

System Commands
Professor Gandham Phanikumar
Metallurgical and Materials Engineering
Indian Institute of Technology Madras
Pattern Matching - Part 02

(Refer Slide Time: 00:14)

```
gphani@icme:~$ dpkg-query(1) dpkg suite dpkg-que
NAME
dpkg-query - a tool to query the dpkg database

SYNOPSIS
dpkg-query [option...] command

DESCRIPTION
dpkg-query is a tool to show information about packages listed in the dpkg database.

COMMANDS
-l, --list [package-name-pattern...]
List all known packages matching one or more patterns, regardless of their
status, which includes any real or virtual package referenced in any dependency
relationship field (such as Breaks, Enhances, etc.). If no package-name-
pattern is given, list all packages in /var/lib/dpkg/status, excluding the ones
marked as not-installed (i.e. those which have been previously purged).
Normal shell wildcard characters are allowed in package-name-pattern. Please
note you will probably have to quote package-name-pattern to prevent the shell
from performing filename expansion. For example this will list all package
names starting with "libc6":

dpkg-query -l 'libc6*'

The first three columns of the output show the desired action, the pac
status, and errors, in that order.
Manual page dpkg-query(1) line 1 (press h for help or q to quit)
```

```
gphani@icme:~$ man dpkg-query
gphani@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n'
```

```
gphanig@icme:~$ x11 xserver-xorg-video-amdgpu
x11 xserver-xorg-video-ati
x11 xserver-xorg-video-fbdev
x11 xserver-xorg-video-intel
x11 xserver-xorg-video-nouveau
x11 xserver-xorg-video-qxl
x11 xserver-xorg-video-radeon
x11 xserver-xorg-video-vesa
x11 xserver-xorg-video-vmware
x11 xterm
x11 xtrans-dev
web xul-ext-ubufox
x11 xwayland
editors xxd
utils xz-utils
misc yaru-theme-gnome-shell
misc yaru-theme-gtk
misc yaru-theme-icon
misc yaru-theme-sound
gnome yelp
gnome yelp-xsl
gnome zenity
gnome zenity-common
utils zip
libs zlib1g:amd64
libdevel zlib1g-dev:amd64
default zoom
gphanig@icme:~$
```

```
gphanig@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep '^(.){4}$'
shells bash
admin bolt
utils cpio
net crda
admin cron
net cups
web curl
shells dash
devel dbus
admin dpkg
utils file
devel flex
utils fuse
```

```
gphanig@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep '^(.){4}$'
x11 xserver-xorg-legacy
x11 xserver-xorg-video-all
x11 xserver-xorg-video-amdgpu
x11 xserver-xorg-video-ati
x11 xserver-xorg-video-fbdev
x11 xserver-xorg-video-intel
x11 xserver-xorg-video-nouveau
x11 xserver-xorg-video-qxl
x11 xserver-xorg-video-radeon
x11 xserver-xorg-video-vesa
x11 xserver-xorg-video-vmware
x11 xterm
x11 xtrans-dev
web xul-ext-ubufox
x11 xwayland
utils xz-utils
misc yaru-theme-gnome-shell
misc yaru-theme-gtk
misc yaru-theme-icon
misc yaru-theme-sound
gnome yelp
gnome yelp-xsl
gnome zenity
gnome zenity-common
libs zlib1g:amd64
libdevel zlib1g-dev:amd64
default zoom
gphanig@icme:~$
```

```
gphani@icme:~$ dpkg-query -f='${Section} ${binary:Package}\n' | egrep '^.{4}'
x11 xserver-xorg-video-amdgpu
x11 xserver-xorg-video-ati
x11 xserver-xorg-video-fbdev
x11 xserver-xorg-video-intel
x11 xserver-xorg-video-nouveau
x11 xserver-xorg-video-qxl
x11 xserver-xorg-video-radeon
x11 xserver-xorg-video-vesa
x11 xserver-xorg-video-vmware
x11 xterm
x11 xtrans-dev
web xul-ext-ubuntox
x11 xwayland
editors xxd
utils xz-utils
misc yaru-theme-gnome-shell
misc yaru-theme-gtk
misc yaru-theme-icon
misc yaru-theme-sound
gnome yelp
gnome yelp-xsl
gnome zenity
gnome zenity-common
utils zip
libs zlib1g:amd64
libdevel zlib1g-dev:amd64
default zoom
gphani@icme:~$
```

```
gphani@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep 'g.{3}'
web lynx
devel make
utils awk
utils melt
editors nano
net nmap
gnome orca
perl perl
ruby rake
ruby ruby
admin sudo
utils time
tex tipa
admin udev
gnome vino
web wget
x11 xorg
gnome yelp
default zoom
gphani@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep 'g.{3}'
interpreters gawk
gnome gdm3
net gftp
graphics gimp
utils gpgv
utils grep
utils gzip
gphani@icme:~$
```

```
gphani@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep 'g.{1,5}'
graphics gimp
utils gpgv
utils grep
utils gzip
gphani@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep 'g.{1,5}'
devel g++
devel g++-9
interpreters gawk
devel gcc
devel gcc-9
gnome gcr
devel gdb
admin gdisk
gnome gdm3
gnome gedit
net gftp
graphics gimp
vcs git
interpreters gjs
utils gnupg
devel gperf
utils gpg
utils gpgsm
utils gpgv
utils grep
x11 guake
utils gzip
gphani@icme:~$
```

So, in the previous session we have learned how to list the packages. So, we have used a command called `dpkg query`, which allowed us to combine the various fields to output them onto the screen and we have seen a list of packages that is coming onto the screen. And we have used the `grep` to pick up and we can actually now modify that to a little bit further extent.

So, I am just going to run the same query, so minus `W` to provide information about the packages and minus `f` to provide the pattern. So, I have the section I would like to have and then one space and then the name of the package has to be shown, and after that I would like to have the newline character and then now you see that this list is quite long. And here is where we would like to use the `grep` features.

So, let us say I use the extended regular expression engine, and I would like to look at all those package names which have exactly four characters. So, `W get` is a package name with four characters, `n map` is a package name with four characters. So, I want to pick up those kinds of package names. So, how do I provide the pattern? So, I would actually have space because the package name is coming after the space.

So, I do not want to match those which are in the section, only in the package I would like to match. And then I do not know what character has to be matched, so I put a dot. So, any character can be there, but that should occur exactly four times. And after the package name has been given, I should go to the end of the line. So, the dollar will tell me that it is going to be end of the line.

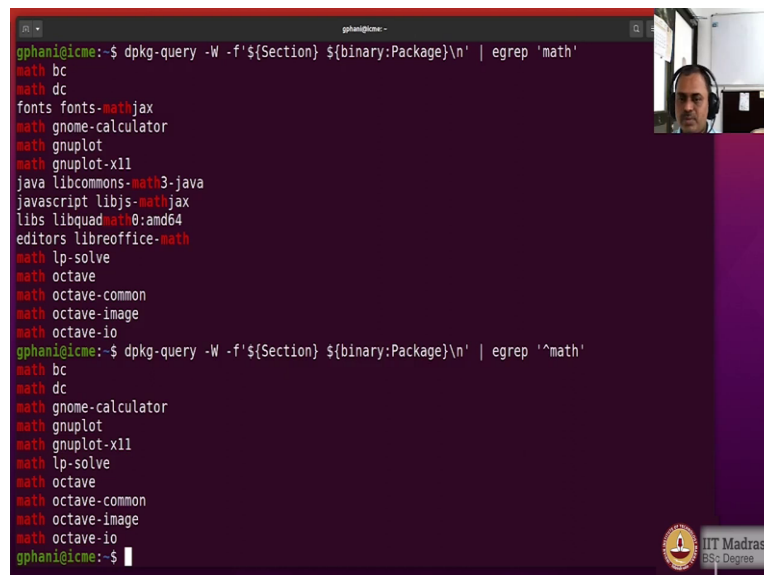
So, now look at that and you see that we are getting package names which are exactly four characters long. And what happens if I do not have the dollar there. You see that it would match four characters, but the remaining characters are also present which are not being looked at. So, by putting the dollar here we are insisting that the package name should have been finished in just four characters.

And what is the blank here for, so I would actually also show you what happens if you do not have the blank here. You would see that the matching is occurring also in the name of the section so that I do not want. So, I put a blank here so that it is matching from the second field onwards and then it is matching only in the name of the package and the dollar is actually matching only the end of the line to be occurring after four characters of the name of the package.

Now, let us say I would like to have the name of the package to be starting with g and the three characters after that, so g something, something, something. Now, you see that this is giving us the package names which starts with the g and are four later words, gmp or grep or gzip or gftp etc. And we could also look at, after g I can have either 1 or up to let us say 5 characters.

And you see that slightly a longer list of g starting packages are there, g plus plus onwards all the way up to guake terminal which we have looked at in one of the earlier classes. So, you could see that the egrep can be used in a very powerful manner to process the matching of the string precisely telling what character you would like to match and how many times etc.

(Refer Slide Time: 04:14)

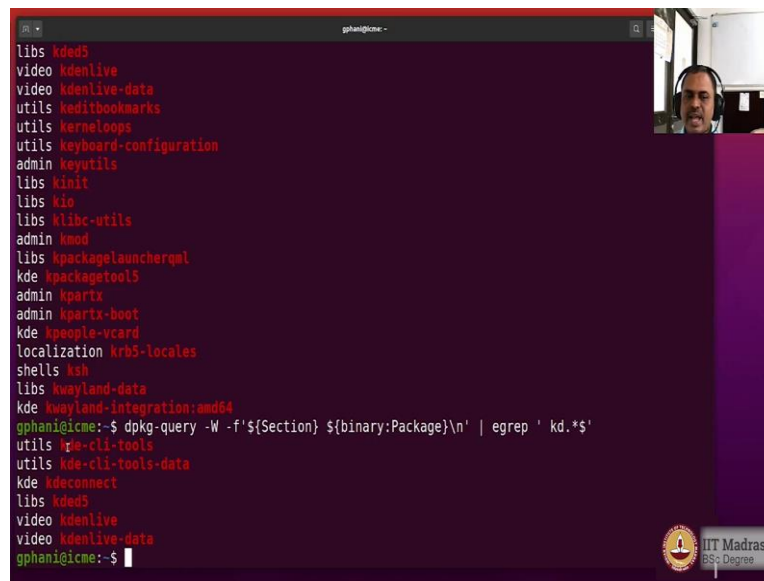


```
gphani@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep 'math'
math bc
math dc
math fonts-mathjax
math gnome-calculator
math gnuplot
math gnuplot-x11
math java-libcommons-math3-java
math javascript-libjs-mathjax
math libs-libquad-math0-amd64
math editors-libreoffice-math
math lp-solve
math octave
math octave-common
math octave-image
math octave-io
gphani@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep '^math'
math bc
math dc
math gnome-calculator
math gnuplot
math gnuplot-x11
math lp-solve
math octave
math octave-common
math octave-image
math octave-io
gphani@icme:~$
```

Now, let us say you would like to have all those packages which are coming under the category called mathematics. So, that you want to go through those packages and be thorough of it. So, let us look at that. Math should be matched within the name of the package. Now, you see that it is matching also names of packages here, but not only sections. Now, I want to match it only for the section. So which means that I want the math character set to should be only in the beginning of the line. So already we know the anchor for that. So I put the hat in the front.

And now you see that only those packages which come under the section called math are being displayed, but not the names of packages which have math in middle but actually belonging to some other section. So, you see the utility of the egrep along with the features such as the hat as well as the number of times the particular character should come so that you can actually precisely pick up the lines of output that you are interested in.

(Refer Slide Time: 05:26)



```
gphani@icme:~$ dpkg-query -W -f='${Section} ${binary:Package}\n' | egrep 'kd.*$'
libs kded5
video kdenlive
video kdenlive-data
utils keditbookmarks
utils kerneloops
utils keyboard-configuration
admin keyutils
libs kinit
libs kio
libs klibc-utils
admin kmod
libs kpackagekit
kde kpackagekit-tools
admin kpartx
admin kpartx-boot
kde kpeople-vcard
localization krb5-locales
shells ksh
libs kwayland-data
kde kwayland-integration:amd64
gphani@icme:~$
```

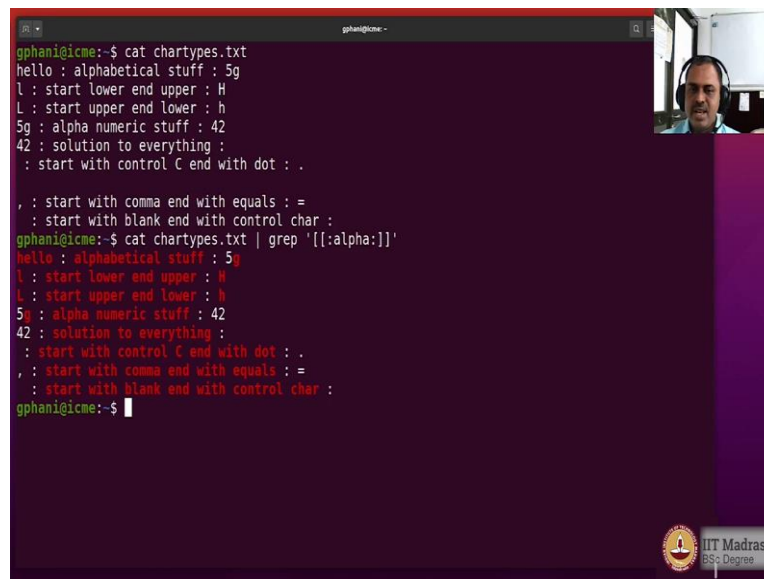
The image shows a terminal window with a dark background and red text. It displays the output of a command that filters packages starting with 'kd.*'. The output lists various KDE-related packages and their sections. A small video feed of a person is visible in the top right corner of the terminal window. The IIT Madras logo is in the bottom right corner.

Now, there are some packages which are coming from the kde graphical user interface of Ubuntu and the default GUI for Ubuntu is gnome. So, you can look at those also here. So, k and after that I have something that starts with k and then a dollar. So, packages that starts with k. And there are many of them. So, I think that k is not sufficient, kd. So, now this is much better, kde is common desktop environment and there are some packages that are coming.

So, kdenlive is like an editor used to trim the videos etc., which is what I do for our course. And there are also other tools. So, these are coming from a common desktop environment, which requires some other libraries also to be present compared to the gnome user interface. So, you think about what way you want to pick up and analyze the pattern that would capture them accurately. And if you have used the thermal colors enabled then you can also find out what has been matched and that would also help you understand the pattern matching better.

Now, there are some features that are available in the grep which help you with respect to the type of character sets that can be matched, so alphabetic, alphanumeric, digits etc. So, we will look at them using a file in which I have put intentionally some punctuation marks or control characters. How do we insert non-typable characters in a text file is something that we will learn when we come to the text editors shortly, but for now I will just display the text file to you and then we will see how we can pick specific lines from it depending upon the type of character set that we would like to match.

(Refer Slide Time: 07:00)

A terminal window with a dark purple background. The prompt is 'gphani@icme:'. The first command is 'cat chartypes.txt', which outputs several lines of text defining character sets. The second command is 'cat chartypes.txt | grep "[[:alpha:]]"', which filters the output to show only lines containing alphabetical characters. In the top right corner, there is a small video feed of a man wearing a headset. In the bottom right corner, there is a logo for 'IIT Madras BSc Degree'.


```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep "[[:alpha:]]"
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$
```

So, here is a file that I have in place. So, character types, I have got lines in which we have got various strings. So, I have got a beginning of the line with something and ending also with something intentionally so that those strings will be matched appropriately. So, I have got a lower letter, lowercase letter, I have got uppercase letter, I have got a digit, and then I also have a non-printable character like Ctrl C. And here also there is a tab character that has been used and so on. And there is a blank space here in front of it. So, and then there is an empty line also in the middle.

So, let us see how we can pick these lines by using the character sets which come with a double square bracket and colons separating the name of the character set that we are using. So, in all of them I will use the same pattern where we will use a cat chartypes pipe grep. Now, let us use this pattern. Here we would like to let us say pick alphabetic characters. So, any line that would have an alphabetical character would be matched, and you see that all the lines will be matched, except the blank line, because the blank line has nothing in it, so that has been skipped.

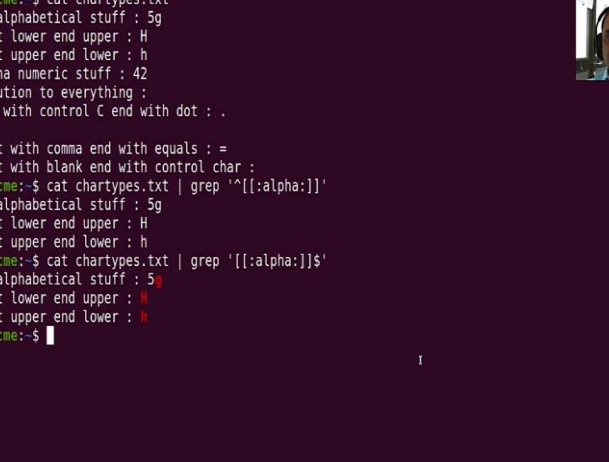
(Refer Slide Time: 08:28)



```
gphanig@icme -  
gphanig@icme:~$ cat chartypes.txt  
hello : alphabetical stuff : 5g  
l : start lower end upper : H  
L : start upper end lower : h  
5g : alpha numeric stuff : 42  
42 : solution to everything :  
 : start with control C end with dot : .  
  
 : start with comma end with equals : =  
 : start with blank end with control char :  
gphanig@icme:~$ cat chartypes.txt | grep '[[:alnum:]]'  
hello : alphabetical stuff : 5g  
l : start lower end upper : H  
L : start upper end lower : h  
5g : alpha numeric stuff : 42  
42 : solution to everything :  
 : start with control C end with dot : .  
 : start with comma end with equals : =  
 : start with blank end with control char :  
gphanig@icme:~$
```

Next, the alphanum, that is lines which would have alphanumeric characters will be matched. And you see that it is the same output as the previous one. And it has also skipped the empty line because it has no alphanumeric character that can be matched on that line.

(Refer Slide Time: 08:50)



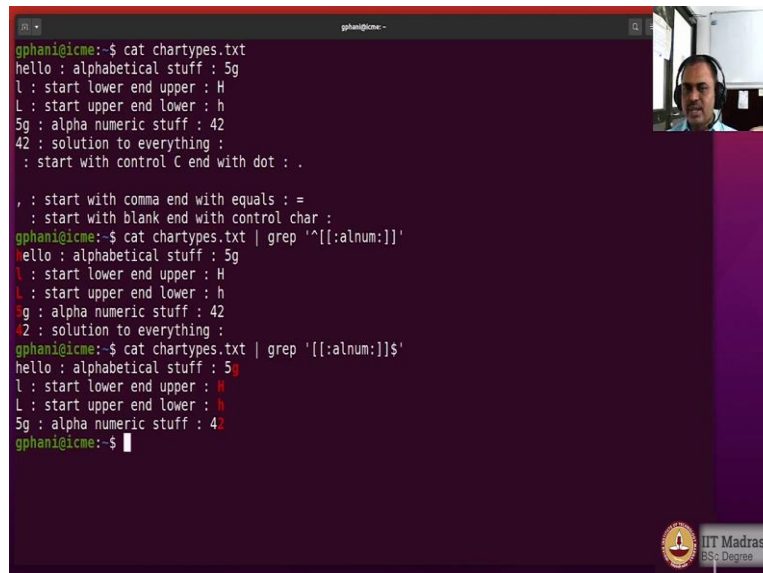
```
gphan@icme: ~  
gphan@icme:~$ cat chartypes.txt  
hello : alphabetical stuff : 5g  
l : start lower end upper : H  
L : start upper end lower : h  
5g : alpha numeric stuff : 42  
42 : solution to everything :  
 : start with control C end with dot : .  
  
 : start with comma end with equals : =  
 : start with blank end with control char :  
gphan@icme:~$ cat chartypes.txt | grep '^[[[:alpha:]]$'  
hello : alphabetical stuff : 5g  
l : start lower end upper : H  
L : start upper end lower : h  
gphan@icme:~$ cat chartypes.txt | grep '[[[:alpha:]]$'  
hello : alphabetical stuff : 5  
l : start lower end upper : H  
L : start upper end lower : h  
gphan@icme:~$
```

Now, let us say I would like to have the alphabetical character in the beginning of the string. Now, you see that only some of the lines are showing up. So, if you see hello and then l and capital L these are getting matched because they have an alphabetical character at the beginning, but the remaining lines do not have. It has 5 and it has a 4 and this fellow has a control c and there is a empty character here null and this is a comma here and there is a blank. So, none of them are actually alphabetical characters, so they are all skipped.

Now, you could also ask for the alphabetical character to be at the end of the line. And you will see that only some of the lines are being shown, the one with a g, capital H and a small h. The remaining lines are not shown because here it is a 2 which is a digit, and here it is a tab character and here it is a dot and here there is a null and here there is an equal to sign, and here again, there is a control character.

So the all of those have been skipped. And we have got only the first three lines which have an alphabetical character in the end. So, you now see that alphabetical character can be matched either at the beginning or anywhere in the line or at the end and you need to be watchful which ones are suitable for your purpose.

(Refer Slide Time: 10:11)

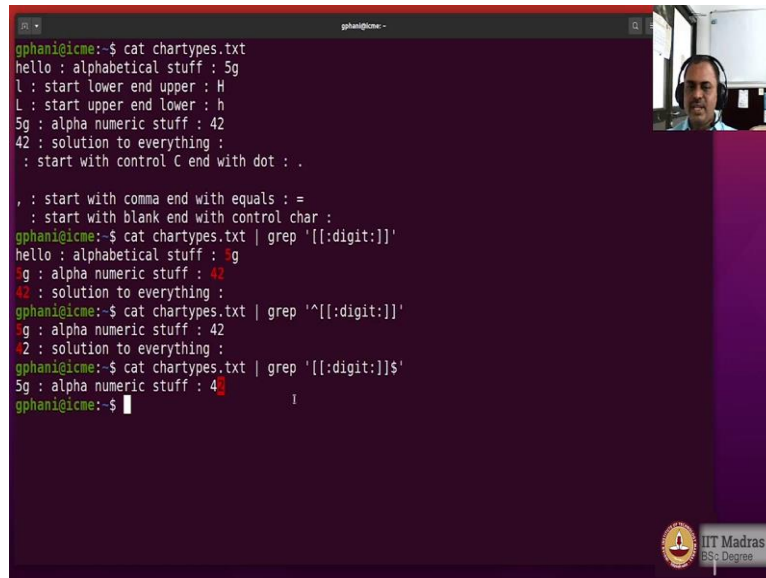


```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '^[:alnum:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
gphani@icme:~$ cat chartypes.txt | grep '[:alnum:]$'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
gphani@icme:~$
```

Now, we can also look at the alphanumeric character at the beginning of the line. And you see that it captures a little bit more number of lines. So, the first character can be either an alphabetical or a numeric one, but not a space or a null or a punctuation mark or a control character. So, those lines have been skipped, the remaining ones have been shown. Now, you could also ask for an alphanumeric character at the end of a line. And you see that again those have g or H or small h or 2 are being shown, but the once which control character or a punctuation or a null or a equal to sign, etc., they are all being skipped.

(Refer Slide Time: 10:50)



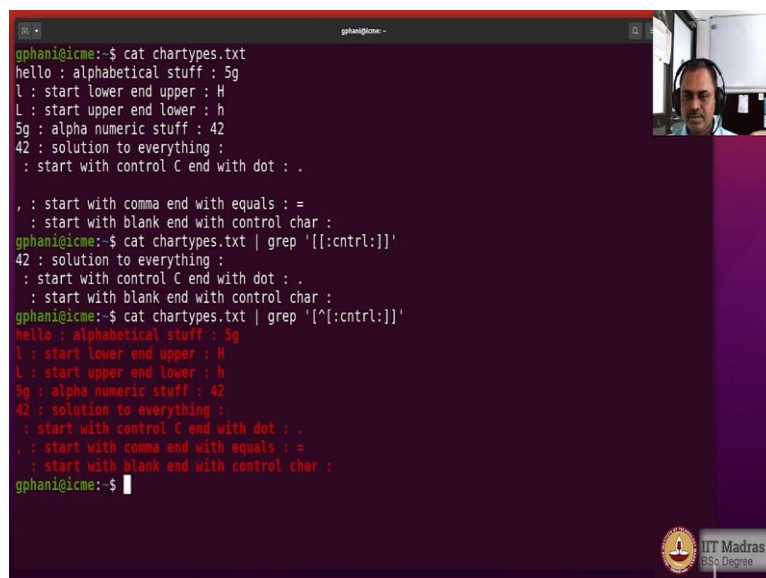
```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:digit:]'
hello : alphabetical stuff : 5g
5g : alpha numeric stuff : 42
42 : solution to everything :
gphani@icme:~$ cat chartypes.txt | grep '^[:digit:]'
5g : alpha numeric stuff : 42
42 : solution to everything :
gphani@icme:~$ cat chartypes.txt | grep '[:digit:]$'
5g : alpha numeric stuff : 42
gphani@icme:~$
```

The terminal window shows a series of commands and their outputs. The first command is `cat chartypes.txt`, which displays the contents of the file. The second command is `cat chartypes.txt | grep '[:digit:]'`, which shows lines containing any digit. The third command is `cat chartypes.txt | grep '^[:digit:]'`, which shows lines where a digit is at the beginning. The fourth command is `cat chartypes.txt | grep '[:digit:]$'`, which shows lines where a digit is at the end. The IIT Madras logo is visible in the bottom right corner.

You can now look for lines that contain digits anywhere in the line. And you could also ask the digit should be at the beginning of the line or it should be at the end of the line. So, you could see that when you do not provide any anchor, it would match anywhere in the line. So, here the 5 is somewhere in the middle of the line, here it is in the beginning of the line. And in the case you provide an anchor with the hat, it means it has to be matched only in the beginning of the line. So, the 5g and 42 are being matched. And when you say end of the line as an anchor, then only the 42 line will be shown as matched line from the file chartypes dot txt.

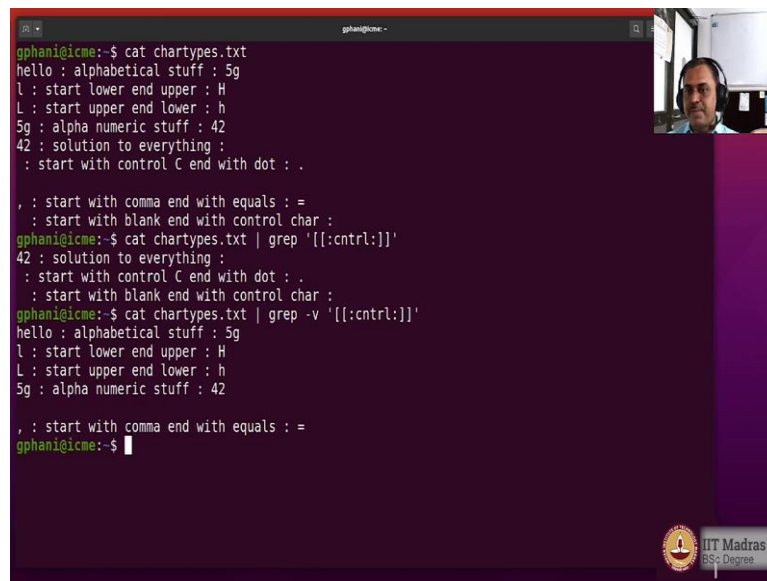
(Refer Slide Time: 11:32)



```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:cntrl:]'
42 : solution to everything :
: start with control C end with dot : .
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '^[[:cntrl:]]'
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$
```

The terminal window shows a series of commands and their outputs. The first command is `cat chartypes.txt`, which displays the contents of the file. The second command is `cat chartypes.txt | grep '[:cntrl:]'`, which shows lines containing any control character. The third command is `cat chartypes.txt | grep '^[[:cntrl:]]'`, which shows lines where a control character is at the beginning. The IIT Madras logo is visible in the bottom right corner.

A terminal window with a dark purple background. The prompt is 'gphani@icme:~'. The user has run 'cat chartypes.txt' and the output is displayed. Then, the user runs 'cat chartypes.txt | grep '[:cntrl:]'' and the output shows three lines containing control characters. In the top right corner, there is a small video feed of a man with a beard and glasses. In the bottom right corner, there is a logo for 'IIT Madras BSc Degree'.

```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

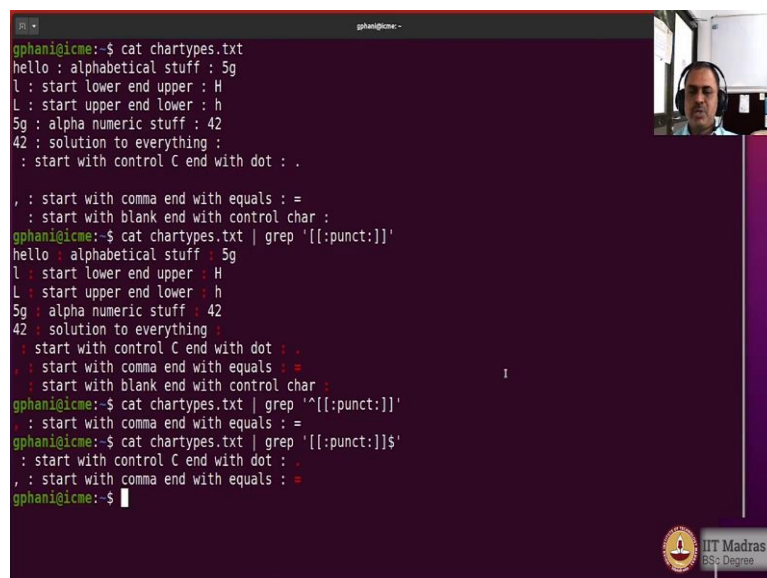
, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:cntrl:]'
42 : solution to everything :
: start with control C end with dot : .
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep -v '[:cntrl:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42

, : start with comma end with equals : =
gphani@icme:~$
```

Now, there are some control characters that I have placed in this file. And we could see whether you could pick the lines that contain the control characters. And you see that these are the three lines. So here I have got a tab character and here I have got a control c sitting here and here also I have got a control character sitting in the front. So, those three lines only have been picked up. Now, you could ask a reverse question saying that I do not want any control character.

Now, you could also ask that you would like to have only the lines which do not contain the control character and the minus v option would actually give you the opposite effect where the lines that are not matching are shown. And you see that the remaining lines include the empty line are being shown.

(Refer Slide Time: 12:25)

A terminal window with a dark purple background. The prompt is 'gphani@icme:~'. The user has run 'cat chartypes.txt' and the output is displayed. Then, the user runs 'cat chartypes.txt | grep '[:punct:]'' and the output shows all lines except the three control character lines. Then, the user runs 'cat chartypes.txt | grep '^[:punct:]\$'' and the output shows only the empty line. In the top right corner, there is a small video feed of a man with a beard and glasses. In the bottom right corner, there is a logo for 'IIT Madras BSc Degree'.

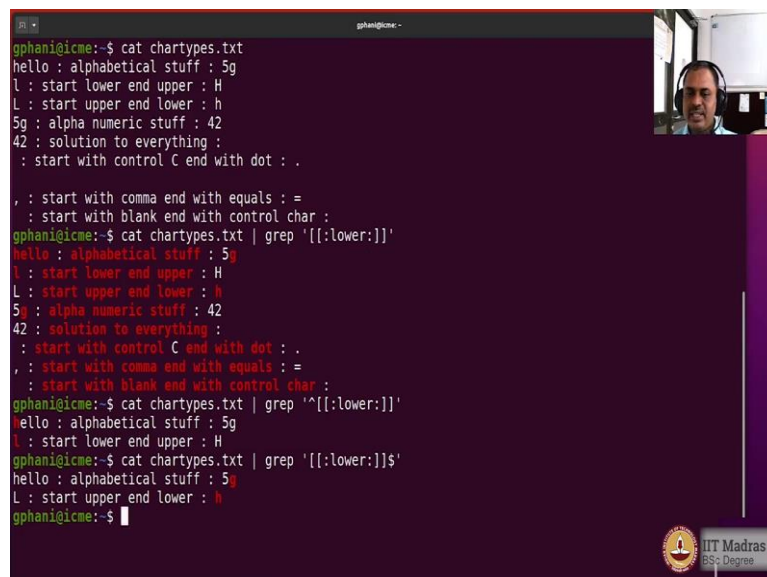
```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:punct:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '^[:punct:]$'
: start with control C end with dot : .
, : start with comma end with equals : =
gphani@icme:~$
```

Now, you could also look for any punctuation characters to be matched. So, let us look at that punct would match for all punctuation characters. So, all those lines which have punctuation characters, colon is one of the punctuation characters, so that has been matched nicely. And almost every line has that except the empty line, so empty line has been skipped. So, now I give an anchor at the beginning of the line.

So, only those lines at the beginning contain a punctuation mark will be shown. So, we have got a comma there in the first, this line here so that has been matched. And you could also ask for a line that ends with a punctuation and you see that the ones with have a dot and the equal to sign have been matched. So, the punctuation marks without specifying what those marks are can also be used as a character set.

(Refer Slide Time: 13:15)

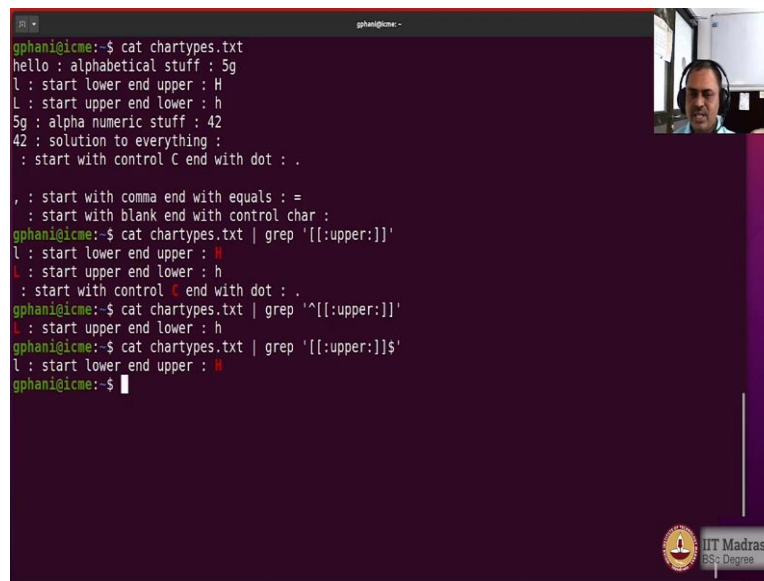


```
gphanigme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphanigme:~$ cat chartypes.txt | grep '[:lower:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :
gphanigme:~$ cat chartypes.txt | grep '^[:lower:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
gphanigme:~$ cat chartypes.txt | grep '[:lower:]]$'
hello : alphabetical stuff : 5g
L : start upper end lower : h
gphanigme:~$
```

Now, the case of the character also can be used. So, if I want to have matching lines that contain only, that contain lower case characters, so you would see that all the lines except the empty line are being shown because they all contain some lowercase letters. And I would put an anchor here to show that the lowercase characters should be matched at the beginning of the line. And so the hello and l will be getting matched there with a capital, with a small h and l and I could give an anchor to indicate that I want the matching to be done at the end and the 5g line and the h in the end will be matched because they contain a lowercase character at the end of the string.

(Refer Slide Time: 14:04)



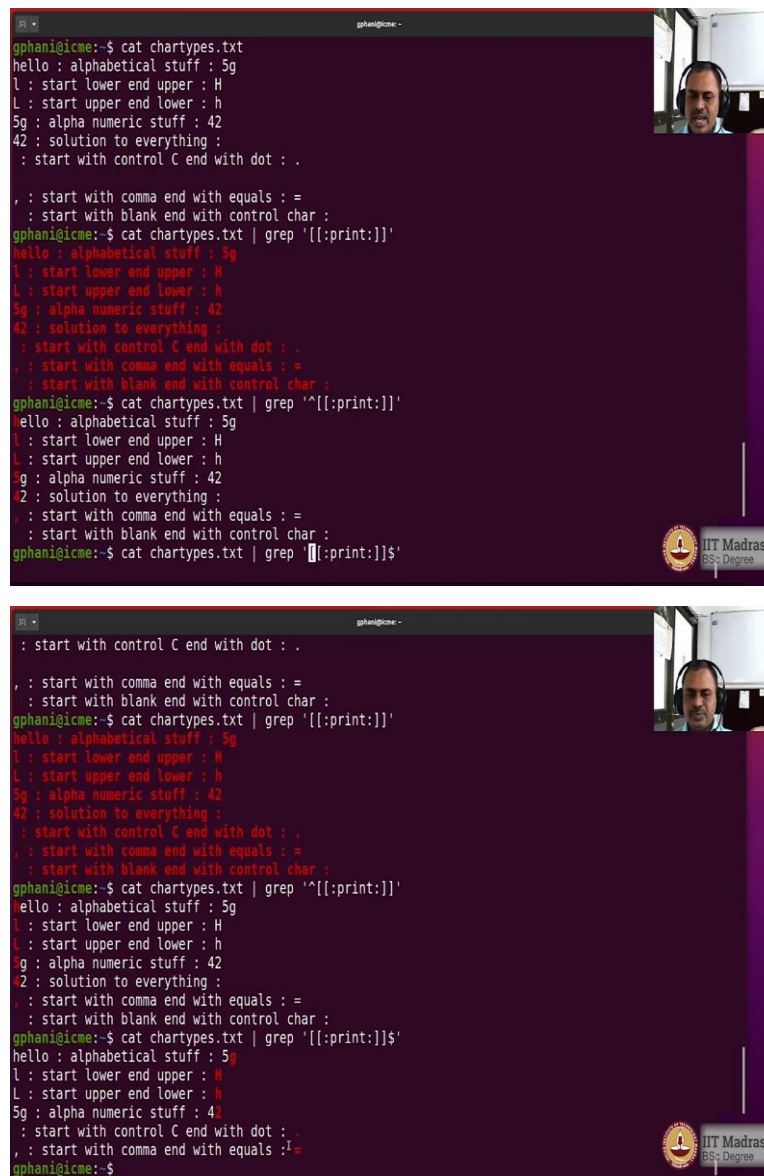
```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:upper:]'
l : start lower end upper : H
L : start upper end lower : h
: start with control C end with dot : .
gphani@icme:~$ cat chartypes.txt | grep '^[:upper:]'
L : start upper end lower : h
gphani@icme:~$ cat chartypes.txt | grep '[:upper:]$'
l : start lower end upper : H
gphani@icme:~$
```

You could also look at the same kind of analysis using the uppercase characters. So, all those lines which contain any uppercase character will be displayed and you see that we do not have uppercase characters used very much so only three lines are used. So, you have got a capital H here, capital L here and capital C here. So, only three lines have been matched. And I will put an anchor here.

So, only the ones which have a starting with an uppercase character will be shown. So, capital L is getting matched there. And I put a dollar here and remove the hat. So, I get a matching where the ending of the line will be a capital letter that would be the capital H. So, now you see that you could also specify the matching pattern using a character set that contains the uppercase characters.

(Refer Slide Time: 14:56)



```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :

gphani@icme:~$ cat chartypes.txt | grep "[[:print:]]"
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :

gphani@icme:~$ cat chartypes.txt | grep "^[[:print:]]"
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :

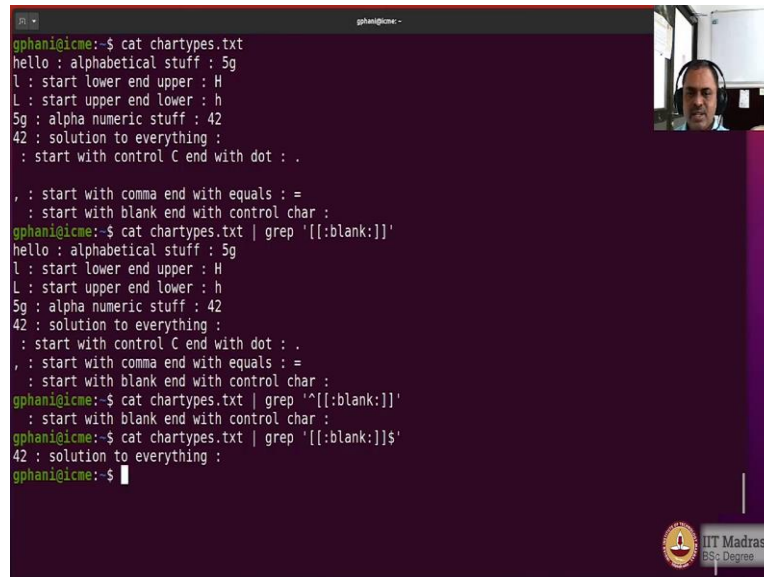
gphani@icme:~$ cat chartypes.txt | grep "[[:print:]]$"
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
gphani@icme:~$
```

The alphanumeric characters as well as the punctuation marks are combined together as printable characters, because they can display on the screen and the control characters are then avoided. So, you can actually use the class, character class called print to indicate that you would like to match only printable characters. And when you want to display you would see that every line except the blank line has been matched, because except the blank line every other line has some printable characters or other.

Now, you could also ask for a printable character to be at the beginning of the line. And you would see that those lines which have a control character in the beginning have been skipped. Then you could also ask for the printable character to be the end of the line. And you see that some of the lines which contain a control character at the end are skipped.

So, here, for example, here there is a control character at the end that has been skipped. And here there is a control character in the beginning and that has been kept because the printable character is matched only in the end and therefore in the beginning can be anything else.

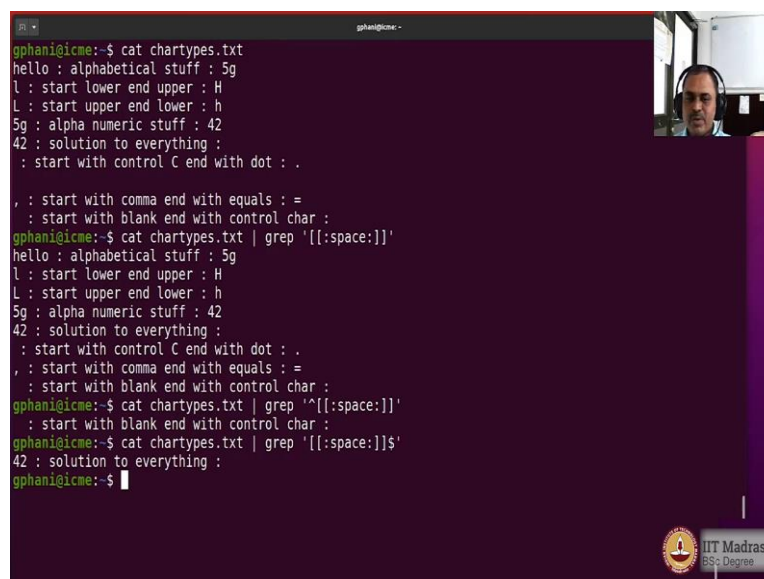
(Refer Slide Time: 16:06)



A terminal window with a dark purple background. The prompt is 'gphani@icme:~'. The user has entered 'cat chartypes.txt' and the output is: 'hello : alphabetical stuff : 5g', 'l : start lower end upper : H', 'L : start upper end lower : h', '5g : alpha numeric stuff : 42', '42 : solution to everything :', and ': start with control C end with dot : .'. The user then enters 'cat chartypes.txt | grep '[:blank:]'' and the output is the same as before. The user then enters 'cat chartypes.txt | grep '^[:blank:]'' and the output is the same as before. The user then enters 'cat chartypes.txt | grep '[:blank:]\$' and the output is the same as before. The terminal window has a small video feed in the top right corner showing a man with a headset. In the bottom right corner, there is a logo for 'IIT Madras BSc Degree'.

```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:blank:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '^[:blank:]'
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:blank:]$'
42 : solution to everything :
gphani@icme:~$
```



A terminal window with a dark purple background. The prompt is 'gphani@icme:~'. The user has entered 'cat chartypes.txt' and the output is: 'hello : alphabetical stuff : 5g', 'l : start lower end upper : H', 'L : start upper end lower : h', '5g : alpha numeric stuff : 42', '42 : solution to everything :', and ': start with control C end with dot : .'. The user then enters 'cat chartypes.txt | grep '[:space:]'' and the output is the same as before. The user then enters 'cat chartypes.txt | grep '^[:space:]'' and the output is the same as before. The user then enters 'cat chartypes.txt | grep '[:space:]\$' and the output is the same as before. The terminal window has a small video feed in the top right corner showing a man with a headset. In the bottom right corner, there is a logo for 'IIT Madras BSc Degree'.

```
gphani@icme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:space:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '^[:space:]'
: start with blank end with control char :
gphani@icme:~$ cat chartypes.txt | grep '[:space:]$'
42 : solution to everything :
gphani@icme:~$
```

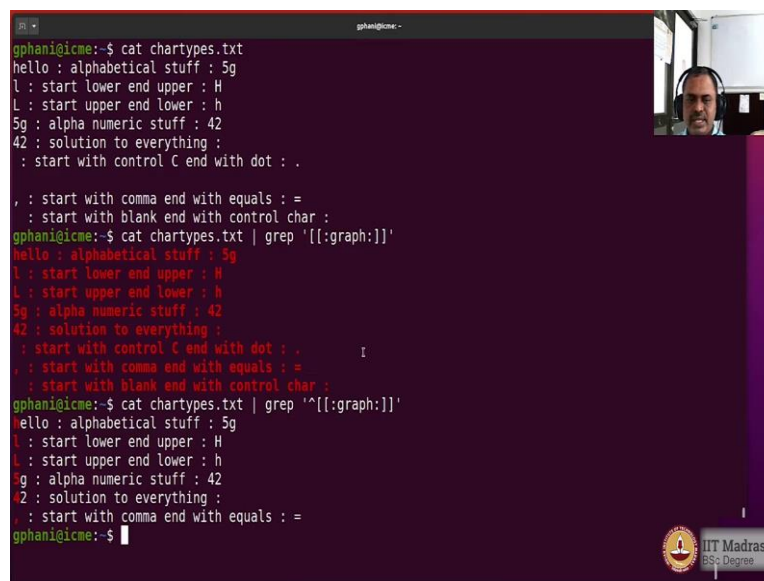
Now, there are some characters which give you a blank space like a space character or a tab character, so they can also be matched. So, these are the whitespace characters. And you would see that except the empty line, all the lines contain some whitespace or other and that has been matched.

And you could also ask for the anchor to show that in the beginning or to show that in the end. So, you see that there is a blank space that is available at the beginning or at the end of

these two lines so they are being shown, but the remaining lines have the blank space somewhere in the middle so those are being picked up.

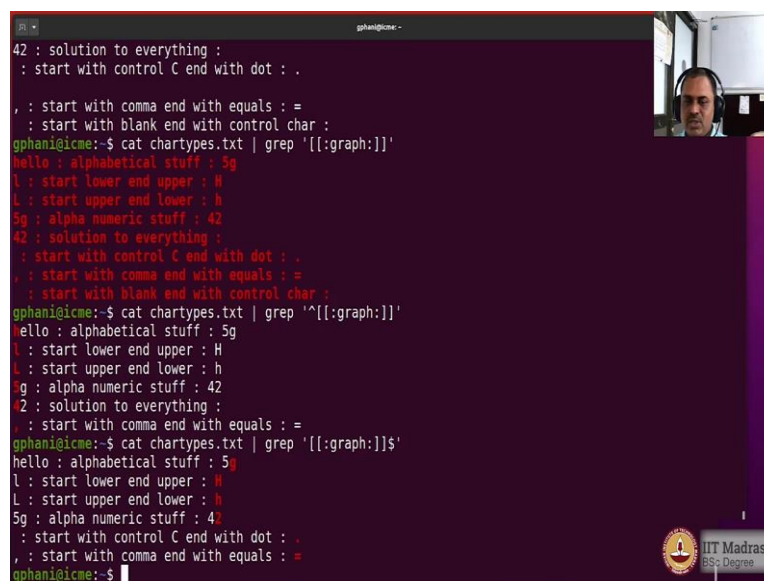
Now, there is a character class called space that can also be used to pick up lines that contain some whitespace. And you could also ask for it to have the beginning with the whitespace or the end with this whitespace. Now, this is not very different from the blank space except that there are more whitespace characters than what we have used. And you can see what type of a whitespace that is allowed for your input file and accordingly filter them out using the difference between these two character sets.

(Refer Slide Time: 17:30)



```
gphanigme:~$ cat chartypes.txt
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .

, : start with comma end with equals : =
: start with blank end with control char :
gphanigme:~$ cat chartypes.txt | grep '[:graph:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :
gphanigme:~$ cat chartypes.txt | grep '^[:graph:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with comma end with equals : =
gphanigme:~$
```



```
42 : solution to everything :
: start with control C end with dot : .

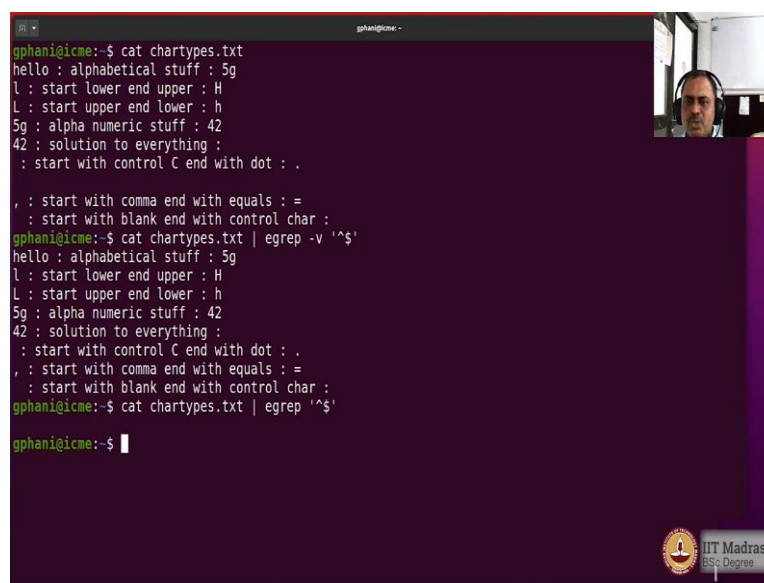
, : start with comma end with equals : =
: start with blank end with control char :
gphanigme:~$ cat chartypes.txt | grep '[:graph:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with control C end with dot : .
, : start with comma end with equals : =
: start with blank end with control char :
gphanigme:~$ cat chartypes.txt | grep '^[:graph:]'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
42 : solution to everything :
: start with comma end with equals : =
gphanigme:~$ cat chartypes.txt | grep '[:graph:]$'
hello : alphabetical stuff : 5g
l : start lower end upper : H
L : start upper end lower : h
5g : alpha numeric stuff : 42
: start with control C end with dot : .
, : start with comma end with equals : =
gphanigme:~$
```

Now, the counter class called graph is used to match any non-space character. So, you can see that it, every line except the blank line has some character which is not a space and

therefore that is getting matched here. And you could then have that as an anchor in the front. And you could see that those lines that contain any non-space character are being displayed in the front.

So, some of them which contain a control C that has been skipped here, a blank that has been skipped here, so the remaining ones only are there, only the remaining ones are being displayed. And I could also put an anchor at the end and you would see that the lines that contain a blank space at the end are skipped. So, a non-space character is tolerated at the end of the line.

(Refer Slide Time: 18:26)

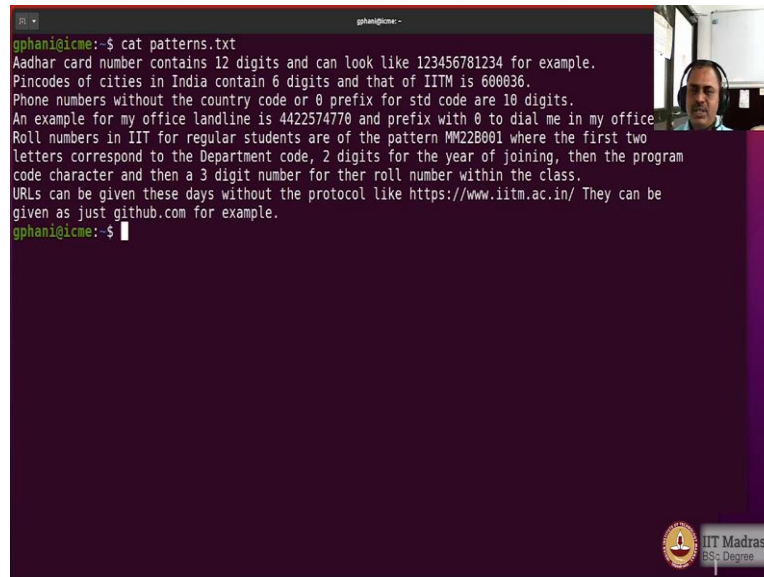
A terminal window with a dark purple background. The prompt is 'gphani@icme:~'. The user enters 'cat chartypes.txt' and the output is: 'hello : alphabetical stuff : 5g', 'l : start lower end upper : H', 'L : start upper end lower : h', '5g : alpha numeric stuff : 42', '42 : solution to everything :', and ': start with control C end with dot : .'. Then the user enters 'cat chartypes.txt | egrep -v \''\$''. The output is the same as before, but the last line is omitted. Finally, the user enters 'cat chartypes.txt | egrep \''^\$''. The output is empty. In the top right corner, there is a small video feed of a man with a beard and glasses. In the bottom right corner, there is a logo for 'IIT Madras BSc Degree'.

Now, let us say we would like to skip all the empty lines. So that can be done using this option. So, what happens is that the matching lines for the two anchor characters side by side would give empty lines and minus we would say that do not print that, print whatever does not match the empty line. So, the non-empty lines are getting printed out now. And the empty lines if you want to print, there is nothing funny about it just an empty line. So, that is not very useful.

But the with the minus v option, you could actually remove the empty lines from an input string, input stream and that is quite useful in processing where you expect something to be there on the line and you do not want to have empty lines coming in. So, you could use this feature of minus v, and then the two anchors side by side for the beginning and the end, so that you skip empty lines.

Time to time we are expected to pick up patterns from a given text file and then analyze them if they correspond to certain fields. So, now let us look at an example file and see whether we can pick up such patterns using the grep command.

(Refer Slide Time: 19:41)



```
gphanigme:~$ cat patterns.txt
Aadhar card number contains 12 digits and can look like 123456781234 for example.
Pin codes of cities in India contain 6 digits and that of IITM is 600036.
Phone numbers without the country code or 0 prefix for std code are 10 digits.
An example for my office landline is 4422574770 and prefix with 0 to dial me in my office.
Roll numbers in IIT for regular students are of the pattern MM22B001 where the first two
letters correspond to the Department code, 2 digits for the year of joining, then the program
code character and then a 3 digit number for their roll number within the class.
URLs can be given these days without the protocol like https://www.iitm.ac.in/ They can be
given as just github.com for example.
gphanigme:~$
```

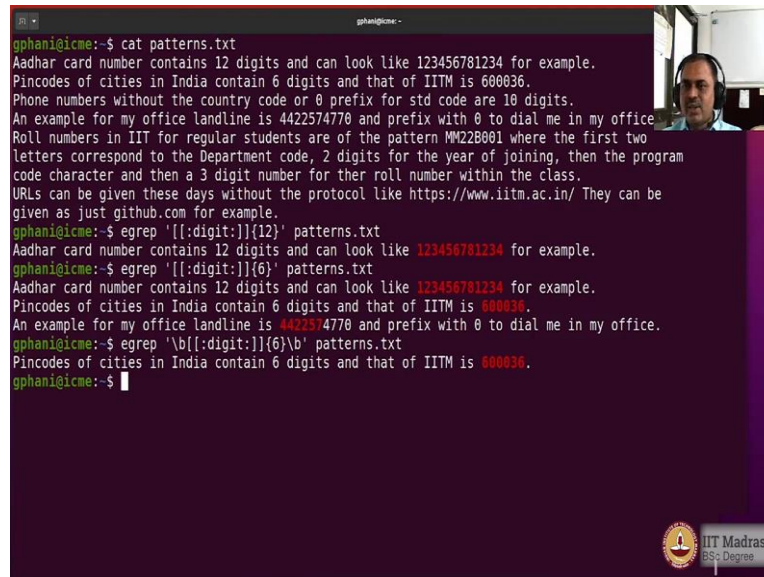
So, here I am providing a random piece of text which contains certain patterns. So, as we know, the Aadhar card numbers have about 12 digits. So, I am giving that as one example in one line. And then the pin codes usually have six digits so that of IIT Madras is a six digit one, 600036. There is no other location in Chennai that uses the same pin code. So, I am giving that as one example. And then the phone numbers for mobile phones are 10 digit and without the country code or the zero for a city dialing the landline numbers also are 10 digits for the cities. So, I am giving that as an example.

The roll numbers in IIT for regular students are following the pattern where the first two letters correspond to the department code. And then the two digits correspond to the year of joining. And then there is a program code which is in one letter followed by three digits corresponding to the roll number within the class. And it is not necessary that we need to give the URLs of our websites with the protocol like HTTPS or HTTP, but if you just give a string where there is a dot and on either sides of the dot there is a word which is alphanumeric, then that is also a valid URL. And you would have seen on your mobile phone that such strings are automatically made clickable and they could also be, and they could be interpreted as URLs.

So, now, let us see whether we can pick these numbers from this text using the grep command. Now, it would not work if any of these are spanning across two lines, because

grep would use only one line at a time for processing. But with that limitation very quickly we can actually put the grep knowledge that we have earned till now to some good use already.

(Refer Slide Time: 21:28)



```
gphan@icme:~$ cat patterns.txt
Aadhar card number contains 12 digits and can look like 123456781234 for example.
Pincodes of cities in India contain 6 digits and that of IITM is 600036.
Phone numbers without the country code or 0 prefix for std code are 10 digits.
An example for my office landline is 4422574770 and prefix with 0 to dial me in my office.
Roll numbers in IIT for regular students are of the pattern MM22B001 where the first two
letters correspond to the Department code, 2 digits for the year of joining, then the program
code character and then a 3 digit number for their roll number within the class.
URLs can be given these days without the protocol like https://www.iitm.ac.in/ They can be
given as just github.com for example.
gphan@icme:~$ egrep '[:digit:]{12}' patterns.txt
Aadhar card number contains 12 digits and can look like 123456781234 for example.
gphan@icme:~$ egrep '[:digit:]{6}' patterns.txt
Aadhar card number contains 12 digits and can look like 123456781234 for example.
Pincodes of cities in India contain 6 digits and that of IITM is 600036.
An example for my office landline is 4422574770 and prefix with 0 to dial me in my office.
gphan@icme:~$ egrep '\b[:digit:]{6}\b' patterns.txt
Pincodes of cities in India contain 6 digits and that of IITM is 600036.
gphan@icme:~$
```

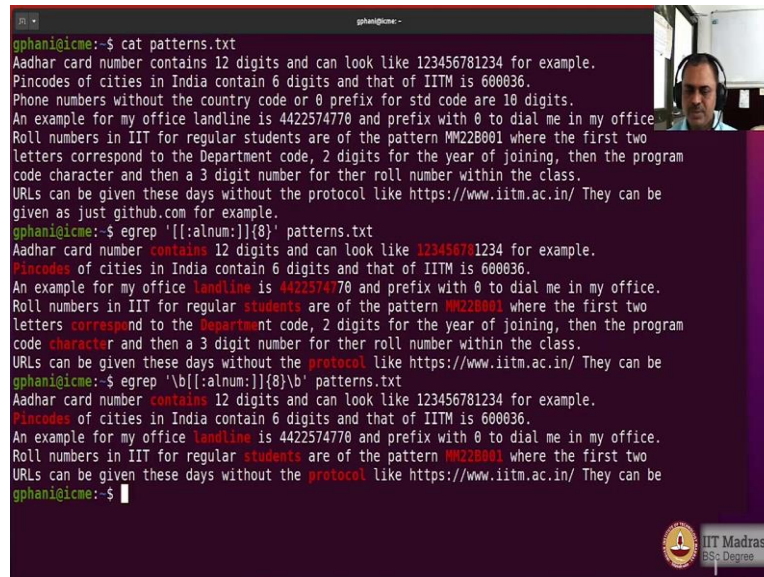
So, first, I would like to pick up the numbers which are having exactly 12 digits long so that the other card could come out of the text. So, I would like to always keep printing the original text so that it is useful for you. So, egrep and then the pattern will be given. The pattern should be such that any digit should be matched and it should have exactly 12 occurrences. So, 12 occurrences should be there. And you see that we have identified the Aadhar card as a matching pattern within the first line. So, no other line is getting displayed. So, that means that there is no confusion there.

Now, let us say we would like to pick up the pin code. So, now you see that pin code is six digit, so I want to pick those numbers that contain six digits. Now, you see that there is a problem here. It is actually picking up numbers which are matching six digits of the Aadhar card as well as for the phone number.

Now, how do I pick only the six digits and not more or not less, and that can be done using the word boundary. So, now, let us say that before and after the six digits, there must be a word boundary. And now you see that it would actually identify only the line that corresponds to the pin code. The phone number and the Aadhar card are not coming because they do not have a word boundary after the six digits are matched.

Now, I would like to identify what looks like a roll number. Now, you see that the roll number is actually a mix of both alphabetical as well as the digits. So, it is a bit painful. Now, alphanumerical characters of that particular length can also be tried out let us do that.

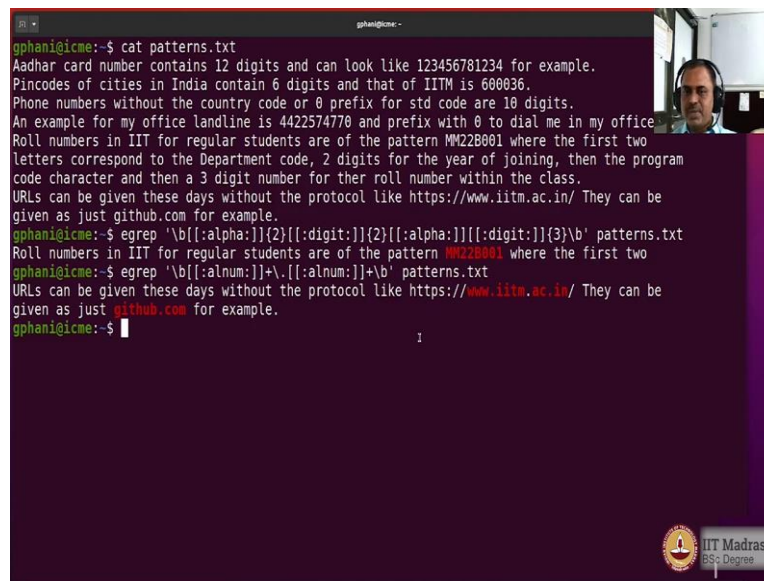
(Refer Slide Time: 23:22)



```
gphani@icme:~$ cat patterns.txt
Aadhar card number contains 12 digits and can look like 123456781234 for example.
Pincodes of cities in India contain 6 digits and that of IITM is 600036.
Phone numbers without the country code or 0 prefix for std code are 10 digits.
An example for my office landline is 4422574770 and prefix with 0 to dial me in my office.
Roll numbers in IIT for regular students are of the pattern MM22B001 where the first two
letters correspond to the Department code, 2 digits for the year of joining, then the program
code character and then a 3 digit number for their roll number within the class.
URLs can be given these days without the protocol like https://www.iitm.ac.in/ They can be
given as just github.com for example.
gphani@icme:~$ egrep '[:alnum:]{8}' patterns.txt
Aadhar card number contains 12 digits and can look like 123456781234 for example.
Pincodes of cities in India contain 6 digits and that of IITM is 600036.
An example for my office landline is 4422574770 and prefix with 0 to dial me in my office.
Roll numbers in IIT for regular students are of the pattern MM22B001 where the first two
letters correspond to the department code, 2 digits for the year of joining, then the program
code character and then a 3 digit number for their roll number within the class.
URLs can be given these days without the protocol like https://www.iitm.ac.in/ They can be
gphani@icme:~$ egrep '\b[:alnum:]{8}\b' patterns.txt
Aadhar card number contains 12 digits and can look like 123456781234 for example.
Pincodes of cities in India contain 6 digits and that of IITM is 600036.
An example for my office landline is 4422574770 and prefix with 0 to dial me in my office.
Roll numbers in IIT for regular students are of the pattern MM22B001 where the first two
URLs can be given these days without the protocol like https://www.iitm.ac.in/ They can be
gphani@icme:~$
```

So, let us try out the alphanumerical characters alnum and we see that it is actually having a length of eight. So, I put an 8 here. So, any eight character long alphanumerical string would be matched and you see that is actually not working out, because it is matching given regular words which do not have specific digits. Now, no point even asking for a word boundary because word boundary would also catch words which are exactly eight characters long so even that does not work. So, we need some other strategy that is where the character set that we have discussed would come off use. So, now let us try that out.

(Refer Slide Time: 24:13)



```
gphanigme:~$ cat patterns.txt
Aadhar card number contains 12 digits and can look like 123456781234 for example.
Pincodes of cities in India contain 6 digits and that of IITM is 600036.
Phone numbers without the country code or 0 prefix for std code are 10 digits.
An example for my office landline is 4422574770 and prefix with 0 to dial me in my office.
Roll numbers in IIT for regular students are of the pattern MW22B001 where the first two
letters correspond to the Department code, 2 digits for the year of joining, then the program
code character and then a 3 digit number for their roll number within the class.
URLs can be given these days without the protocol like https://www.iitm.ac.in/ They can be
given as just github.com for example.
gphanigme:~$ egrep '\b[[:alpha:]]{2}[[:digit:]]{2}[[:alpha:]]{3}\b' patterns.txt
Roll numbers in IIT for regular students are of the pattern MW22B001 where the first two
gphanigme:~$ egrep '\b[[:alnum:]]+\.[[:alnum:]]+\b' patterns.txt
URLs can be given these days without the protocol like https://www.iitm.ac.in/ They can be
given as just github.com for example.
gphanigme:~$
```

So, what we would do is that we would use a word boundary and then the alphabetical characters should occur exactly two times following by that a digit has to come exactly two times and then an alphabetical character has to come, and after that again a digit has to come exactly three times, after that there is again a word boundary. So, you see that the pattern when you type it is quite long, but it is quite logical.

What we are doing is I am asking for a word boundary, after that I am asking for an alphabetical character coming exactly twice, and then I am asking for a digit that comes exactly twice, and then again an alphabetical character, only one, and then I have got a digit that comes exactly thrice, after that again a word boundary.

So, this is a pattern that we are giving and you see that it identifies exactly the roll number and nothing else. So, by carefully using the character sets, we can actually identify the pattern of the string that we are trying to pick up. Now, for the URLs also we can actually use the same kind of a strategy. What is the strategy?

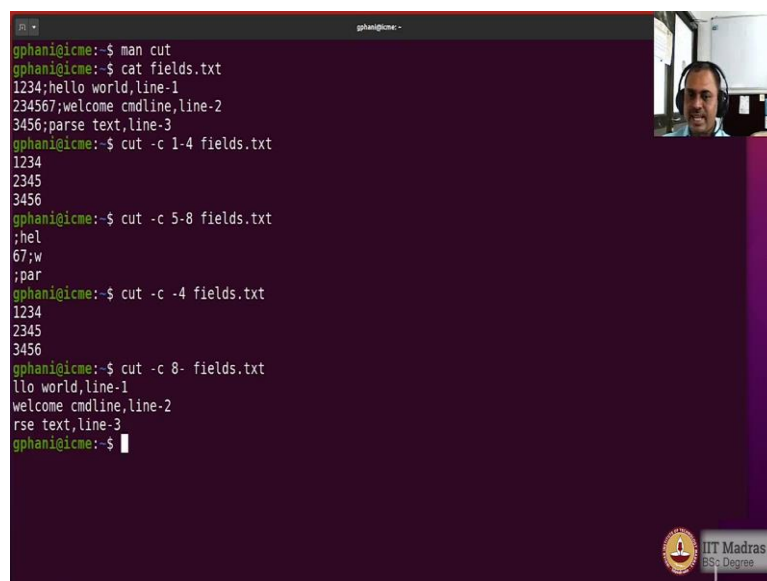
So, you know that the protocols are not necessary to be used now. Even the names like this also are possible. So, what we would do is as follows. And then an alphanumeric character can come any number of times, but not zero, after that a dot has to come, but the dot has to be also escaped. And again alphanumeric character can be there and that also has to come once or more, but definitely not zero times and then the word boundary.

So, you see now that it identifies the two URLs quite nicely. And so you can see that the usage of the word boundary here and then the plus sign which is available in the extended

regular expression engine under egrep and then the backslash to escape the meaning of the dot so that the dot is matched as it is, and then again, a word boundary to signal that is at the end of a particular word would actually help you in identifying the URLs quite easily.

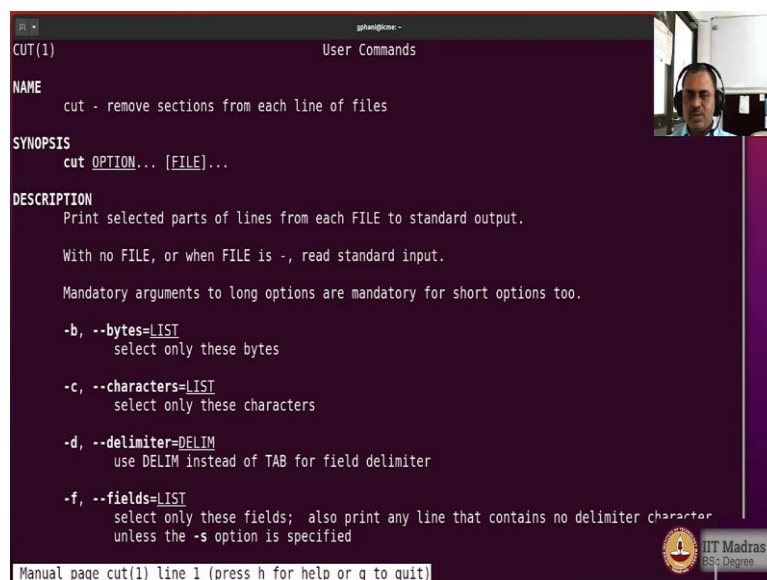
Now, I have used only alphanumerical characters. Underscore is also an allowed character. So, you can actually expand the scope of this matching to include every possible URL that can come in a random text that you are not expecting.

(Refer Slide Time: 27:06)



```
gphani@icme:~$ man cut
gphani@icme:~$ cat fields.txt
1234;hello world,line-1
234567;welcome cmdline,line-2
3456;parse text,line-3
gphani@icme:~$ cut -c 1-4 fields.txt
1234
2345
3456
gphani@icme:~$ cut -c 5-8 fields.txt
;hel
67;w
;par
gphani@icme:~$ cut -c -4 fields.txt
1234
2345
3456
gphani@icme:~$ cut -c 8- fields.txt
llo world,line-1
welcome cmdline,line-2
rse text,line-3
gphani@icme:~$
```

The terminal window shows a series of commands using the 'cut' utility. It first displays the manual page for 'cut', then cat's the contents of 'fields.txt'. Subsequent 'cut' commands are used to extract specific columns (characters) from the file, demonstrating how to isolate parts of a line based on character positions.



```
CUT(1) User Commands
NAME
cut - remove sections from each line of files

SYNOPSIS
cut OPTION... [FILE]...

DESCRIPTION
Print selected parts of lines from each FILE to standard output.

With no FILE, or when FILE is -, read standard input.

Mandatory arguments to long options are mandatory for short options too.

-b, --bytes=LIST
    select only these bytes

-c, --characters=LIST
    select only these characters

-d, --delimiter=DELIM
    use DELIM instead of TAB for field delimiter

-f, --fields=LIST
    select only these fields; also print any line that contains no delimiter character
    unless the -s option is specified

Manual page cut(1) line 1 (press h for help or q to quit)
```

This block shows the manual page for the 'cut' command. It provides a detailed description of the command's purpose, its synopsis, and a list of available options with their functions. The options include selecting by bytes, characters, delimiters, or fields.

Now, time to time, there is a need for us to trim a text from the top to the bottom or sideways. So, top to bottom, we have already learned how to trim by using the command a head and tail. So, the top end lines of a text file can be picked up using the head command, the bottom end lines can be picked up using the tail command. But what about trimming the text from

the left and right where we can use either the number of fields or the number of characters. So, for that, there is a function that is available. So, the command is called cut.

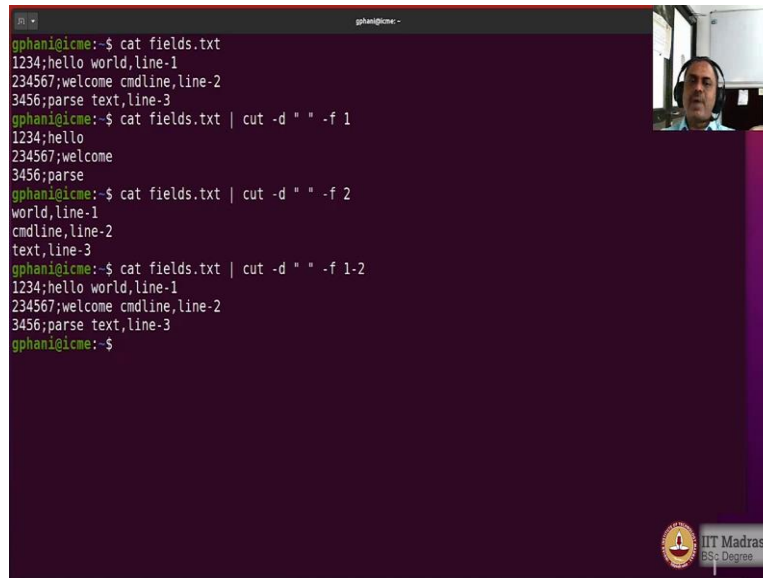
So, when you use `man cut`, you would see that it is enabling a horizontal trimming of the lines and that is used to remove sections from each of the files and it is going line by line. And it can actually trim using either the characters or fields, but for any field you also need to know what is the character used for the delimiter so minus `d` option also has to be there. And the default is tab, but you can provide any other delimiting character.

So, now I have prepared a simple text file using which we can actually start trimming to understand the usage of the cut command. So, `fields dot txt` is a simple three line file in which I have got fields like this. So, 1, 2, 3, 4 and then a semicolon, after that hello world has a blank, after that there is a comma. So, you can see that you could use the semicolon or the comma or the blank space as a field separator, and you can also use the number of characters as a way to cut. So, now let us go and inspect the how the cut command can be done.

Now, it can be used both ways. So, you can actually use `cut fields dot txt` and then you can actually give some options how to cut, so minus `c` is the number of characters and 1 to 4 tells you that from 1 to 4 characters from the beginning should be taken out and displayed. So you can see that the first four characters have been taken out and displayed. Now, you could also say that I want from 5 to 8 and I have got the next four characters coming on the screen.

Now, this does not pay attention to what was the field. It just goes as a series of characters and picks up the character sequence that you want so 1 to 4 or 5 to 8 or whatever. And if you do not specify the initial part, it will actually assume that you are going to have the, from the beginning okay. And if you do not specify the ending, it will also assume that you want it till the end. So, the option is there for the hyphen either the left side of the hyphen or the right hand side of the hyphen can be can be provided. If you provide both then it will be indicating as a range.

(Refer Slide Time: 30:00)



```
gphani@icme:~$ cat fields.txt
1234;hello world,line-1
234567;welcome cmdline,line-2
3456;parse text,line-3
gphani@icme:~$ cat fields.txt | cut -d " " -f 1
1234;hello
234567;welcome
3456;parse
gphani@icme:~$ cat fields.txt | cut -d " " -f 2
world,line-1
cmdline,line-2
text,line-3
gphani@icme:~$ cat fields.txt | cut -d " " -f 1-2
1234;hello world,line-1
234567;welcome cmdline,line-2
3456;parse text,line-3
gphani@icme:~$
```

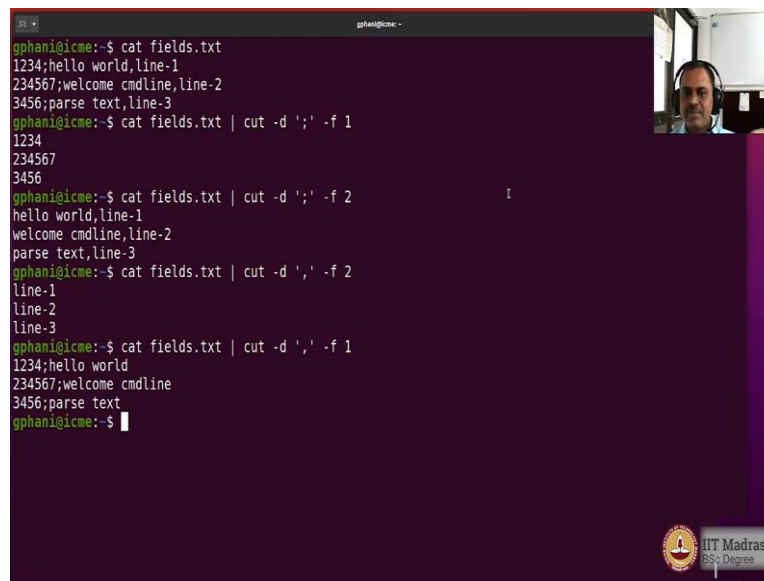
The image shows a terminal window with a dark background. The prompt is 'gphani@icme:~'. The user enters 'cat fields.txt' and the output is '1234;hello world,line-1', '234567;welcome cmdline,line-2', and '3456;parse text,line-3'. Then the user enters 'cat fields.txt | cut -d " " -f 1' and the output is '1234;hello', '234567;welcome', and '3456;parse'. Next, the user enters 'cat fields.txt | cut -d " " -f 2' and the output is 'world,line-1', 'cmdline,line-2', and 'text,line-3'. Finally, the user enters 'cat fields.txt | cut -d " " -f 1-2' and the output is the original three lines. In the top right corner, there is a small video feed of a man wearing a headset. In the bottom right corner, there is a logo for 'IIT Madras 65th Degree'.

Now, let us use a delimiter to pick up some of the fields. So, let us use it as a piped command. So, cut can take the input from a pipe. So, we will do that now. And let us use the delimiter as a space. So, now you will see that if I use the delimiter as a space, then this will be the first field and this is the second field. Now, I want to print only the first field.

Now, you see that it comes out quite nicely. And the length of each of these fields need not be same. So which means that when the number of characters is not what is uniform, but the way the characters are separated is what is uniform, then using space as a delimiter you can pick the first field. You can also pick the second field and you see that the second half of the text is coming up onto the screen.

Now, if you want both, you can also get it. So, then you have got both the, entire lines coming up, so that is actually not meaningful in this case. But in some situations where there are more than two fields, then you can actually take part of those fields also.

(Refer Slide Time: 31:07)

A terminal window with a dark purple background. The prompt is 'gphan@icme:~'. The user enters 'cat fields.txt' and the output is '1234;hello world,line-1', '234567;welcome cmdline,line-2', and '3456;parse text,line-3'. Then the user enters 'cut -d ';' -f 1' and the output is '1234', '234567', and '3456'. Next, the user enters 'cut -d ';' -f 2' and the output is 'hello world,line-1', 'welcome cmdline,line-2', and 'parse text,line-3'. Then the user enters 'cut -d ',' -f 2' and the output is 'line-1', 'line-2', and 'line-3'. Finally, the user enters 'cut -d ',' -f 1' and the output is '1234;hello world', '234567;welcome cmdline', and '3456;parse text'. A small video inset in the top right shows a man with a beard and headphones. In the bottom right corner, there is a logo for 'IIT Madras 65th Degree'.

Now, let us use the semicolon as a delimiter. And you would see that when I use semicolon, then the first field is this number, the second field is a string that contains a comma also in between. So, let us check that out. And you see that only the numbers are coming when I asked the first field. And when I asked the second field, I am getting the string, and then the comma. So, this is quite nice. You can change the delimiter and cut fields as per the delimiter, and pick up only those fields that you would like to have.

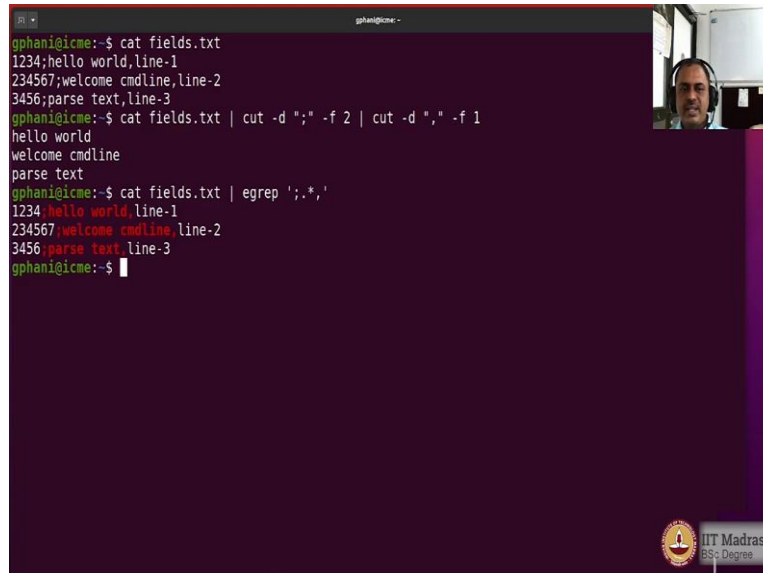
So, similarly, we can actually also use comma as a delimiter. And you see that when I have comma as a delimiter, the second field will contain the line number. So, I would like to take the second field, and you see only the line numbers are being displayed here, because after the comma, this is the second field, this is the second field. And if you want the first field before the comma, the rest of it actually is coming onto the screen, which actually does not have the same length for each of the lines.

So, the comma, the semicolon, the space or the tab, all of these can be used as field separators. So, look at the input stream once with your own eyes to see what is the delimiter that was used by the program which is throwing that particular stream at you. And then once you understand, you can go ahead and use the cut command to pick the fields that you want without having to tell the original developer to provide them as you like so you can just cut and take what you need from the output stream.

Now, you can also combine the streams by piping them multiple times. Let us say I would like to pick only the string in between the semicolon and the comma, so what I will do is I

will use semicolon as the first field separator and then comma as a second time. So, let us look at that now.

(Refer Slide Time: 33:01)



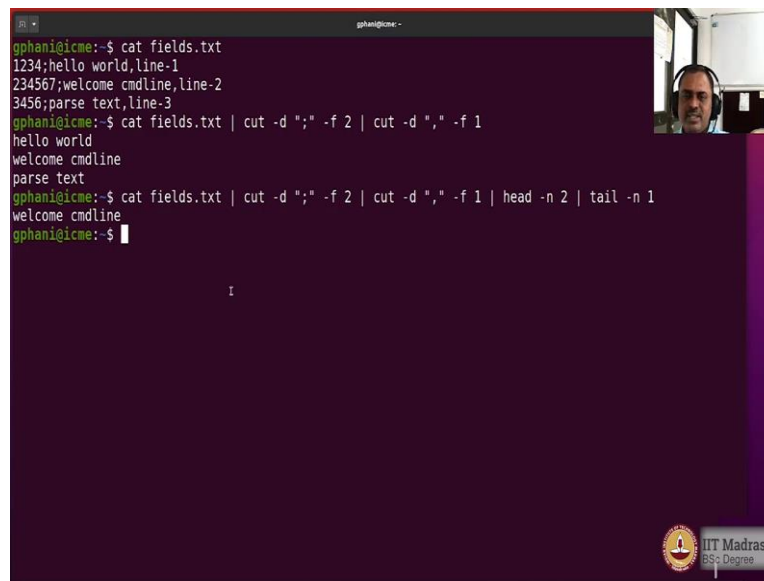
```
gphanigme:~$ cat fields.txt
1234;hello world,line-1
234567;welcome cmdline,line-2
3456;parse text,line-3
gphanigme:~$ cat fields.txt | cut -d ";" -f 2 | cut -d "," -f 1
hello world
welcome cmdline
parse text
gphanigme:~$ cat fields.txt | egrep ';\.,'
1234;hello world,line-1
234567;welcome cmdline,line-2
3456;parse text,line-3
gphanigme:~$
```

So, I want to pick only this part. So what I would do is, I would say cat fields, then cut the minus delimiter, I would first use this, and take the first field, and then I would use comma as the delimiter, and again take the first field. So, second field for the semicolon as delimiter because I want the second half of it and then after that I will separate using the comma and then I will take the first half of it. So, that is what I am meaning by this command. And you see that I am getting the string that is exactly between the semicolon and the comma.

So, now, this is actually not bothering about anything else. So, you can also achieve this to some extent using grep also. But in this case we are looking at the text in a slightly different manner. We are looking at as a set of fields that are separated and picking the fields which is slightly different way of looking at than a grep command. But we can also try to see whether the grep would work to give the same result. So, let us try that out now.

And you see that the grep command was able to match exactly the part that you would like to match. Now, but you have not cut it off. So, you have not yet trimmed the rest of it off. For the trimming part, you can pass on the output of the grep to said command which we would learn in the future classes, but for now you can see that you can also identify the string that you want by using pattern matching quite well. But the cut command is very useful because you can actually remove that portion and then go on to look at the output in a way that you would like to parse it with any other tool later on.

(Refer Slide Time: 35:11)



```
gphan@icme:~$ cat fields.txt
1234;hello world,line-1
234567;welcome cmdline,line-2
3456;parse text,line-3
gphan@icme:~$ cat fields.txt | cut -d ";" -f 2 | cut -d "," -f 1
hello world
welcome cmdline
parse text
gphan@icme:~$ cat fields.txt | cut -d ";" -f 2 | cut -d "," -f 1 | head -n 2 | tail -n 1
welcome cmdline
gphan@icme:~$
```

Now, you could actually combine the concept of cutting the text from left and right using the cut command and then the head and tail to actually cut the lines from the top and the bottom to pick a particular entry from the input string. So, let us try that out and I want to pick up only the welcome command line as the text. So, what I would do is, I will cut it from the left and right first and after that top and bottom. So, let us look at that now.

So, here I am getting the three lines. Now, I would like to only have the second line. So, I would do like this, head minus n and then two lines. The top two lines are being picked up. And in the top two lines I want the last line. So, I would pick the last line coming out. And now you see that I have picked up exactly the string welcome command line coming on to the screen and I have used many, many commands here.

So, I have catinated the contents of the file fields dot text and I have cut it using the field separator semicolon and picked up the second field and after that I have again cut it using the comma as a field separator and pick the first field and after that I have picked up the top two lines and then from those top two lines I have taken the bottom most line and then displayed it and we have got the string that we wanted.

So, you can actually combine the commands in various ways to get the output that you are interested. And it is up to our imagination to find the easiest way. And of course the most elegant way would be when we are experts in all these commands for which a lot of practice is required. So, keep practicing various of these commands and be thorough with regular expressions so that you make the best use of the tools that are available in the Linux command line environment.