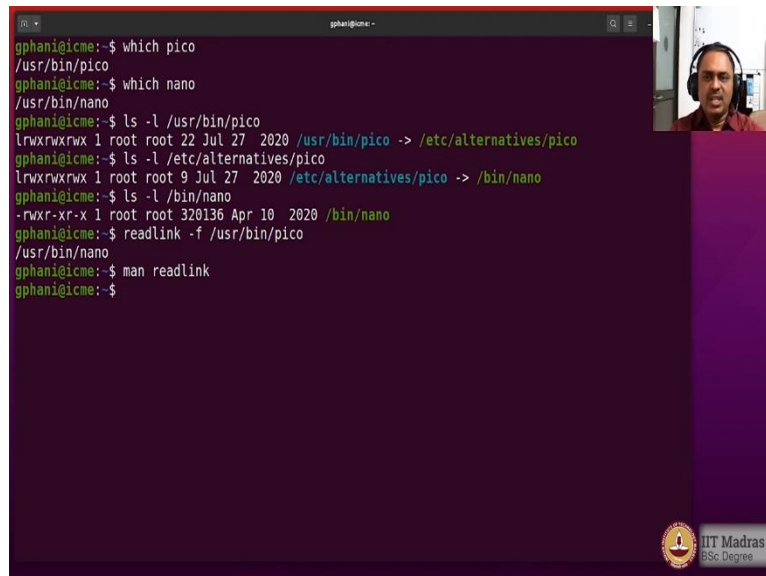


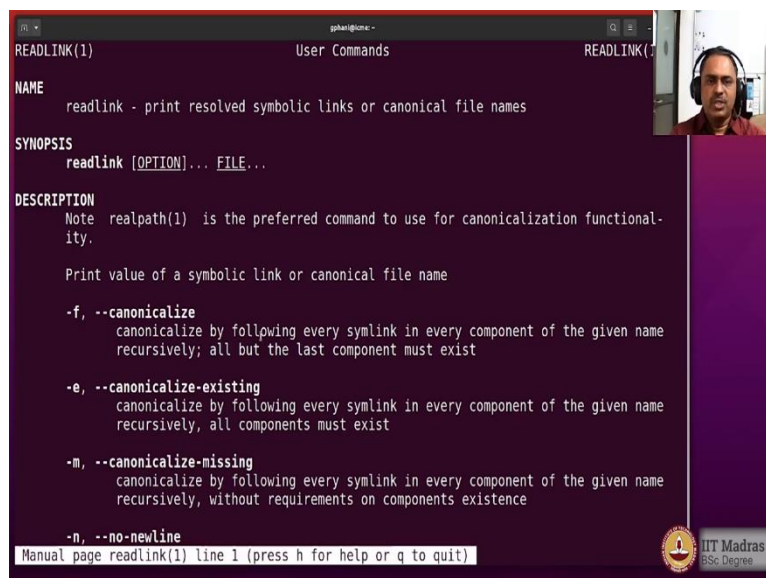
Systems Commands
Professor Gandham Phanikumar
Metallurgical and Material Engineering
Indian Institute of Technology, Madras
Command line editors – Part 02

(Refer Slide Time: 00:14)



A terminal window with a dark purple background. The user 'gphani@icme' is running several commands to find the location of 'pico' and 'nano' editors. The output shows that 'pico' is a symbolic link to '/etc/alternatives/pico' and 'nano' is a symbolic link to '/bin/nano'. The user also runs 'readlink -f /usr/bin/pico' to get the full path and 'man readlink' to view the manual page. A small video feed of the professor is visible in the top right corner, and an IIT Madras logo is in the bottom right.

```
gphani@icme:~$ which pico
/usr/bin/pico
gphani@icme:~$ which nano
/usr/bin/nano
gphani@icme:~$ ls -l /usr/bin/pico
lrwxrwxrwx 1 root root 22 Jul 27 2020 /usr/bin/pico -> /etc/alternatives/pico
gphani@icme:~$ ls -l /etc/alternatives/pico
lrwxrwxrwx 1 root root 9 Jul 27 2020 /etc/alternatives/pico -> /bin/nano
gphani@icme:~$ ls -l /bin/nano
-rwxr-xr-x 1 root root 320136 Apr 10 2020 /bin/nano
gphani@icme:~$ readlink -f /usr/bin/pico
/usr/bin/nano
gphani@icme:~$ man readlink
gphani@icme:~$
```



The same terminal window now displays the manual page for the 'readlink' command. It includes sections for NAME, SYNOPSIS, and DESCRIPTION. The DESCRIPTION section explains that 'realpath(1)' is preferred for canonicalization and lists several options: -f (canonicalize recursively), -e (canonicalize existing), -m (canonicalize missing), and -n (no new line). A footer indicates it's manual page readlink(1) line 1.

```
READLINK(1)                                User Commands                                READLINK(1)

NAME
  readlink - print resolved symbolic links or canonical file names

SYNOPSIS
  readlink [OPTION]... FILE...

DESCRIPTION
  Note  realpath(1) is the preferred command to use for canonicalization functional-
  ity.

  Print value of a symbolic link or canonical file name

  -f, --canonicalize
        canonicalize by following every symlink in every component of the given name
        recursively; all but the last component must exist

  -e, --canonicalize-existing
        canonicalize by following every symlink in every component of the given name
        recursively, all components must exist

  -m, --canonicalize-missing
        canonicalize by following every symlink in every component of the given name
        recursively, without requirements on components existence

  -n, --no-newline

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

There are two editors in Linux, which work on the terminal enrollment and give you a visual appearance. One is called the pico and other is called the nano, so let us look at those. So, which pico and it there is an editor like that, which nano. And sure enough, there is an editor like that.

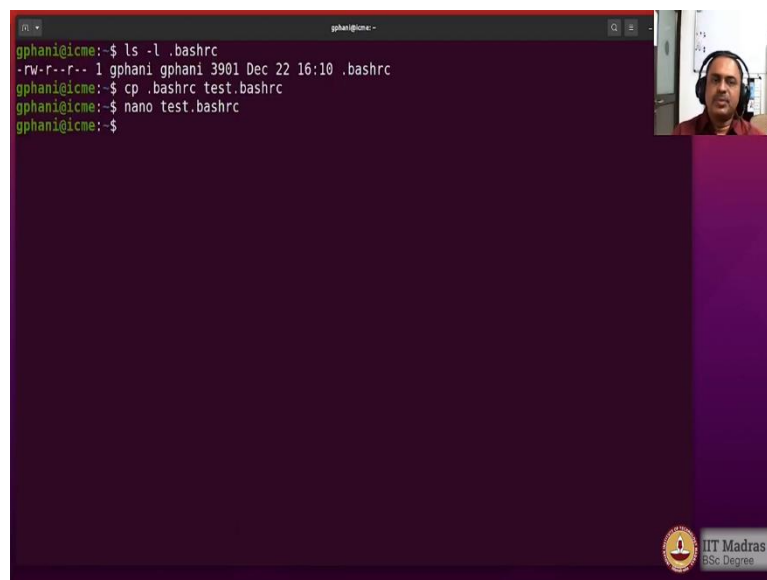
So, we want to look at how these are implemented. So, the pico editor, let us look at it, ls minus l usr bin pico, and it looks like it is a symbolic link to something else. So, now I want

to go and look at what is that file, and now that is a symbolic link to something else. Let me look at what is that and you can see that finally there is an application called nano.

So, when you are invoking pico editor in Ubuntu 20.04 at least, then actually you are only calling nano editor. And if you want to refer to these links, you could also use a command called readlink minus f and slash user slash bin slash pico, and it will tell you ultimately after whatever number of hops what is the final executable it is pointing to, so readlink is a very good utility.

So, you can actually know about it by using the manpage. So, it says resolved symbolic links are canonical file names has to be printed on the screen, so that is a very useful utility that you may want to look at. So basically, whether you run pico or nano, it is a same editor, so we may as well call it by the actual name namely nano. So, nano editor can be started.

(Refer Slide Time: 01:45)

A terminal window with a dark purple background. The prompt is 'gphani@icme:~'. The commands and output are: 'ls -l .bashrc' followed by '-rw-r--r-- 1 gphani gphani 3901 Dec 22 16:10 .bashrc'; 'cp .bashrc test.bashrc'; and 'nano test.bashrc'. A video call inset in the top right shows a man with a headset. An IIT Madras BSc Degree logo is in the bottom right corner.

```
gphani@icme:~$ ls -l .bashrc
-rw-r--r-- 1 gphani gphani 3901 Dec 22 16:10 .bashrc
gphani@icme:~$ cp .bashrc test.bashrc
gphani@icme:~$ nano test.bashrc
gphani@icme:~$
```

```
GNU nano 4.8 test.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
  *i*) ;;
  *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit          ^R Read File    ^M Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line
```

```
GNU nano 4.8 test.bashrc

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "*" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
#shopt -s globstar

# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi
```

```
GNU nano 4.8 test.bashrc

#alias vdir='vdir --color=auto'

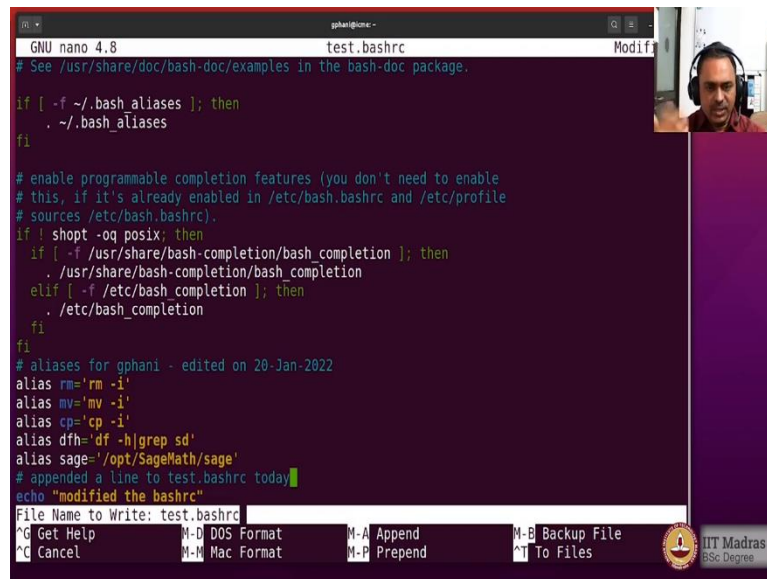
alias grep='grep --color=auto'
alias fgrep='fgrep --color=auto'
alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
# sleep 10; alert
alias alert='notify-send --urgency=low -i "${[ $? = 0 ] && echo terminal || echo error}" "${@}'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.
```



```
GNU nano 4.8 test.bashrc
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
  . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

# aliases for gphani - edited on 20-Jan-2022
alias rm='rm -i'
alias mv='mv -i'
alias cp='cp -i'
alias dfh='df -h | grep sd'
alias sage='/opt/SageMath/sage'
# appended a line to test.bashrc today
echo "modified the bashrc"
File Name to Write: test.bashrc
^G Get Help      ^M-D DOS Format  ^M-A Append      ^M-B Backup File
^C Cancel        ^M-M Mac Format  ^M-P Prepend     ^T To Files
```

Now, which file we would like to edit? So, I want to use the nano editor to show you the so called bashrc file. So, my ls minus l dot bashrc, so this is a file which is not very big, and it is used every time that you open the bash shell. And certain commands that are written in this bashrc are going to be executed.

So, I do not want to make any mistake of changing it and then ending up with some errors, so what I would do is, I will make a copy of it and then go on to change it. So, I would say copy dot bashrc, and then I want to copy it to my home directory something else, so I would say test dot bashrc.

Now, I have a file which I can go and play with it. So, I would use the, I would use the nano editor to play with the test dot bashrc. So, you can see that the nano editor looks fairly visual because you have a cursor that is blinking at the corner here you can see that blinking at the very first character, and it is also sensitive to the syntax. So, any line in a shell script that starts with the hash is a comment. So, you can see that it is showing the comments in a different color, and then there are some loops that we will actually understand in a moment, which are applicable for a batch language.

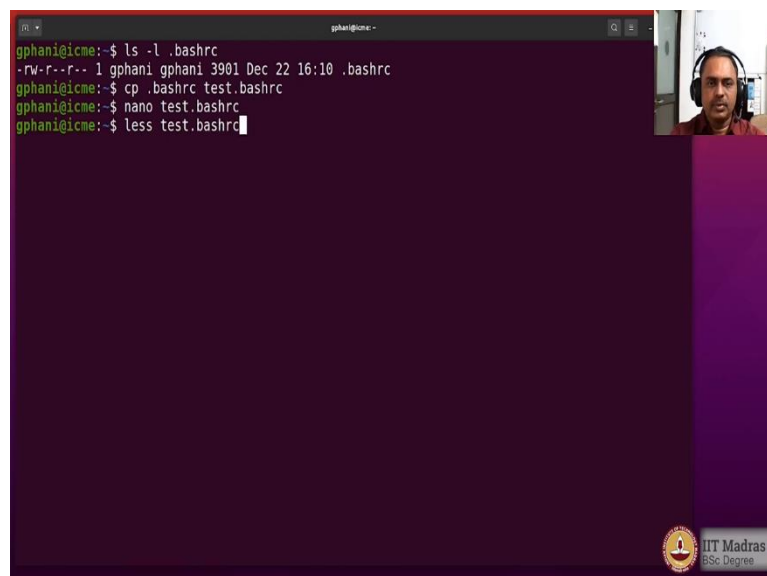
And then there are lines which are actually active, so which are shown in white color. And you can also separate out the words from operators, so they are also colored differently. Even options are being colored in differently. And so, this is a shell script which is being executed every time you run the bash executable. So, all these commands are being run and they are basically preparing the environment for your bash shell to be there.

Now, here are some aliases, which I have added to my bashrc, so that when I type ll it will execute the command ls minus aLF and when I type the command l it will run ls minus CF. So, it is just my own way of shortening the commands by using aliases. And you can put anything else in the end and those comments will always be applicable. So, you could actually go on to edit now.

So, let us say I want to edit this file and first initially start with a command. So, you can say edited on and then we can give a date. So, let us say on 20th Jan 2022. So, you can write already without any effort dislike almost like a notepad of Windows environment and you can start making some changes. So, I would also give one more comment in the end appended a line to test dot bashrc today.

And then I want to also have something on the screen to be displayed. So, let me have say echo, then modified a bashrc. So, I am writing a command there, so this command will be executed if you run that particular script. Now, we have done some editing, we were able to go up or down, left or right using the keys, and we are able to add text directly. And how do we save this? So, Ctrl O would write the file to whatever file name you have given. So that is the file which you open, so we will overwrite those content using this command Ctrl O. And once you have already written, then you can come out using Ctrl X.

(Refer Slide Time: 05:21)



```
gphani@icme:~$ ls -l .bashrc
-rw-r--r-- 1 gphani gphani 3901 Dec 22 16:10 .bashrc
gphani@icme:~$ cp .bashrc test.bashrc
gphani@icme:~$ nano test.bashrc
gphani@icme:~$ less test.bashrc
```

The image shows a terminal window with a dark purple background. The terminal output shows the user 'gphani' at host 'icme' in the directory '~'. The user runs 'ls -l .bashrc', which shows the file permissions, owner, size, and date. Then, the user runs 'cp .bashrc test.bashrc' to create a backup. Next, the user enters 'nano test.bashrc' to edit the file. Finally, the user runs 'less test.bashrc' to view the contents of the file. In the top right corner of the terminal window, 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'.

```
gshan@home: ~$ cat ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
  *) ;;
  *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "*" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
test.bashrc
```

```
gshan@home: ~$ shopt -s checkwinsize

# If set, the pattern "*" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
#shopt -s globstar

# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
  debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
  xterm-color|*-256color) color_prompt=yes;;
esac

# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
#force_color_prompt=yes

if [ -n "$force_color_prompt" ]; then
  if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then
    # We have color support; assume it's compliant with Ecma-48
    # and ...
```

```
gshan@home: ~$

if [ "$color_prompt" = yes ]; then
  PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\[\033[00m\]\$ '
else
  PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
  xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
  *)
    ;;
esac

# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
  test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
  alias ls='ls --color=auto'
  #alias dir='dir --color=auto'
  #alias vdir='vdir --color=auto'

  alias grep='grep --color=auto'
  alias fgrep='fgrep --color=auto'
```



```
gphani@gphani: ~$
# Add an "alert" alias for long running commands.  Use like so:
# sleep 10; alert
alias alert='notify-send --urgency=low -i "${?} = 0" && echo terminal || echo error)'
istory|tail -n|sed -e '\s/\s*[0-9]\+\s*//;s/[]\s*alert$/\s*alert$/'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

# aliases for gphani - edited on 20-Jan-2022
alias rm='rm -i'
alias mv='mv -i'
```

```
gphani@gphani: ~$
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

# aliases for gphani - edited on 20-Jan-2022
alias rm='rm -i'
alias mv='mv -i'
alias cp='cp -i'
alias dfh='df -h|grep sd'
alias sage='/opt/SageMath/sage'
# appended a line to test.bashrc today
echo "modified the bashrc"
[END]
```

Now, you can go ahead and look at the file, once more using that to say, less command. And you can see that at the bottom, you can make out the changes that you have just made. So, you can see that the editing we have done just now is all available in the file, so it was successful editing of the file. So, you could use nano to edit files quite comfortably. And some of these features, let us just review the commands that you can use to explore the nano editor.

(Refer Slide Time: 05:55)

nano

File handling

- Ctrl+S Save current file
- Ctrl+O Offer to write file ("Save as")
- Ctrl+R Insert a file into current one
- Ctrl+X Close buffer, exit from nano

Editing

- Ctrl+K Cut current line into cutbuffer
- Alt+6 Copy current line into cutbuffer
- Ctrl+U Paste contents of cutbuffer
- Alt+T Cut until end of buffer
- Ctrl+] Complete current word
- Alt+3 Comment/uncomment line/region
- Alt+U Undo last action
- Alt+E Redo last undone action

Search and replace


- Ctrl+Q Start backward search
- Ctrl+W Start forward search
- Alt+Q Find next occurrence backward
- Alt+W Find next occurrence forward
- Alt+R Start a replacing session

Deletion


- Ctrl+H Delete character before cursor
- Ctrl+D Delete character under cursor
- Alt+Bsp Delete word to the left
- Ctrl+Del Delete word to the right
- Alt+Del Delete current line

Information

- Ctrl+C Report cursor position
- Alt+D Report line/word/character count
- Ctrl+G Display help text



<https://www.nano-editor.org/dist/latest/cheatsheet.html>



nano

Moving around


- Ctrl+B One character backward
- Ctrl+F One character forward
- Ctrl+← One word backward
- Ctrl+→ One word forward
- Ctrl+A To start of line
- Ctrl+E To end of line
- Ctrl+P One line up
- Ctrl+N One line down
- Ctrl+↑ To previous block
- Ctrl+↓ To next block
- Ctrl+Y One page up
- Ctrl+V One page down
- Alt+↑ To top of buffer
- Alt+↓ To end of buffer

Special movement


- Alt+G Go to specified line
- Alt+] Go to complementary bracket
- Alt+↑ Scroll viewport up
- Alt+↓ Scroll viewport down
- Alt+< Switch to preceding buffer
- Alt+> Switch to succeeding buffer

Various

- Alt+A Turn the mark on/off
- Tab Indent marked region
- Shift+Tab Unindent marked region
- Alt+V Enter next keystroke verbatim
- Alt+N Turn line numbers on/off
- Alt+P Turn visible whitespace on/off
- Alt+X Hide or unhide the help lines
- Ctrl+L Refresh the screen



<https://www.nano-editor.org/dist/latest/cheatsheet.html>



So, here are some features of the nano editor that we have just now looked at. So, you could save the file using Ctrl S, you could write the contents of the buffer into the file using Ctrl O, and you could also read a different file content into the current file using Ctrl R at the current position of the cursor and Ctrl X, you could come out of the window.

If you have made changes and you want to come out it would always prompt you whether you want to save the changes or not, so that way it is quite user friendly. And you have various editing features to cut and paste stuff. You also have the ability to search and replace using a ctrl Q or Alt Q. And then you also have the ability to delete a text either character by character or word by word or line by line. And at any point you can also look at what is the

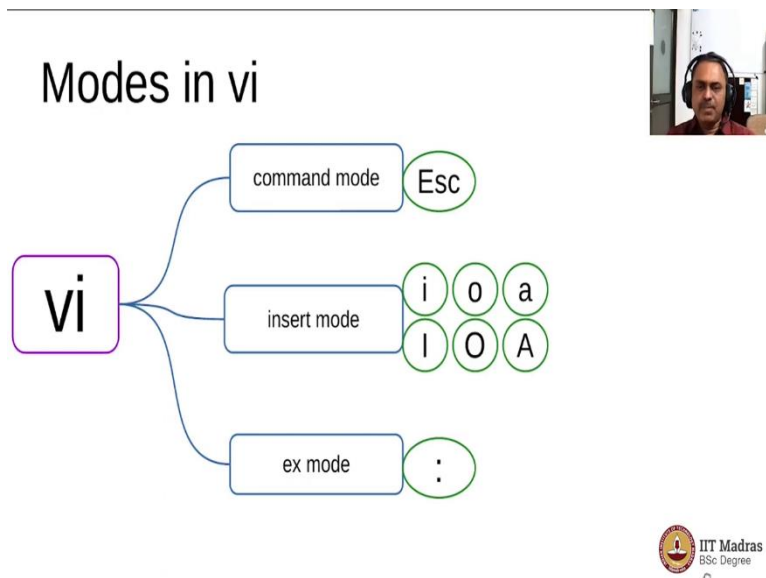
position of the current cursor with respect to the whole file, and you could also look at this help right on the screen.

And there are various ways by which you can move around within the nano editor, either character by character or word by word or going to the start of the line or end of the line upwards or downwards. Even block by block in terms of text blocks or even page by page, and also the very first line or the last line.

And you also have the ability to use features which are quite interesting for programmers. For example, you can actually use Alt N to make the line numbers visible or not like a toggle. And that is quite useful when you are writing code, so you do not have to have the line numbers always visible, because it seems to clutter the window, but you could have them switched on whenever you like to refer.

So, that way, nano is a very powerful and quite a good editor. And look at the link that I have provided here to have a copy of the cheat sheet and print it out and practice all the comments so that you make yourself comfortable with this editor. And it is pretty good that you can actually use it for most of your programming needs in a terminal environment.

(Refer Slide Time: 08:08)



People say that the most complex editor in the Linux world is Vi. That is because it has acquired so many features over the years that the list of commands itself runs into several pages. So, I will show you some commands that are very useful for you to get going in the Vi editor. Because if you can make yourself comfortable with this editor, then you almost do not need to know any other editor in any other environment because the newly improved version

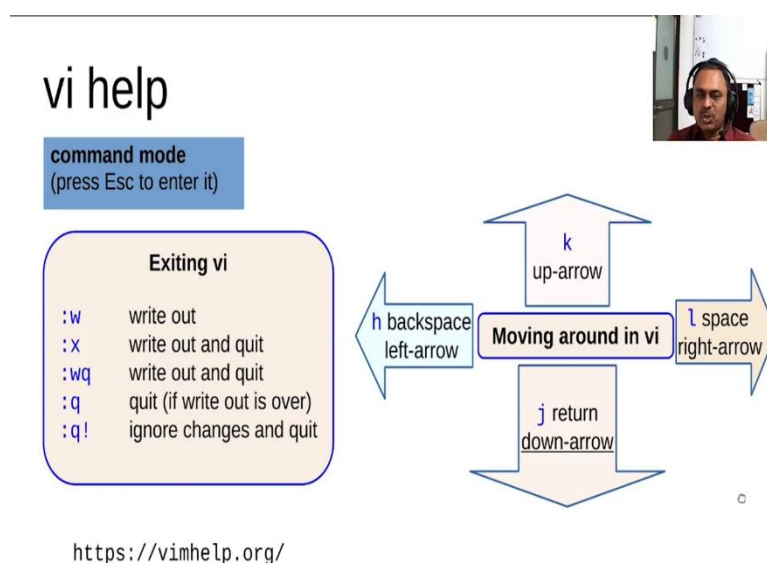
of the Vi editor called the vim is available for all operating systems. So, there are three major modes of Vi editor. There is a command mode which by default is opened when you open the file. And by pressing escape symbol, you can go into the command mode.

You can press multiple times because it does not hurt, so escape being the last character on the left side of the keyboard left top, so you can keep pressing that to ensure that you are in the command mode. And if you are in the insert mode then what you type will actually go into the text buffer or into the file, and there are some characters which trigger the insertion, so i small i would insert the characters from the current position of the cursor, and O will enter insert a new line and A will also insert text as an appending, and therefore all these characters I O U both the capital and the small would actually start the insert mode.

And at the bottom of the window in the Vi editor you will also see a help that says that you are right now in insert mode, so whatever you type will actually go into the file, so it is like an alert. And when you press escape and come out of the insert mode into the command mode, you could also enter what is called ex mode.

That is, in the command mode, you can press colon, and then the cursor will be going to the bottom of the screen, and there you can write some commands, which are same as the ex commands or the ed comments we have seen just now, and so that is called the ex mode. So, there are three modes, and we will see them right now as a demonstration.

(Refer Slide Time: 10:25)



So, here are some very basic commands that you need to be familiar. So, very often, when you get into a an editor, first thing you have to figure out is how to get out of the editor, so W would actually write the contents of the file, and X would write, save the contents and then quit wq would actually save and quit so is the same as x.

Sometimes you have not made any change, so you can just quit without making any changes to the file by using q. And if you have actually made some changes, but you do not want to save them, so you can quit by pressing colon q exclamation, so that you are coming out by force without writing any changes in the buffer to the original file.

Now, navigation within the file in a visual mode is done using multiple keys. So, you can use the arrow keys, no problem, they will work. But sometimes, if the terminal has not been configured properly, particularly for non-English keyboards, then you can always rely on the h, k, l and j characters for navigating within the text file.

So, I have put an underline here for j because on the keyboard you can feel the j character and you see that there is a small line below it, so that is a guide for your finger so you can place it on j, and then your middle finger will be on k, and the ring finger will be on l, so that is usually the way that Vi editors would actually start working.

And the k would actually take you upwards in the screen. And after some practice, you start getting used to it, so you do not need the arrow keys at all, but they also work if there is no problem with the settings of the terminal. And the backspace would remove the text only if you are in the insert mode, but you are in the command more then the backspace is same as moving to the left.

Similarly, if you press the spacebar, if you are in the insert mode, it will insert a space, but if you are in the command mode, it would actually take the cursor to the right side. So, watch out that what you press will be interpreted differently depending on which mode you are, and by default, you are in the command mode. And you have to always watch out at the bottom of the screen whether you are in the insert mode or in the command mode.

(Refer Slide Time: 12:37)

vi command mode



Screen manipulation

cntrl+f	Scroll forward one screen
cntrl+b	Scroll backward one screen
cntrl+d	Scroll down half screen
cntrl+u	Scroll up half screen
cntrl+l	Redraw screen
cntrl+r	Redraw screen removing deleted stuff

Moving around

0	start of the current line
\$	end of the current line
w	beginning of next word
b	beginning of preceding word
:0	first line in the file
1G	first line in the file
:n	nth line in the file
nG	nth line in the file
:\$	last line in the file
G	last line in the file

vi command mode



changing text

r	replace single char under cursor
R	replace chars from cursor till Esc
cw	change word under cursor, from current char till Esc
cNw	change N words, from current char till Esc
C	change chars in current line till Esc
cc	change current line till Esc
Ncc	change next N lines, starting from current, till Esc

deleting text

x	delete single char under cursor
Nx	delete N chars from cursor
dw	delete one word, from the char under cursor
dNw	delete N words, from the char under cursor
D	Delete rest of the line, from the char under cursor
dd	delete current line
Ndd	delete next N lines, starting from current

vi command mode



copy-paste text

yy	copy current line to buffer
Nyy	copy next N lines, including current, into buffer
p	Paste buffer into text after current line

searching text

u	undo previous action
/string	search forward for string
?string	Search backward for string
n	move cursor to next occurrence of string
N	move to previous occurrence of string

:se nu set line numbers
:se nonu unset line numbers

So, here are some simple commands. So, Ctrl f, and Control b will take the screen forward or backward, so you can move the text within the window, and the ctrl d and ctrl u would actually move the text by half a screen only, so that what you are reading is still within the viewport, but some more lines are available for you to read and edit.

And sometimes let us say you have made some changes and in a very slow terminal you would like to refresh the screen Ctrl l would do that the same thing as what you do on the shell Ctrl r would also do it by deleting some characters that you have done with some commands, so if the terminal was not fast enough, then you can actually ask it to refresh it for you by pressing these keys.

You can actually navigate within the file quite well using Vi. So, you could go to the beginning of the line by pressing 0 and end of the line pressing dollar, beginning of the word following the cursor with the w and beginning of the preceding word by b. And by using colon you can also go to a specific line by giving the number there so you can give either 0 or 1 to go to the slide, and then any number to go to that particular number of line.

And you could also use the capital G to go to a specific line by using 1G, 2G etc. And the G alone would take you to the end of the file, and that is the same as dollar with a colon in front. So, you can go to the x mod and press dollar, it will go to the last line, which is same as what we have seen in ed editor, and it will also work in Vi.

And there are some commands like how to replace a single character under the cursor using R and then the capital R would replace indefinitely till you press escape, and how to change a word so you can cw change the word, and you can keep typing the new word until you press escape. And if you want you can change multiple words by using CNw, N is the number. So, C to W would mean the next two words would actually be changed to be what you type, and capital C would actually change the text from the cursor all the way to the end of the line.

And what you enter up to pressing the escape key would be inserted into the particular line. And the entire line can also be changed by using the CC command. So, let us try these out in a demo to understand how they work. You could also do some minute editing by using x to delete a character exactly at the cursor position, and you can also delete a particular number of characters by using Nx.

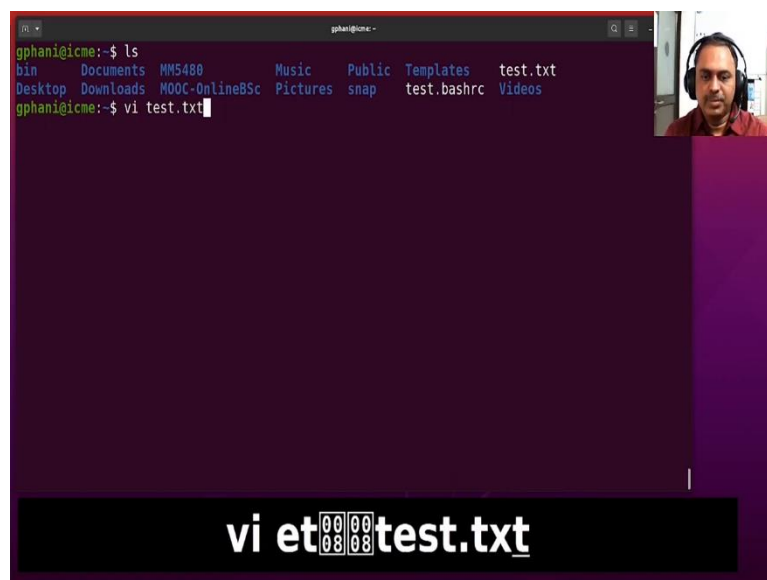
Let us say five characters I want to delete, so 5x will delete exactly five and dw would delete the word, d to w would delete two words, capital D would delete from the cursor to the rest of

the line, a dd would delete the whole line, and let us say 10dd would delete 10 lines from the current cursor position.

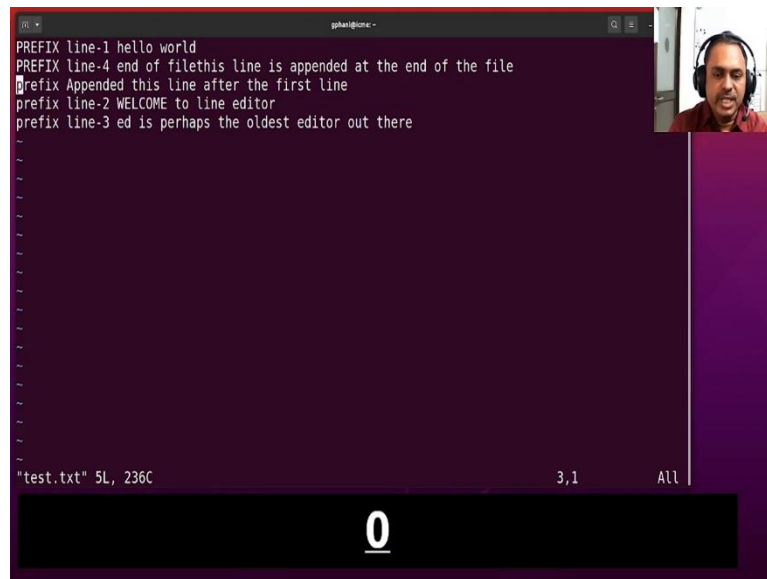
And you could do copy paste. So, why is for yanking? Yanking is like, you know, grabbing or taking away, so y is used to copy stuff, so yy would copy the current line, and 5yy would copy five lines 10yy would copy 10 lines, so you can copy how many ever lines from the cursor position, and then you can paste them anywhere in the buffer by using the cursor to move around. And then at that particular position, if you press a p, then below that line, whatever you have copied will be pasted.

And when you think you have done a mistake, you can always press u and then undo the change that you have done. And you can search for whichever line contains a particular pattern using slash and then a string which is a regular expression, and you can do the forward searching or backwards searching by using either forward slash or question mark followed by the regular expression for searching. And once you have started searching, you can either move forward in identifying those locations or backward by using either small n or capital N.

(Refer Slide Time: 16:34)



The image shows a terminal window with a dark purple background. At the top, there's a window title bar that says "gphani@icmc: ~". Below that, the terminal shows the command "ls" being executed, resulting in a list of files and directories: "bin", "Documents", "MMS480", "Music", "Public", "Templates", "test.txt", "Desktop", "Downloads", "MOOC-Online8Sc", "Pictures", "snap", "test.bashrc", and "Videos". Below the list, the command "vi test.txt" is entered. In the bottom right corner of the terminal window, there is a small video call inset showing a man with a beard and headphones. At the bottom of the terminal window, there is a black bar with white text that reads "vi et[00][00]test.txt".



So, let us look at a demo of the Vi editor. Now, I have enabled the display of keys that I have pressed at the bottom of the screen, so that what I press will be visible to you. So, you will see at the bottom of the screen. So, I am now going to put ls, and then enter, I am seeing the text here. Now, I want to edit the test dot txt to start with, so vi test dot text.

I press j I am going down, I press l, I am going to the right side down the line. So, you can see I press multiple times, and you can see how many times I am pressing and how the cursor

(Refer Slide Time: 18:18)

A screenshot of a terminal window titled "gshan@cs.cmc:". The terminal shows the following commands and output:

```
PREFIX line-1 hello world  
PREFIX line-4 end of filethis line is appended at the end of the file  
PREFIX Appended this line after the first line  
prefix line-2 WELCOME to line editor  
prefix line-3 ed is perhaps the oldest editor out there
```


Below the text are several tilde (~) symbols representing blank lines. At the bottom left, it says "-- INSERT --". At the bottom right, it shows "3,1" and "ALL".
In the top right corner, there is a small video inset showing a man wearing headphones.

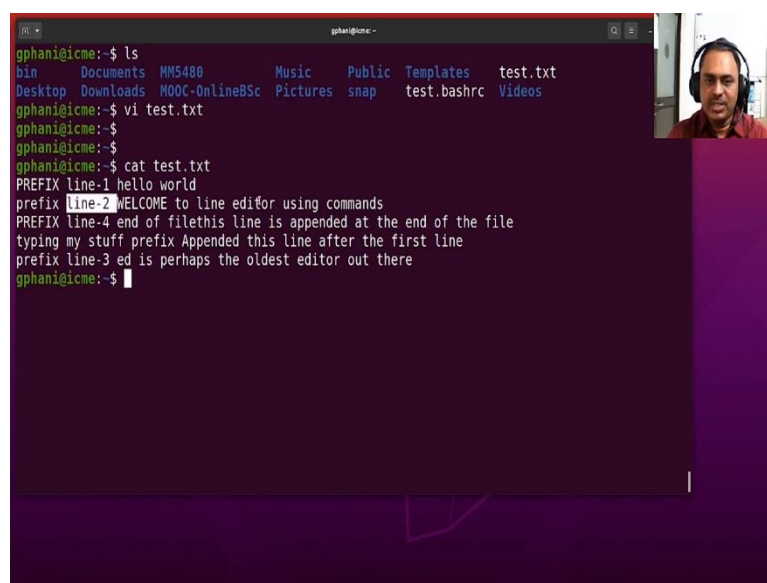
Now, I press colon, and then 1, enter, and it will take me to the first line. I press shift G, and it will take you to the last line. I press three capital G, it will go to the third line. If I want to add some text, I can press i and then I will be in the insert word. So, you can watch what is happening. I press i, and now you see at the bottom there is an insert more that is displayed. And now whatever I type would actually enter into the buffer, so I can right now.

So, I can type something and it all goes into the text. Now, I am still in the insert mode. So, if I want to come out so that I can move around, I press escape, and then now I am in the command mode again. So, now if I press anything, so I you see that I am just navigating now. So, watch out for example here again.

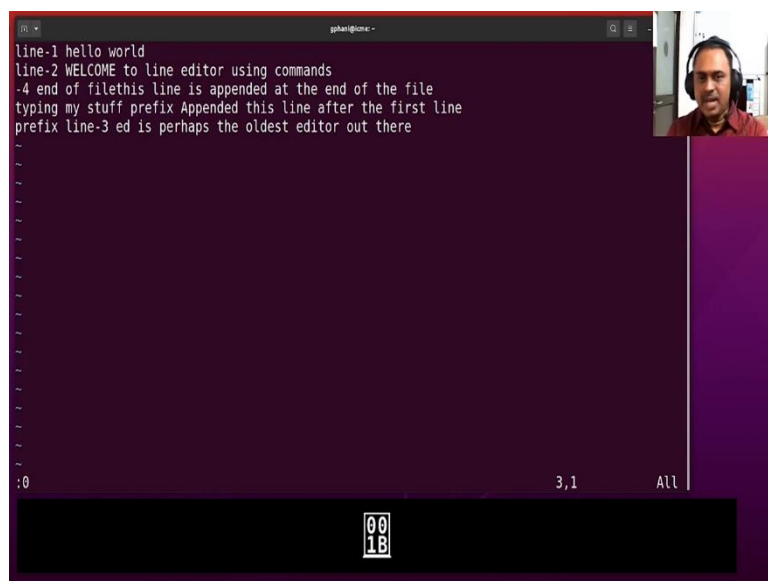
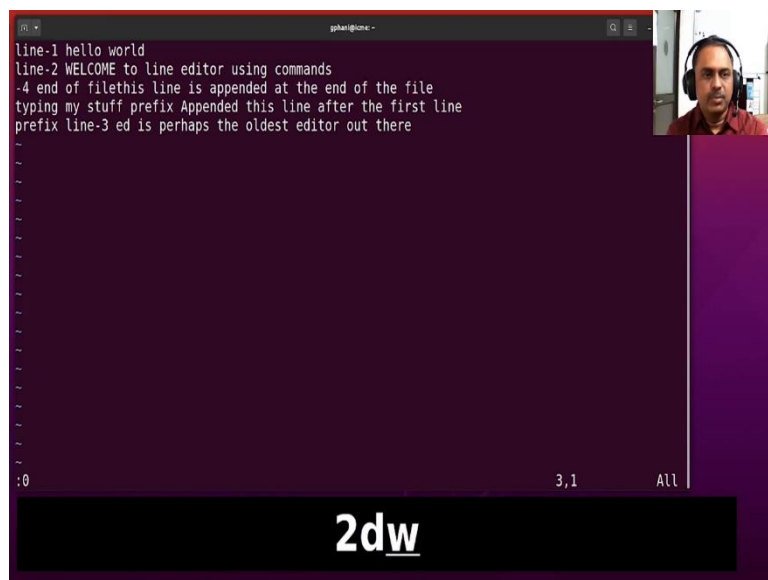
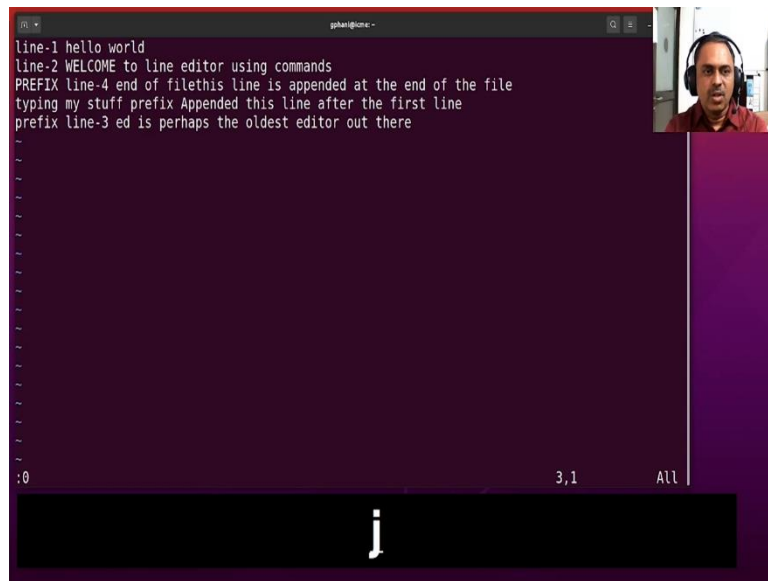
I press a, I am in the insert mode again. So, I can say welcome to line editor commands let us say, escape. Now, I come out then I am able to again, navigate. Now, let us say I do not like this line, so I want to delete it, so I press dd and the line is gone. But it is in the buffer as a memory so I can print it after the current position, so I can go up. And after this current position, I want to press p and then whatever line I have deleted has come there.

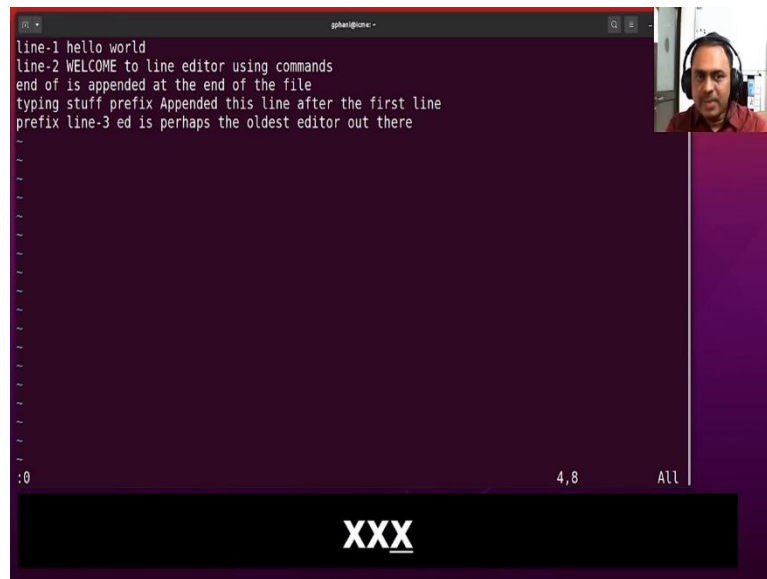
Now, this editing whatever I have done, I can actually save and then come out so I can confirm that I am in the command mode. So, I press escape once or twice to ensure that I am in the command mode, press colon, and then I am now in the ex mode. In ex mode if I press w and q that it would write the file and then save and come out. I am coming out of the Vi Editor into the shell.

(Refer Slide Time: 20:26)



```
gphani@icme:~$ ls
bin      Documents  MW5488      Music      Public  Templates  test.txt
Desktop  Downloads  MOOC-Online85c  Pictures  snap    test.bashrc  Videos
gphani@icme:~$ vi test.txt
gphani@icme:~$
gphani@icme:~$ cat test.txt
PREFIX line-1 hello world
prefix line-2 WELCOME to line editor using commands
PREFIX line-4 end of filethis line is appended at the end of the file
typing my stuff prefix Appended this line after the first line
prefix line-3 ed is perhaps the oldest editor out there
gphani@icme:~$
```

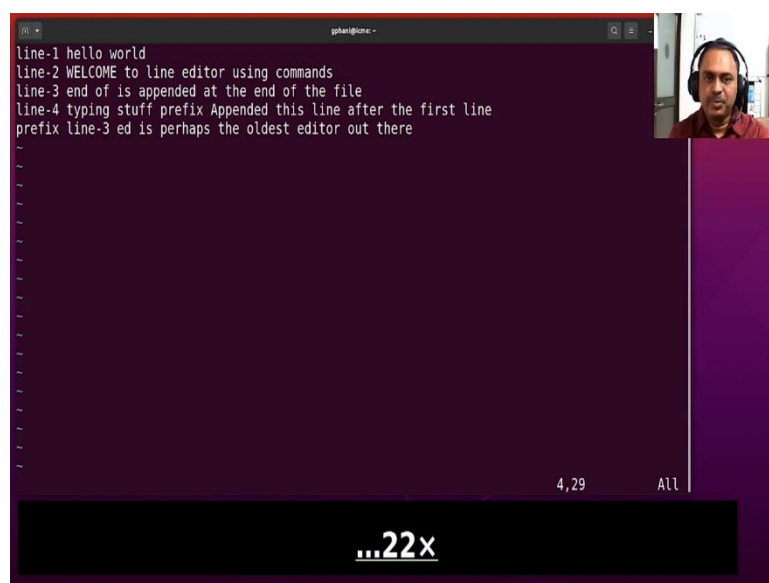
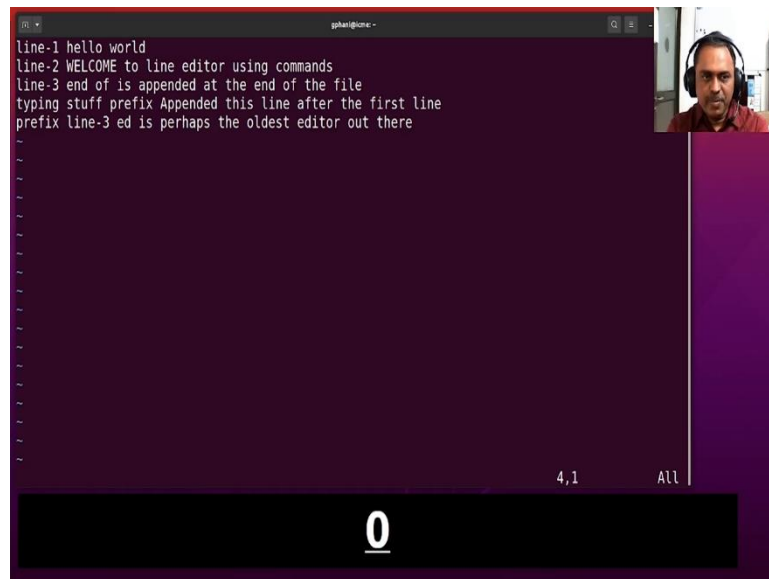



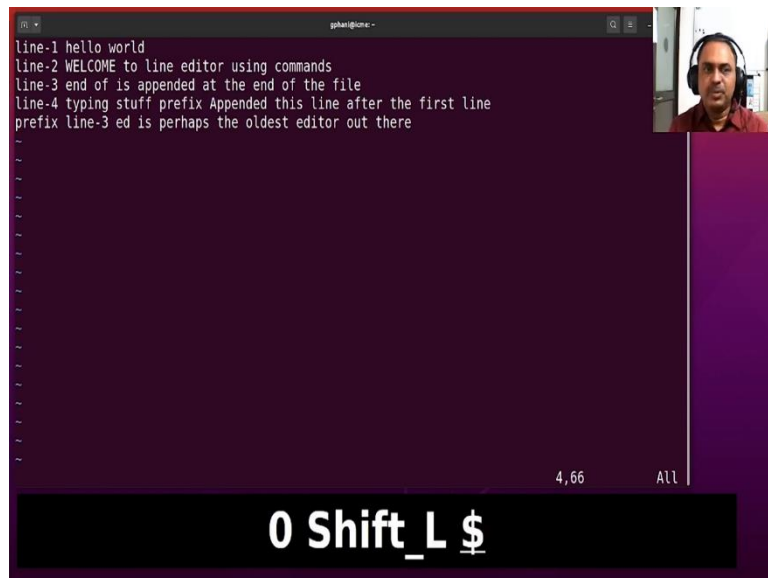


What I now do is that I delete two words 2dw. And you can see the two words have been deleted, escape I am back to command mode. Now, I want to delete only one character I can do that with x. One character I can delete, one more character I can delete. You can delete

(Refer Slide Time: 22:04)

[illegible]

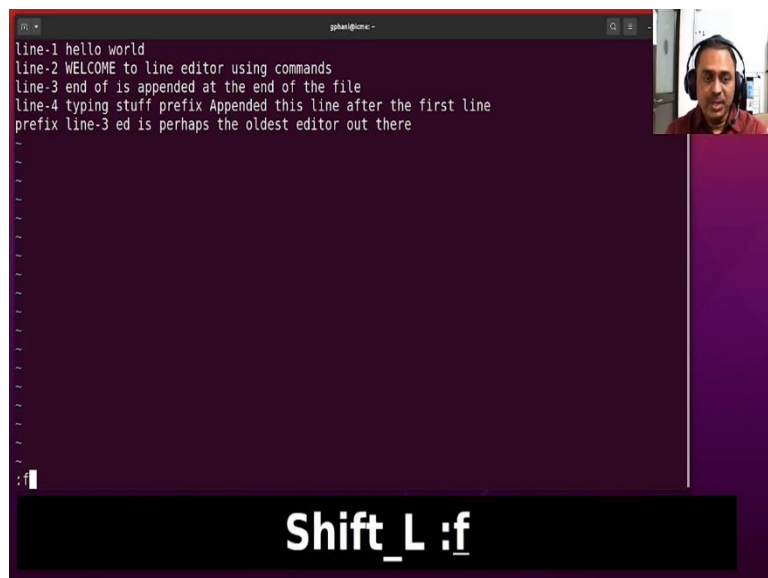




line-1 hello world
line-2 WELCOME to line editor using commands
line-3 end of is appended at the end of the file
line-4 typing stuff prefix Appended this line after the first line
prefix line-3 ed is perhaps the oldest editor out there

4,66 All

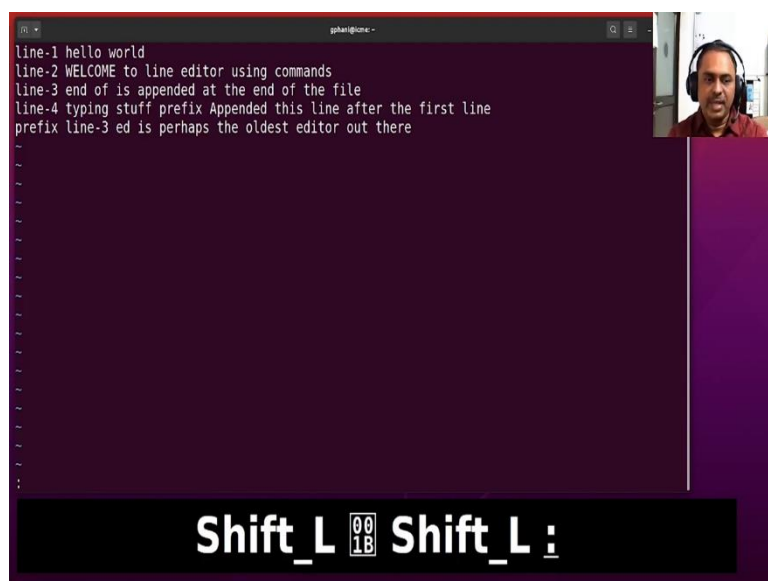
0 Shift_L \$



line-1 hello world
line-2 WELCOME to line editor using commands
line-3 end of is appended at the end of the file
line-4 typing stuff prefix Appended this line after the first line
prefix line-3 ed is perhaps the oldest editor out there

:f

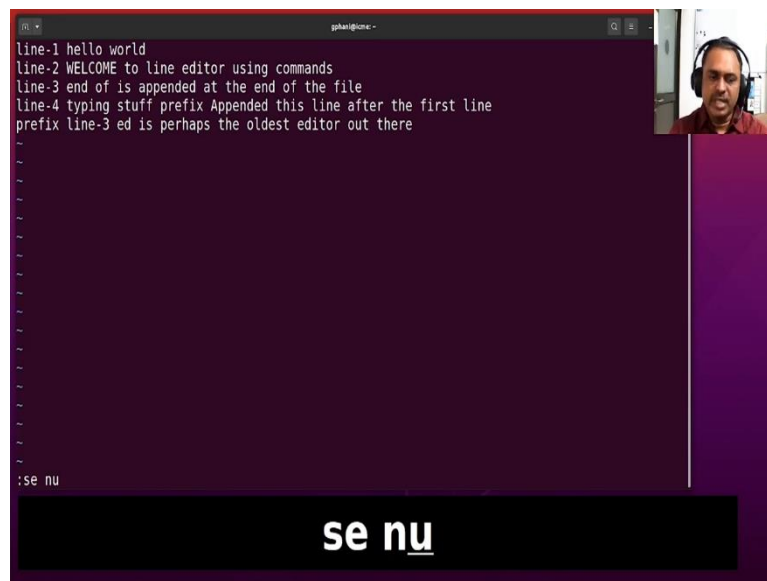
Shift_L :f



line-1 hello world
line-2 WELCOME to line editor using commands
line-3 end of is appended at the end of the file
line-4 typing stuff prefix Appended this line after the first line
prefix line-3 ed is perhaps the oldest editor out there

:

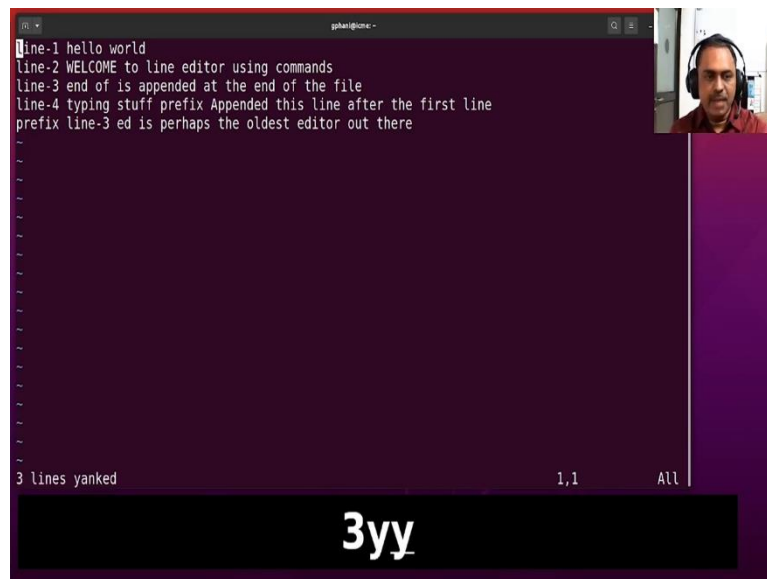
Shift_L 00 1B Shift_L :



And you can insert at any position so you can navigate and start writing, so I can navigate to a particular position and then press I and I can insert text at that particular position. I am now typing line 3. And if I am done with editing for now I press escape and I can start again going to navigate. So, go to the beginning of the line using pressing 0 and then press i to start typing line hyphen 4 space and then escape I am now in command mode. When I am in command mode, if I press the space, I am actually going to go right side 0 and dollar.

Now, I can actually also see which file I am right now. So, F was the command that we used to know what is the file name. So, it also shows that even in Vi editor. And there are some nice features where you can actually see the visual nature of this editor. So, I go to the ex mode by escape followed by a shift colon, so the time in the ex mode, and I type a command like set numbers senu set number, and it will actually show me the line number, and which is

(Refer Slide Time: 22:32)

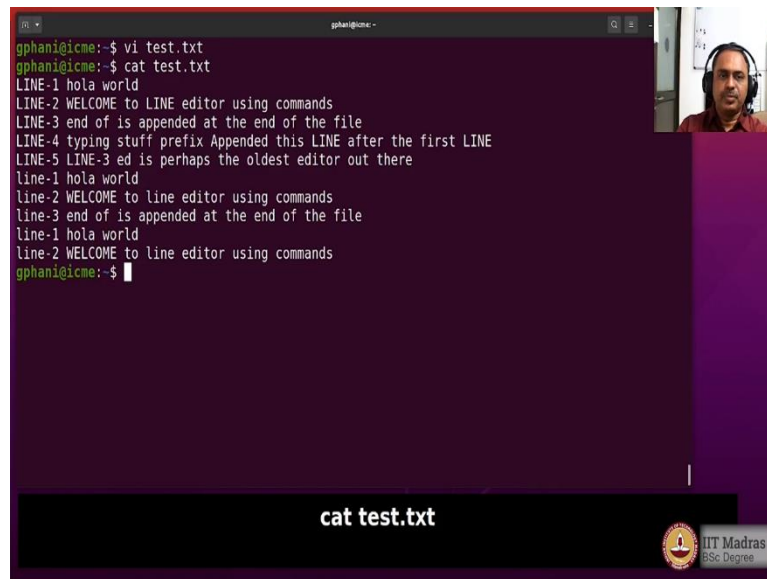




Now again, I go to the first line 1, and I can now copy just two lines. So, if you press y and enter it is same as pressing 2yy, so I press y and enter, so two lines will be copied, and I come to the bottom and press p to copy and paste so I am copying and pasting two lines. So, y enter is same as pressing 2yy. Now, I have done all these. So, you can see that there is something in the front, line one, line two, etc, but this line does not have it, so I want to change it, so cw will change the word.


(Refer Slide Time: 24:34)





```
gphani@icme:~$ vi test.txt
gphani@icme:~$ cat test.txt
LINE-1 hola world
LINE-2 WELCOME to LINE editor using commands
LINE-3 end of is appended at the end of the file
LINE-4 typing stuff prefix Appended this LINE after the first LINE
LINE-5 LINE-3 ed is perhaps the oldest editor out there
line-1 hola world
line-2 WELCOME to line editor using commands
line-3 end of is appended at the end of the file
line-1 hola world
line-2 WELCOME to line editor using commands
gphani@icme:~$
```

cat test.txt



So, cw, changing the word and until time that I typed something, so I would type it as a line hyphen 5, and then I press escape. So, now the escape means a changing is over. I come here, and I want to remove the space by pressing x and that space is remote. I can press escape come into the command line mode. So, you can see that I am now cleaning it up and using the navigation, as well as the commands of the Vi editor.

Now, the Ed editor has a search and replace features, which we can also use. So, let us see the numbers. So, from the first to the fifth line I want to apply that search and replace features, so let us try that out. From the first to fifth line, I want to substitute. What do I want to substitute, I want to substitute a line with a capital word. And g is for global substitution without that only one substitution, so I just leave it as one substitution first occurrence.

And you will see that the first five lines, I have changed the word from small line to capital line. And you will see that there is one more line here, which would be changed here also, which will be changed if I had a g operator. So, let us try that out. I undo the change right now I have made with u, and now again, I will type the ex-command mode.

So, 1 comma 5 substitute line with capital line and g, so which means multiple times it occurs in the line then it would be doing multiple substitutions. So, watch out what is happening to this word, this word and this word here, because they are going to be targeted by the g. And you see that those have been also modified along with the first occurrence of line in the beginning of each of these lines.

Now, I could actually do for the entire buffer also if I wish. So, wherever hello is coming, I want to change it to some other word that is hola, so I can actually do like this percentage.

Percentage mister there is entire buffer that is what we discussed in Ed editor. So, percentage substitute hello with let us say hola, and everywhere it should be affected. So, you see that wherever hello is there, it will be replaced with hola.

Now, this is something that is very useful for code because I want to change the name of a particular variable, so wherever that variable name is occurring it has to be replaced and so this is kind of a useful feature for that. And the line numbers are being displayed I can just set no numbers and the clutter is now removed.

So, you can see that this is how we go around making simple additions or deletions copy paste or search and replace in a Vi editor. So, I want to come out without saving it, so there is always a alert for that too. So, I can say that, save and come out and there we are out of that now. Cat test dot text, whatever text has been typed is there.