. And you will see that it match both Singh and Sankaran, because Sin and San would be matched, i and a are the ones which are taking the position of the dot. Now, this dot can also be in the beginning or at the end of the string.

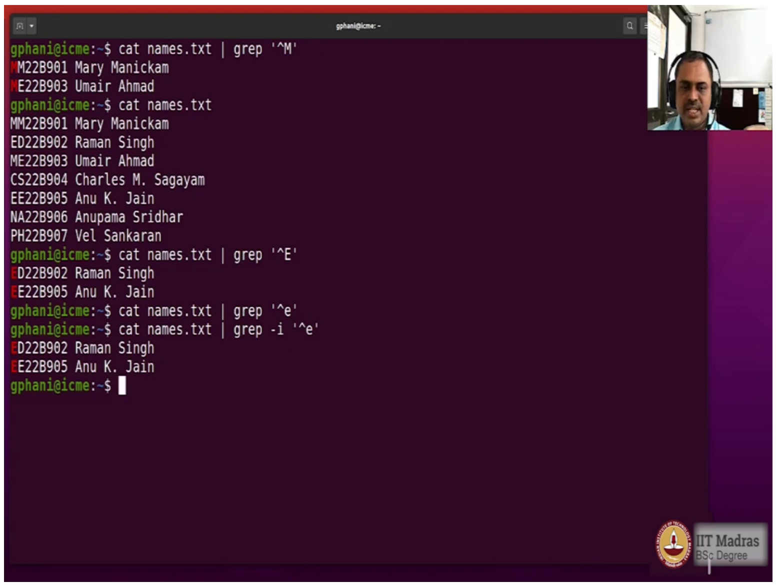
So, if you put dot am as the string, then you would see that the kam will be matched, ram will be matched yam and pam. So, dot am would match when the dot is replaced with any of the characters that is available. Now, some of these are to be at the end of a line. So, I would like to see whether there is a name that would end with am. So, for that, you have to indicate that it should be with an anchor dollar. And you would see that only those matching with the end of the line would be shown on the screen.

Now, if you look at the output you see that there is a dot here after the M and I would like to match that, so that means I want to pick up names of people who have an initial available in their names, and you cannot actually use the dot directly, you have to escape it. So, let us try that out, cat names grep and then backslash dot. So, it would now look for a dot in the name and then print. So, there are two names with dot and those will be printed for you.

And of course, if you want the dot to be necessarily after a character so that it is an abbreviation for their middle name or an initial then you can put one more dot, so the interpretation would be that the first dot is matching the initial and/or the abbreviation of the middle name, the second dot is the dot itself which has been escaped.

So, in the first matching, if you see, you have not provided the initial or the abbreviation, so the matching was happening only for the dot and not for the abbreviation. So, in the second case, the abbreviation is being matched with this dot that has been provided.

(Refer Slide Time: 18:56)

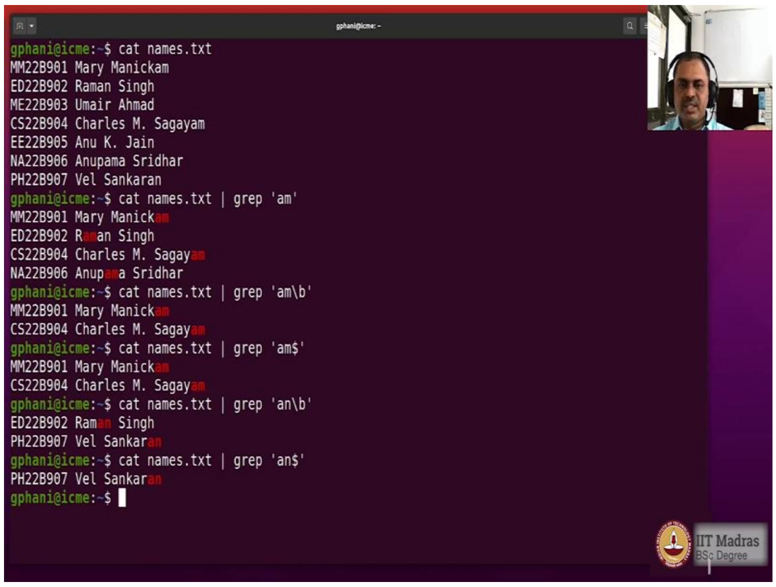


```
gphani@icme:~$ cat names.txt | grep '^M'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '^E'
ED22B902 Raman Singh
EE22B905 Anu K. Jain
gphani@icme:~$ cat names.txt | grep '^e'
gphani@icme:~$ cat names.txt | grep -i '^e'
ED22B902 Raman Singh
EE22B905 Anu K. Jain
gphani@icme:~$ 
```

Now, you could also match strings using the anchors at the beginning. So, let us say at the beginning I would like to have a capital M and you would see that among the various lines the ones which are corresponding to a roll number that looks like capital MM or capital ME, these two lines have been matched. So, you can see that this line and this line are matched because the beginning is with a capital M.

And you could also do that with the other characters and in this case all the beginning of the lines is with the capital character so that would be fine. And if you want to give a small character that would not work, but you can ask grep to do a case insensitive matching and then that would work by giving you the patterns that match with the e in the beginning without looking at whether it is capital or small.

(Refer Slide Time: 19:56)



```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep 'am'
MM22B901 Mary Manickam
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
NA22B906 Anupama Sridhar
gphani@icme:~$ cat names.txt | grep 'am\b'
MM22B901 Mary Manickam
CS22B904 Charles M. Sagayam
gphani@icme:~$ cat names.txt | grep 'am$'
MM22B901 Mary Manickam
CS22B904 Charles M. Sagayam
gphani@icme:~$ cat names.txt | grep 'an\b'
ED22B902 Raman Singh
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep 'an$'
PH22B907 Vel Sankaran
gphani@icme:~$
```

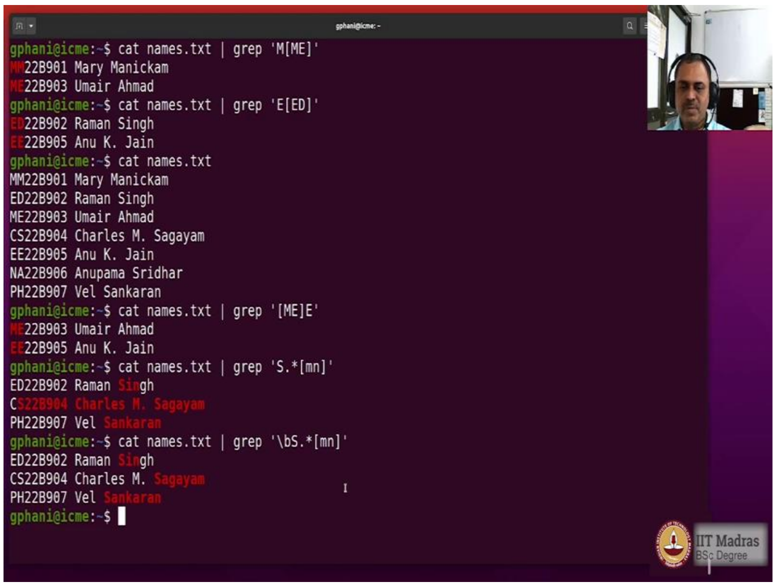
Now, there is a concept of word boundary that we would like to look at. So, you see that the end of the words are available as an, so, or am. So, let us look at the am character. So, you see that the am is matching both as a part of a name, like in Raman Singh, as Anupama, but it is also matching as a end of the word as in Manickam and Sagayam.

So, let us say I want to pick only those lines which are having the am at the end of the word. So, of course, in this case, it is also matching the end of the line, but you could also use the concept of a word boundary. So, slash b is for the word boundary. And you would see that it would match with the word boundary with the end of the line.

Now, it is also matching with the end of the line by a dollar anchor so both would give you the same output. But the difference would occur if you were to have an, then you will see that here, so if I look at an and then the word boundary, so Raman Singh is also getting matched, whereas Sankaran also is getting matched. Raman Singh is matched because it is occurring as a word boundary, Sankaran is occurring both as a word boundary as well as the end of the line.

And the difference will be illustrated if you put an anchor here with a dollar, I would match only Sankaran because Raman, the ending an is not at the end of the line, but it has only a word boundary. So, the choice of the word boundary with a slash b or a dollar as the end of the line is up to you to pick depending upon the situation.

(Refer Slide Time: 21:42)



```
gphani@icme:~$ cat names.txt | grep 'M[ME]'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
gphani@icme:~$ cat names.txt | grep 'E[ED]'
ED22B902 Raman Singh
EE22B905 Anu K. Jain
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '[ME]E'
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
gphani@icme:~$ cat names.txt | grep 'S.*[mn]'
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '\bS.*[mn]'
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
gphani@icme:~$
```

Now, let us look at the use of square brackets. So, the square brackets are essentially to give options. So, let us say I want to give the options of two characters M followed by either M or E and you will see that the roll numbers MM and ME will be shown. So, if you see the first M is matched with the first M here, the second character is matched to be either an M or an E. So, you could also now look at it as E followed by either E or a D and you would see that ED and EE both are matched. So, the second character there is basically an option that is being made available.

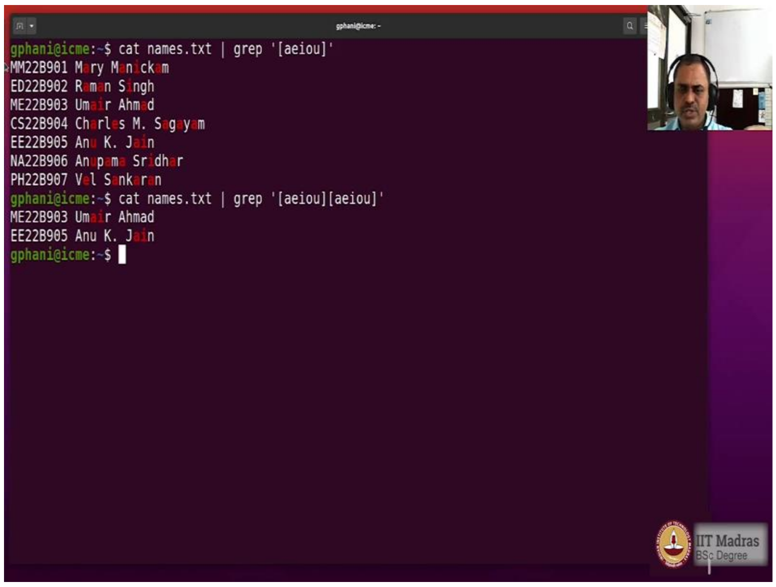
So, now, let us see whether that can be used in a nice manner. So, you could actually see that in the first character, I can choose either M or an E and the second character can be E. So, it would match either ME or EE, and you can see that the roll numbers are given in the same way. So, the matching pattern is shown in red color text, which is easy for you to identify. Now, this can be used also with the other parts of the string. So, let us try that out.

So, I want to match anything that starts with an S and ends with either an m or an n. So, you see that Singh and then Sagayam and Sankaran all of them are matched. Now, you see that the second line is matching with an S that is coming as a part of the roll number, but let us say our intention was to match it only in the name, but not in the roll number and we know that the roll number and the name have been separated by a space, and therefore, there is a word boundary that is involved there.

So, we could go ahead and try to introduce a word boundary before S so that the interpretation is that after a word boundary there should be a word that starts with S and then any number of characters followed by either an m or an n.

And you see that now the matching is happening with the second line not with the roll number but with the Sagayam that is actually the word in the end. So, the amount of matching that is done is only for the last word but not for the one starting with the roll number. Now, why is this useful, because whatever is matched can be placed in a back reference variable which we have already seen in the lectures and that is something that we could actually be making use of.

(Refer Slide Time: 24:22)

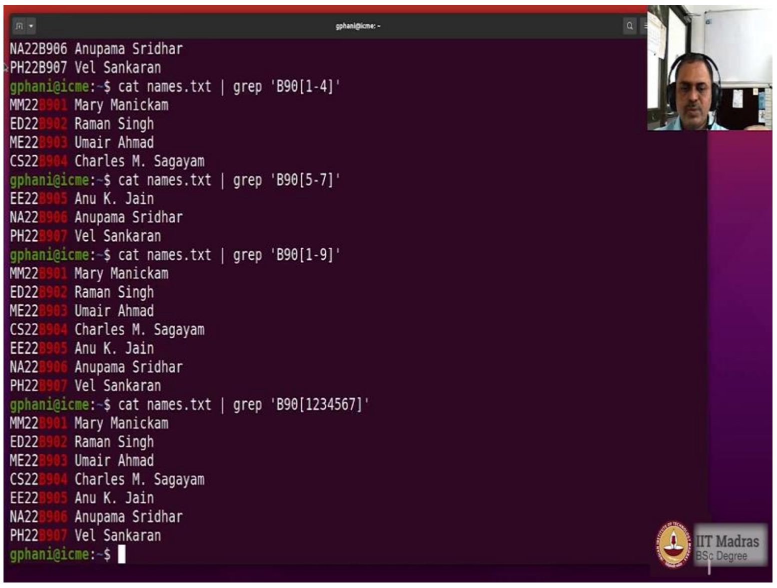


```
gphani@icme:~$ cat names.txt | grep '[aeiou]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '[aeiou][aeiou]'
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
gphani@icme:~$
```

Now, the square bracket can be used to also be giving us an options of characters that can be matched. So, let us match let us say vowels. So, aeiou is a vowel and let us see what are all the strings that match vowel containing them and you would see that all names would usually have a vowel so that would be matched. But let us say we would like to match those names that contain two vowels side by side.

So for that I would give the pattern like this. So, it is going to be matching two characters because there are two pairs of square brackets side by side. The first pair contains option of characters that is listed which is vowels and the second also contains the same set of vowels. And you see that it could match two names where they are two vowels side by side ai as in Umair and Jain.

(Refer Slide Time: 25:30)

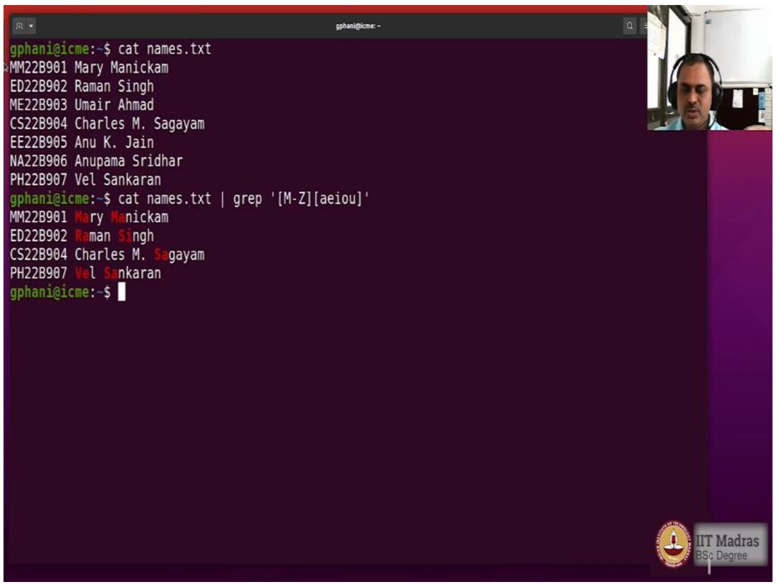


```
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep 'B90[1-4]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
gphani@icme:~$ cat names.txt | grep 'B90[5-7]'
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep 'B90[1-9]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep 'B90[1234567]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$
```

Hyphen can be used within the square brackets to indicate a range of characters. So, let us also try that out. So, look at the roll numbers, they contain a sequence. So, I would have 9 0 and then from 1 to 7 the roll numbers are starting. So, I would like to give a range there. I would like to have only 1 to 4 roll numbers to be considered. So, now you see that the first four rows are being shown. So, which means that the pattern that is matched is B90 and then the fourth character can be either 1 or 2 or 3 or 4. So, I am not listing all of them, I am just giving a range.

Now, I could modify the range to let us say 5 to 7, and you see that the remaining ones are being shown. And I could also give it as 1 to 9, and then all the names are shown, of course, 8 and 9 are not there, but that does not matter, because if any pattern does not match, then that is anyway being ignored. Now, this is definitely much more simpler than let us say providing 1, 2, 3, 4, 5, 6, 7 which would be also the same output. So, naturally, if there is any sequence of characters that could also be reduced using a hyphen, then we should go ahead and use that feature.

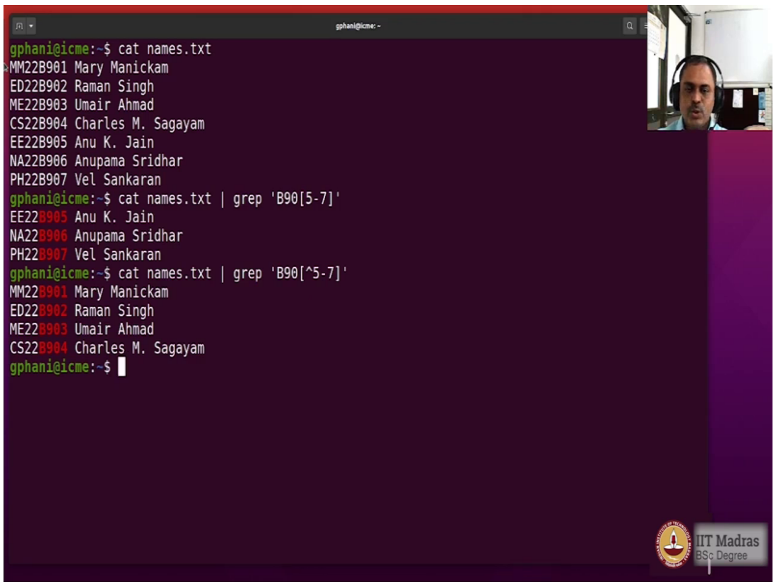
(Refer Slide Time: 26:49)



```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '[M-Z][aeiou]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
gphani@icme:~$
```

Now, let us try this out using the names also. So, I would go by looking at those names which would have the first character in the second half of the alphabet and then followed by vowel. And you would see that Mary Manickam, Raman etc. are coming, but Anu and Anupama are not coming, because their first character comes in the first half of the alphabet. If you have a hat inside the square brackets, it is implying that there is a negation of the pattern. So, let us go ahead and try that out.

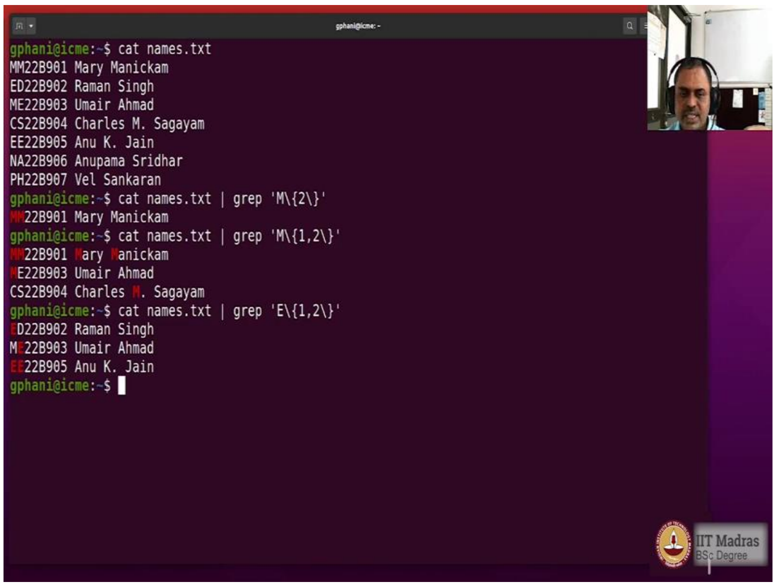
(Refer Slide Time: 27:30)



```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep 'B90[5-7]'
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep 'B90[^5-7]'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
gphani@icme:~$
```

So, here, if you see, 5 to 7, and I am going to negate that, so that means that the character that follows B90 should not be either 5 or 6 or 7. So, you see that it is then giving the remaining of the lines. So, the hat inside the square brackets is to negate the range of characters that is provided within these square brackets.

(Refer Slide Time: 28:00)



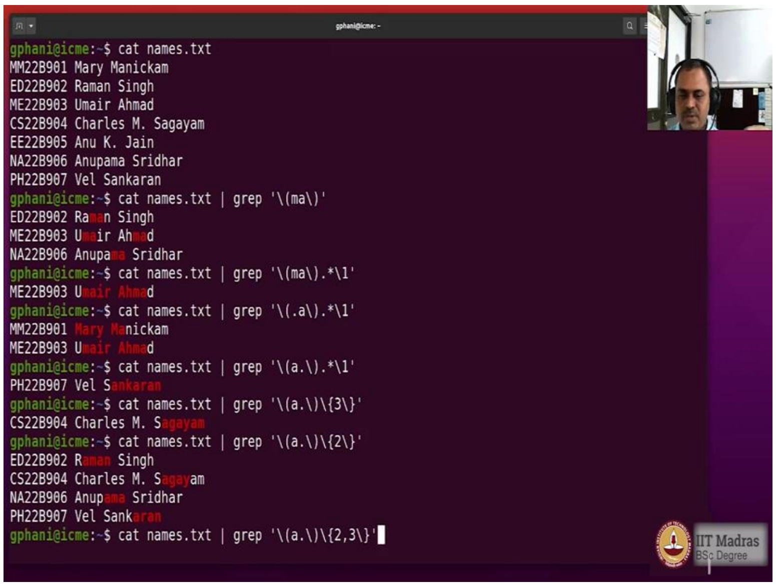
```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep 'M\{2\}'
MM22B901 Mary Manickam
gphani@icme:~$ cat names.txt | grep 'M\{1,2\}'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
gphani@icme:~$ cat names.txt | grep 'E\{1,2\}'
ED22B902 Raman Singh
ME22B903 Umair Ahmad
EE22B905 Anu K. Jain
gphani@icme:~$ 
```

Now, let us use in the BRE system where using the grep command we could provide the number of times a particular character can occur. So, let us say the M should occur twice. So, exactly twice should it should occur. And you see that MM is being matched. Now, if you say that it should occur either once or twice, then you see that ME is matching M dot is also matching. So, you could see that you could specify exactly how many times it has to be matched side by side by using the braces and in BRE the braces have to be preceded by the backslash.

So, you can see that the braces are preceded by a backslash. And then the number that is between the braces is the number of times the preceding character has to be matched. And here when you say only one number, it means exactly that many types of matching is to be required. And when you give a range of numbers like 1 comma 2, that means it should be matched at least once but not more than 2 times including 2 times.

So, let us try another feature let us say E and then at least once or at most two times and you will see that it would match ED as well as ME because all of them would actually have the matching either once or twice. Now, you could also group the patterns that are being matched. So, let us look at that now.

(Refer Slide Time: 29:39)



```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '\(ma\)'
ED22B902 Raman Singh
ME22B903 Umair Ahmad
NA22B906 Anupama Sridhar
gphani@icme:~$ cat names.txt | grep '\(ma\).*\1'
ME22B903 Umair Ahmad
gphani@icme:~$ cat names.txt | grep '\(.a\).*\1'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
gphani@icme:~$ cat names.txt | grep '\(a.\).*\1'
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '\(a.\)\{3\}'
CS22B904 Charles M. Sagayam
gphani@icme:~$ cat names.txt | grep '\(a.\)\{2\}'
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | grep '\(a.\)\{2,3\}'
```

So, the grouping is done by using parenthesis which are also to be preceded by a backslash in the case of BRE. So, let us do like that. So, ma is a string and we are just grouping it. And when you just do it, it is no different from just matching it ma, but our idea was to see if I can match it once more. So, I would like to match it once more with any other character between them. So that means that I would like to have two occurrences of ma within the same line. And you will see that what is the line that I am going to pick up, I am going to pick up Umair Ahmad there.

So, you could see that it comes up. So, you can see that ma is matched with this ma. And then ir blank ah is being matched with the dot star and then again one more ma is coming because we have matched that with the first group which is ma. So, backslash 1 means first group which has been given in parenthesis and backslash 1 refers to the back reference of the matched pattern that has been provided before. So, this is a very neat way by seeing that what has to be matched.

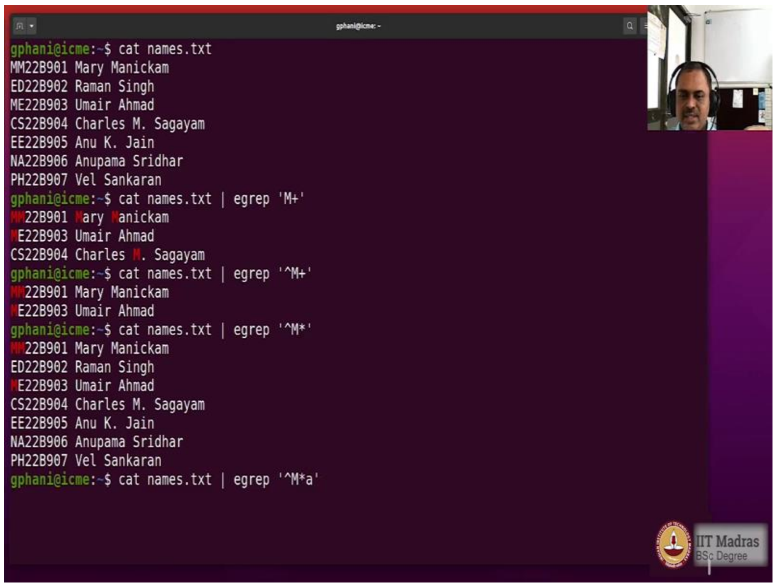
So, now, it also means that this ma need not be provided as it is. We could also say a character followed by a should be repeated once more. So, whatever is character that is following a should be repeated and you see that here capital M is also getting matched because in Mary Manickam capital M is coming before a, and in the case of Umair Ahmad small m is coming before a. So, we are able to catch both of them and this dot is matching either the capital M or small m, but whatever is matched that is being repeated a second time by use of backslash 1.

Now, you could also do that with a dot to see what happens if you are going to match an occurrence that is following a coming twice. So, Sankaran is having one name where n is followed by a and you can see that it is coming second time again towards the end of the word, so an, an. So, the first an is getting matched, the dot is the n here and after the first an, kar is getting matched by the dot star here and after that an is getting matched again by back reference here. So, the n that is matched in the dot is getting reflected also in the backslash 1.

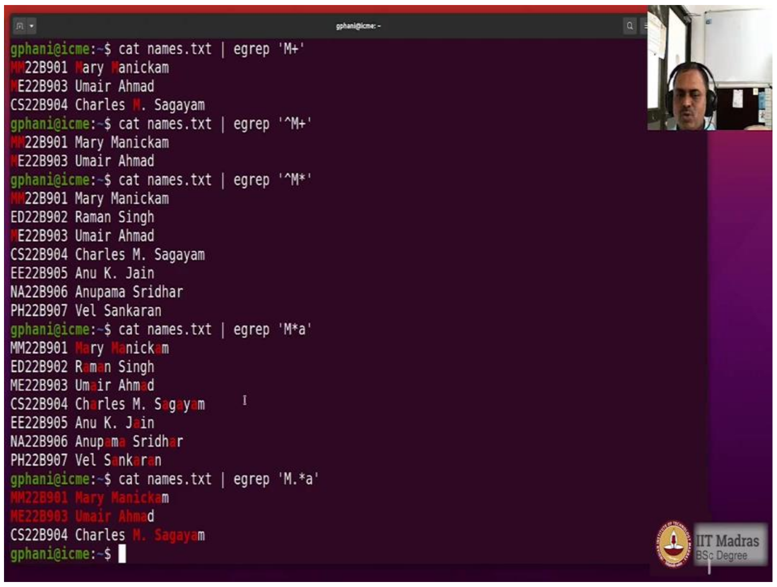
Now, let us say if I want that to be appearing three times that also can be looked up. So, you could also use this pattern to see how many times it has to be matched. So, here I am using the braces to tell how many times I want that particular pattern to be matched. So, in the past, we have used it for a character, but here we are using for the pattern. So, you see that the Sagayam is coming because agayam, so the pattern called a and dot is coming three times and it is coming because here we have put a and then dot as a group and that group called a and character after that is to be repeated three times and that is coming here.

Now, if it is to be repeated two times, then you see Raman will come because am, an, then Anupama will come because am a blank and then Sankaran also will come because it is ar and an. So, you could see that the parenthesis can be used to group character or a sub-expression and then after that, you can either back reference it for one more matching or you could also provide the braces to tell how many times you would like to have it matched. Now, if you want to get it matched either two times or three times that also is possible for you to say and you will see that two that will be listed with a union of these two outputs that we have seen.

(Refer Slide Time: 33:52)



```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | egrep 'M+'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
gphani@icme:~$ cat names.txt | egrep '^M+'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
gphani@icme:~$ cat names.txt | egrep '^M*'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | egrep '^M*a'
```



```
gphani@icme:~$ cat names.txt | egrep 'M+'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
gphani@icme:~$ cat names.txt | egrep '^M+'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
gphani@icme:~$ cat names.txt | egrep '^M*'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | egrep 'M*a'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | egrep 'M.*a'
MM22B901 Mary Manickam
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
gphani@icme:~$
```

The extended regular expression engine can be used while running the command grep either by using an option capital E or by running the command as E-grep. So, let us look at that possibility. Now, the patterns that you can use in extended regular expression engine are slightly more, so plus is one such character.

So, by plus we mean it should be matched either once or more number of times. So, I would like to say M followed by a plus that means M can occur either once or more number of times. So, you would see that it would match in lines where M is occurring either once or twice.

Now, I can also put a hat in front of it to say that this kind of matching should occur only in the beginning of the line. And you would see that both MM and ME will occur because in

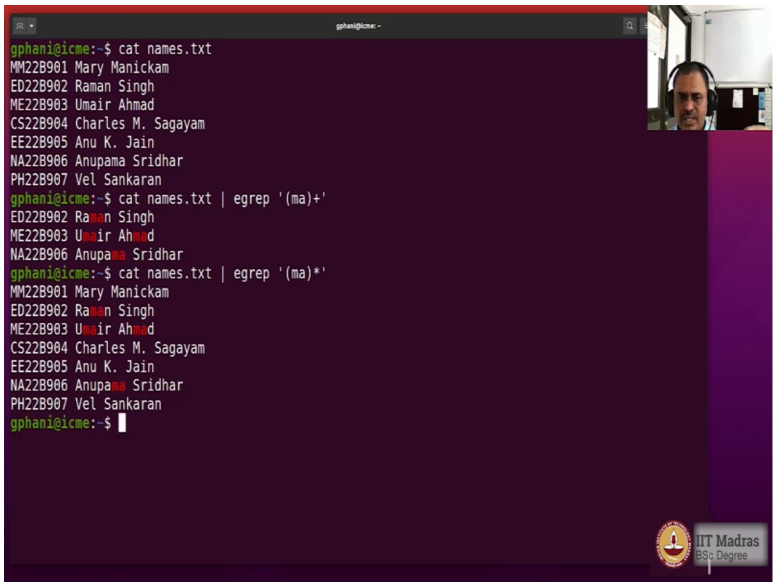
ME case it is matching once and in MM case it is matching twice. The plus will indicate that it should match either once or more number of times, not 0. Star would actually occur for 0. So, you would see that when you put a star there, so it would occur 0, which means that other lines which do not have M are also getting matched.

Now, what is the purpose of those lines? Well, when you put something else after the star, then it would actually make a sense. So, you would see that after capital M, I would, let us say, put an a there without the hat and you would see here. So, you would see that M can be matched with Ma or it could also be just na itself, because anything that is in front of a can also be matched either M or not. So, there are situations where one has to watch when star is used. So, star does not mean any number of characters or any number of characters it should be dot.

So, when you put a dot there, now the dot is actually being operated upon by star to indicate any number of characters and now you see that definitely M has to occur following that the any number of characters can come and after that an a has to come. So, you can see that here the M is starting, after that any number of characters have been matched and then there is an a. So, here also M is matching, after that any number of characters and then a. And here also M is there after that any number of characters and then a.

So, M dot star a is more close to what we would interpret based upon our experience with the shell, where star indicates any number of matches, but this is actually something that you have to watch. The star is operating on M which means that either M can be present or not and therefore you could see that even when M is not present before a, those strings are also getting matched. So, watch out for the interpretation of star.

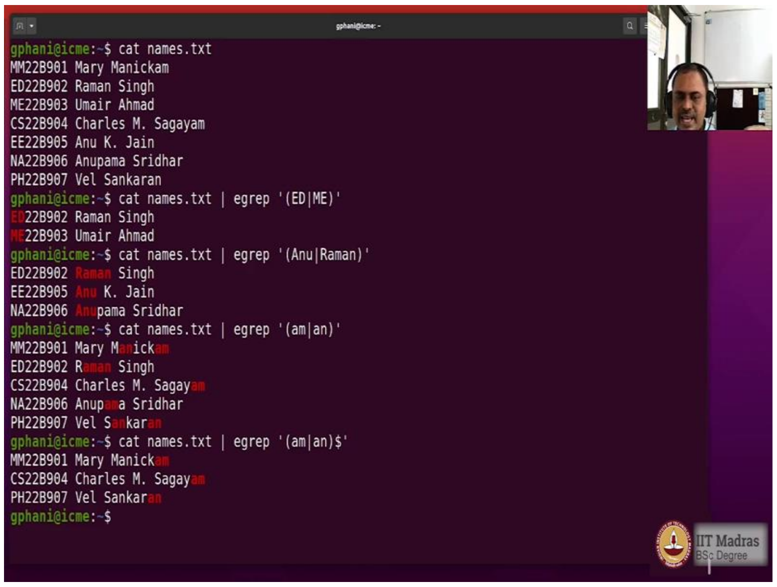
(Refer Slide Time: 36:50)



```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | egrep '(ma)+'
ED22B902 Raman Singh
ME22B903 Umair Ahmad
NA22B906 Anupama Sridhar
gphani@icme:~$ cat names.txt | egrep '(ma)*'
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$
```

Now, let us use the other features of ERE. So, apart from plus, you could also have other features. You can also group the characters and then use a plus. So, let us say I would have a group called ma and then after that I would put a plus, which means that the pair of characters ma can occur either once or twice. So, you could see that Raman Singh, Umair Ahmad will come because they are all occurring once in those particular patterns. And the difference with respect to star also will be evident because it would match even those situations where the pattern ma is not occurring.

(Refer Slide Time: 37:38)



```
gphani@icme:~$ cat names.txt
MM22B901 Mary Manickam
ED22B902 Raman Singh
ME22B903 Umair Ahmad
CS22B904 Charles M. Sagayam
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | egrep '(ED|ME)'
ED22B902 Raman Singh
ME22B903 Umair Ahmad
gphani@icme:~$ cat names.txt | egrep '(Anu|Raman)'
ED22B902 Raman Singh
EE22B905 Anu K. Jain
NA22B906 Anupama Sridhar
gphani@icme:~$ cat names.txt | egrep '(am|an)'
MM22B901 Mary Manickam
ED22B902 Raman Singh
CS22B904 Charles M. Sagayam
NA22B906 Anupama Sridhar
PH22B907 Vel Sankaran
gphani@icme:~$ cat names.txt | egrep '(am|an)$'
MM22B901 Mary Manickam
CS22B904 Charles M. Sagayam
PH22B907 Vel Sankaran
gphani@icme:~$
```

Now, we would also explore the use of a pipe as an alternation between two patterns of strings that will be matched. So, I would like to match, let us say, in the beginning either ED should be there or ME should be there. The mechanical or engineering design both are very close by. So, I would like to match these two sets of students. So, you would see that those two are being matched. And you could also have names that can be matched with the different lengths also. So, I may want to have, let us say, Anu should be matched or Raman should be matched and that would also work quite fine.

So, the length of the strings that are coming on either sides of the pipe need not be same. And let us put am and an and you will see that the pattern is matching. Those who have the two characters in their names, either am or an. Now, you could combine this with other things also. For example, you can combine with the anchor of dollar and you see that those who have either an am or an in the end are getting matched. The dollar is indicating that it should be done at the end of the line.