Module 9

Text Editors

"Wherever smart people work, doors are unlocked"

- Steve Woznaik

Learning Objectives

By the end of this module, you should be able to:

- Know different text editors available in GNU/Linux.
- Perform text editing functions in vim editor.
- Customize vim editor
- Perform text editing functionality in GNU/Emacs editor

1 Introduction

A text editor is program that allows opening, viewing, and editing text files. Text editors are used by almost everyone for a wide variety of purposes. Software programmers use text editors to write and edit the source code while system administrators use them to create or modify configuration files. Editors are ideal tools for anyone who needs to write quickly, read source code or create/edit text files. Text editors are often provided with operating systems or software development packages, and can be used to change configuration files and programming language source code.

Unlike word processors, text editors do not add formatting to text like changing the font size or creating bold headings, instead it focuses on editing plain text. In Microsoft Windows, edit command is a very basic text editor with minimal features and low capabilities. Application programs like Word Processors contain features like formatting text, adding content that enables text to appear in boldface and italics, to use multiple fonts, and structuring the text into columns and tables. These capabilities were once associated only with desktop publishing packages, but are now common in Word Processors like MS Office, LibreOffice,

OpenOffice etc. Word processing applications such as wordpad or applications that are part of office suites cannot be called basic text editors as they add lot of extra formatting information (that is not visible usually). These extra information may make the configuration files unusable. This is the main reason why we advocate using basic text editors for programming or administration purpose.

A plain text file uses a simple character set such as ASCII to represent numbers, letters, and a small number of symbols. The only non-printing characters in the file, usable to format the text, are newline, tab, and formfeed. Advanced text editors come with more features like search and replace, undoing, etc. Typical features of text editors include string searching algorithm, cut/copy/past features, undo/redo facilities, navigation facilities, syntax highlighting and ability to handle UTF encoded text.

Text Editors can be classified into two broad categories viz. Line Editors and Screen Editors. In a line editor, modifications can be applied to a single line or group of lines, for example adding an extra space at the beginning of certain lines can be accomplished easily with line editors. ed is the most common line editor. The screen editor presents the screen of text at a time where a user can navigate across multiple lines or scroll the screen to see different parts of the text document. The difference between both the editors can be best explained by giving an analogy of storing records in Oracle database and querying them using SQL which resembles line editors and storing records in Excel sheet resembles screen editors. Another category of editors is the stream editor that deals with input and an output stream of file, sed is an example of stream editor. This module discuses screen editors.

2 Graphical Editors for desktop environment

There are various editors available with the default installation of GNU/Linux. Text editors can be roughly bifurcated into two categories viz. basic editors and advanced editors. Basic editors come with minimal functionality for editing the programs while advanced editors support buffer management, customizing of the editors etc.

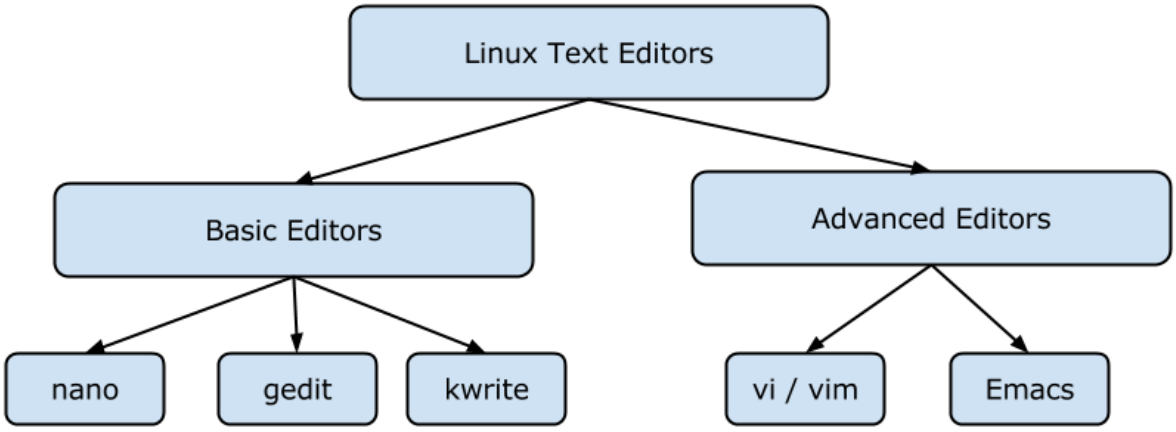

Figure 1: Classification of few GNU/Linux editors

Graphical editors need a GUI interface while others can be invoked from the command prompt. We can use editors from remote system with the help of telnet/ssh protocol. Graphical editors like gedit (GNOME desktop) and kwrite (KDE desktop) are very easy to use. They are similar to Notepad in MS Windows. Few other graphical editor variants like kate and kedit are also found in KDE desktop. Graphical editors are easy to learn, straight-forward and don't require much training.

3 The Vim Editor

Ken Thomson was the first person to develop a line editor called 'ed' for the Unix system. A more powerful line editor called 'ex' was developed by Bill Joy, at the University of California, Berkeley. Later, in 1976 he developed a screen oriented interface for the 'ex' known as 'vi' (Visual in ex). Vi (pronounced as 'vee-eye') editor is also referred as Visual editor that allows to view the file on screen and edit the contents by navigating the cursor or using line numbers. Various clones of vi like Elvis (for MINIX), Nvi (for BSD), Vim (for Amiga) were developed. The improved version of 'vi' was developed by Bram Moolenaar and was known as 'vim' editor. Vim editor has found its place in most of the GNU/Linux Distros.

The vim editor (or Vi IMproved) was first released by Bram Moolenaar in 1991 as a clone of the Unix editor vi. Vim for the Unix platform started to become an alternative to the vi editor. Vim is a superset of vi's functionality and is nearly 99% compatible with vi. Today, vim is a feature-rich, fully configurable editor. It supports syntax-highlighting of more than 200 different programming languages, folding, undo/redo, multiple buffers/windows/tabs, and a lot of other features. Figure-2 illustrates the default screen of vim editor obtained by entering the vim command. In most of the Linux distros, vim is aliased to the name vi.

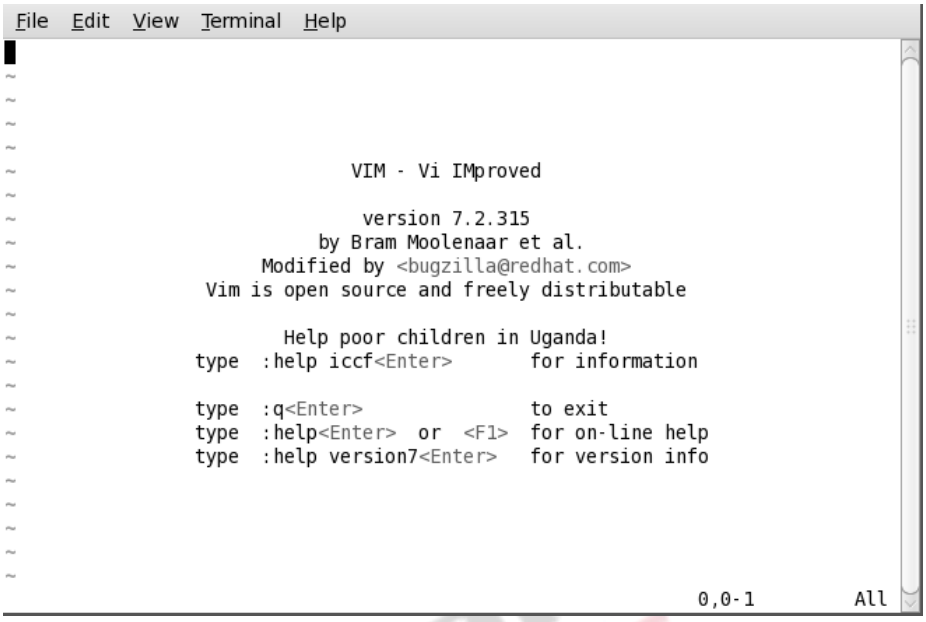

Figure 2: Startup screen of Vim

Vim is the editor that uses different modes. This can be explained by an analogy of driving a geared vehicle. A geared vehicle may use different gears like forward gears, backward gear or neutral gear, similarly vim has different modes of operation like input mode, command mode and ex mode (last line mode). It must be noted that driving a geared vehicle requires some amount of practice and human reflexes must be set accordingly, Vi/Vim editor may appear to be tricky initially but later will be quite appealing to work with. Vim has continued to be the widely used editor by most of the programmers.

3.1 Modes of the Vim Editor

During 1970s, the computer systems did not come with mouse or any sort of pointing device. The keyboards were too small with minimum number of keys to type. Figure-3 illustrates a similar keyboard. They did not come with function keys or arrow keys. It was very difficult to navigate the screen without the arrow keys. The programmers of vi and its clones created navigation functionality, shortcuts using combination of various characters. There was a need to separate the navigation functionality from typing, for example keys like h/j/k/l can work like arrow keys during navigation, in command mode but the same keys will type respective characters in insert mode.

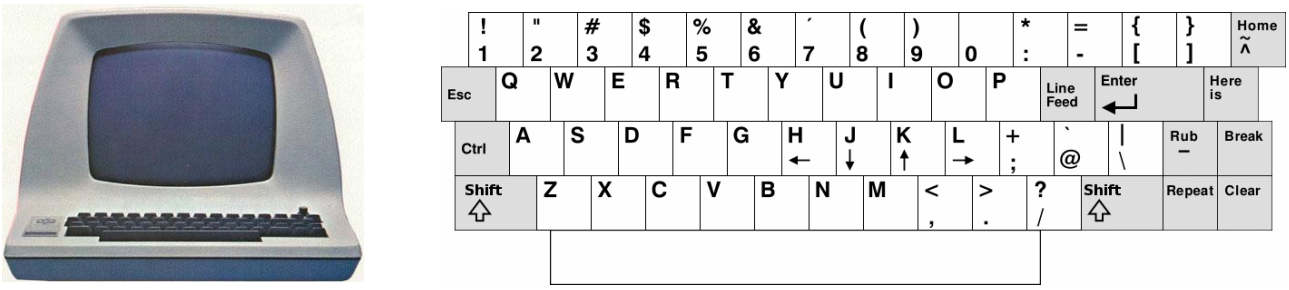

Figure 3: ADM3 Machine and its keyboard

Vim editor comes with different modes. These modes are the command mode, the insert/input mode and the ex mode. The command mode is mostly used for navigation purpose or to perform certain editing functions like delete/copy/paste etc.; input mode is used to type the characters and ex mode is used to execute commands like saving a file, configuring the editor etc. Brief overview of each mode is discussed below followed by the discussion of switching between different modes.

Command Mode

The command mode is the default mode when the user invokes vim. By entering the following command on shell prompt a user can start the vim editor.

```
$vim sample.txt
```

Figure-4 illustrates a new created file by issuing the command vim sample.txt, by default the editor opens in command mode.

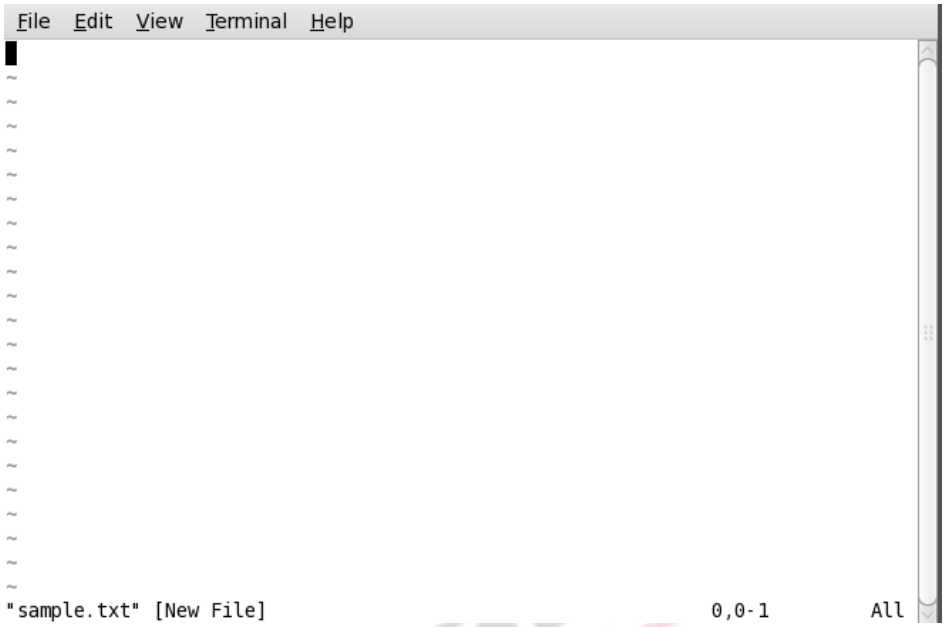

Figure 4: Screen of Vim – Creating a new file

In the command mode, the keys pressed by the user are not displayed on the screen and are interpreted as commands to be executed. Few commands like colon (:), question mark (?) or a slash (/) are displayed at the bottom most line of the screen. Most of the commands that are displayed at the bottom requires a return key (enter key) to be pressed, rest of the commands should not be followed by a return key. Command mode can be used to navigate across the screen, search and replace patterns, delete a character or a word, perform undo/redo actions, perform cut/copy/paste operations etc.

Input Mode

This mode is also known as insert mode or text mode. In the input mode, the keys pressed by the user are displayed on the screen. The keyboard works like a typewriter in the input mode. The characters that are pressed are inserted at the cursor position in the text. A user can know whether he has entered the input mode by examining the bottom line of the editor, the bottom line shows 'INSERT' when the user has switched to the input mode.

Ex mode

This mode is also known as Last Line mode. It is generally used to perform operations like saving a file, executing shell commands, perform substitution or to quit the editor. A user can type colon (:) to enter into this mode, ex mode requires return key to be pressed after entering a command.

Switching between the Modes

The vim editor enters into the command mode by default when it is invoked from the shell. To perform specific operations, a user has to switch between different modes. Figure-5 illustrates various ways to switch from one mode to another. It requires considerable effort and practice to get mastery over these modes.

By pressing 'i', user can switch from command mode to input mode. Alternatively a user may press either o, O, r, R, i, I, s, S, a, A or C to enter into input mode. Each character has a separate significance while switching from command mode to input mode, discussed in later sections. A user can switch from input mode to command mode by pressing Esc key. By pressing ZZ, a user can exit from command mode to Shell prompt. To switch from command mode to ex mode, a user must type colon (:) key. To quit from vim, user can type :x (exit after saving the contents) or :wq (write/save and quit) or :q! (exit with discard changes) in the ex mode. By pressing return key (Enter), a user can switch from ex mode to command mode.

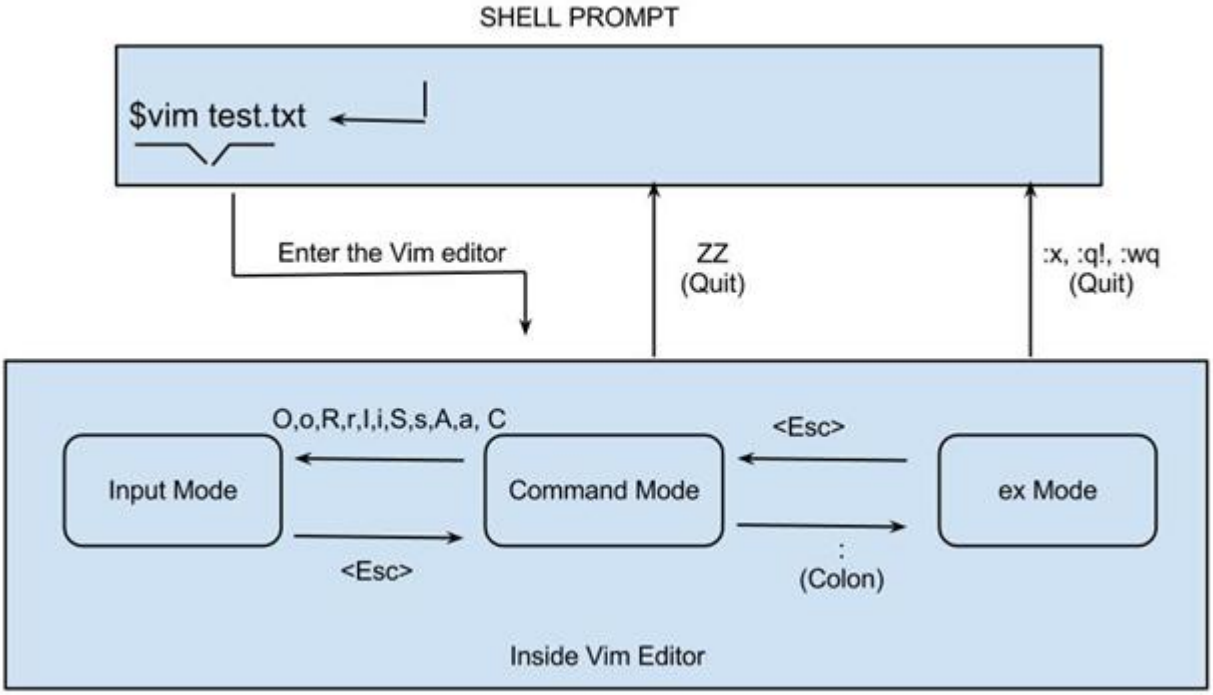

Figure 5: Switching between different modes

3.2 Entering and Replacing Text (Input Mode)

As discussed earlier, by default a user enters into command mode as soon as he invokes the vim editor. To type some characters, a user has to be in Input mode. Either of the characters from o, O, r, R, i, I, s, S, a, A or C can be used to switch from command mode to input mode. Generally the programmers prefer

typing 'i'. The bottom most line of vim will display the 'INSERT' text to signify that the user is now in input mode. The significance of each of the characters from oOrRiIsSaAC is shown in the below table

Command	**Effects in Mode Switching from Command mode to Input Mode**
a	Append, contents being typed will be inserted to the right of current cursor position
A	Append, contents being typed will be appended at the end of current line
C	Delete the contents after cursor position upto the line end and start appending
i	Insert, contents being typed will be inserted to the left of current cursor position
I	Insert, contents being typed will be inserted at the beginning of current line
o	Open a blank line below the current line, contents being typed goes into new line
O	Open a blank line above the current line, contents being typed goes into new line
r	Replace, replace a single character from current cursor position with another typed in
R	Replace, replaces all the text on the right of the cursor position
s	Replace, substitutes one character under the cursor position, we can also specify the count of characters to substitute
S	Replace, replaces the entire line without considering the current cursor position

Note: It is easy to remember the sequence of above characters by remembering orisa – the state of India

3.3 Exiting from the Vim (ex mode)

While a user is typing in the editor whatever typing is being done, the contents are not directly written to the disk but are stored in a buffer. Vim editor makes changes to the buffer as the contents are being typed. An accidental termination of the editor will generate a .swp (swap) file that contains the buffer. This avoids loss of data that was already typed. It is advisable for the user to save the work at regular intervals to avoid unnecessary hassle of .swp file recovery. The user may face challenges using vim in a networked terminal due to

disconnection of networks. In case of accidental termination of vim, whenever the file is opened again it will ask the user to recover the contents from .swp file to the original filename. The -r option helps in recovering a swap file.

The following section discusses the ways and means to quit the vi editor. It provides various options to save the work, save and quit or discard the changes and quit.

Most of the commands are executed in the ex (last line) mode. By pressing ':' (colon) key, the user enters into the ex mode. The bottom most line of the editor indicates the number of lines and characters present in the file along with the filename. For example the line may contain "sample.txt" 67L, 323C that indicates the filename is sample.txt which contains 67 lines and 323 characters. The bottom line may also indicate whether the file is read only or the contents are written to the disk from buffer.

```
:w [Enter]

"sample.txt" 67L, 323C written
```

By typing ':' (colon) the cursor moves down to the bottom most line of the editor, user has to type w followed by pressing Enter key, the editor will display the summary of file as shown above. If the user enters :wq, it implies write and quit from vim editor. By quitting the vim editor the user will get a $ prompt. The following table summarizes the commands to quit the editor.

Command	**Functionality**
(Quit from Ex mode)	
:w	Save changes to the file and remain in editing mode
:x	Save changes to the file and quit the editor to get a $ prompt
:q	Quit the editor when no changes are made
:q!	Quit the editor and discard the changes made to the file (Quit without saving)
:wq	Save changes to the file and quit the editor to get a $ prompt
:w newfile.txt	Save the changes to a file named newfile.txt (Similar to Save As in MS Windows)
:w! newfile.txt	Similar to above, but overwrites the changes to newfile.txt
:n1,n2w new.txt	Write from line n1 to n2 into a file new.txt

:w >> new.txt	Append the contents to a file new.txt (The file new.txt must exist)
:sh	Go to the shell prompt temporarily (By typing exit at the $ prompt will return to the vim editor)
(Quit from Command Mode)	
ZZ	Save changes to the file and quit the editor to get a $ prompt

3.4 Navigating the Screen (Command Mode)

In earlier days, the keyboards were too small without arrow keys or function keys, devices like mouse were not invented. The keyboard characters were used to navigate in the editor. Although modern day keyboards do allow navigation using arrow keys, it is quite exciting in using the traditional ways of navigation. Navigation can be across the words, lines or screens. The following section discusses navigation in the text documents.

Use of **repeat factors** enhances the capabilities of navigation and editing features. A repeating factor is an integer number prefixed to the command. By using a repeat factor, we can perform a particular command execution those many number of times. For example, command 'h' moves the cursor by one position to the left, using repeat factor we can move 5 positions to the left by pressing '5h'.

The navigation commands are generally used by programmers to jump to a particular line or word. It can help to debug programs easily and efficiently. The following table summarizes the navigation commands for vim.

Command	**Effects in Navigation**
Moving by one Position	
h or Backspace	Moves cursor to the left position by one character
l or Space	Moves cursor to the right position by one character
k	Moves cursor up by one line
j	Moves cursor down by one line
+ or Enter	Moves cursor down by one line and places at the beginning of the line
-	Moves cursor up by one line and places at the beginning of the line

Moving by words	
w	Moves cursor forward to the first character of next word
e	Moves cursor forward to the last character of the current word
b	Moves cursor backward to the first character of current word or previous word
Moving to end of Lines	
$	Moves the cursor to the end of the current line
0 (zero) or |	Moves the cursor to the beginning of the current line
^ (caret)	Moves the cursor to the first non-blank character, at the beginning of current line

Figure-6 illustrates the cursor movements within a line, the $ key in command mode moves the cursor to the end of line while by pressing the 0 (zero) or | moves the cursor to the beginning of the line. It also illustrates the cursor movements to the start or end of the current word using keys b and e respectively.

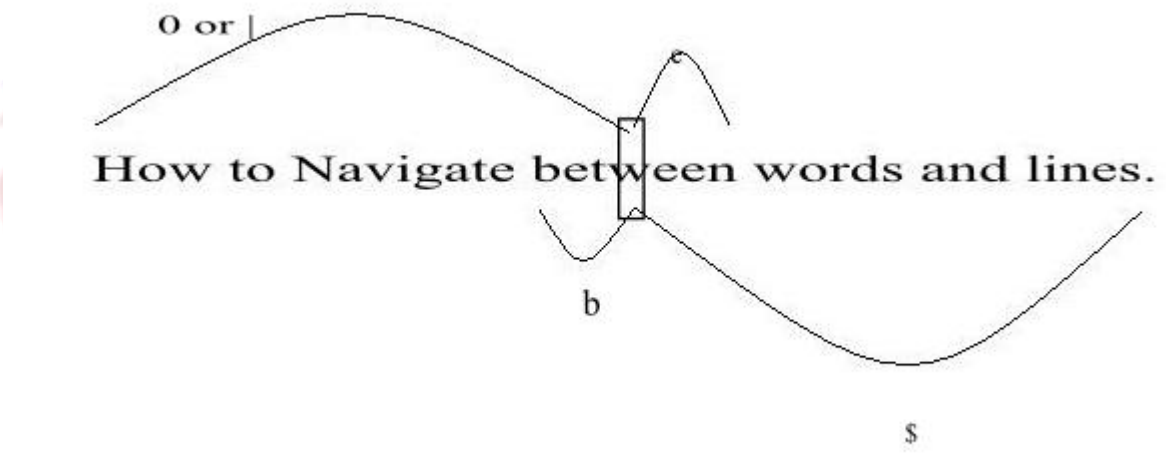

Figure 6: Cursor movement within a line

Moving to Top, Middle or Bottom of the screen	
H (High)	Moves the cursor to the topmost line of the screen
M (Middle)	Moves the cursor to the middle line of the screen
L (Low)	Moves the cursor to the bottom most line of the screen
Moving across Screens	

Ctrl + f	Scroll the screen forward
Ctrl + b	Scroll the screen backward
Ctrl + u	Scroll the screen by half, upwards
Ctrl + d	Scroll the screen by half, downwards
Go to a line	
G	Moves the cursor to the beginning of the last line, end of the file
1G	Moves the cursor to the beginning of the first line
5G	Moves the cursor to the beginning of the fifth line
$G	Moves the cursor to the last line of the file

A programmer will find G as a very useful command, it allows him to directly jump to a specific line of the source code while debugging the programs. Suppose the gcc compiler issues error message on line 275 then the programmer can open the file in vim editor and type 275G that allows him to directly move to the 275th line. The alternate ways to jump to a specific line from $ prompt can be

```
$vim +275 <filename>

$vim+/exit <filename>
```

In the above example, the first case allows to open a file and position the cursor in the beginning of the 275th line, while the second case allows positioning the cursor at the first occurrence of the "exit" keyword found in the file to be opened. The above techniques are very useful in debugging of the source code.

3.5 More Editing with Vim

Deleting

Deletion of single character, multiple characters, words or lines is possible in command mode. Use of repeat factors will give additional capabilities to the delete commands. Deletion of text can be done using 'x' or 'X' keys. These keys can be used to delete single characters, moreover operator 'd' performs deletions of words and lines. Character 'J' in command mode is used to join lines. The following table gives a summary of commands used for deletion and explains few repeat factors

Command	Effects in Deletion
Delete operations in Command Mode	
x	Deletes a single character at cursor position
X	Deletes a single character previous to cursor
4x	Deletes four characters; 4 is a repeat factor
dd	Deletes a single line
4dd	Deletes four lines from cursor position
dw	Deletes a word
4dw	Deletes four words from cursor position
d$	Deletes from cursor to end of line
d5G	Deletes lines from cursor position to line number 5
J	Joins the current line with next line
Delete operations in ex Mode	
:d	Deletes current line
:1,5d	Deletes lines numbered from 1 to 5
:$d	Deletes the last line

Copy Paste Operations (Yanking and Putting)

In vim, yanking is referred to copying of the text, such operations are performed in command mode. Single character, multiple characters, words, sentences or part of sentences can be copied using y or Y and repeat factors. Yanking temporarily stores the text in buffers. Vim editor offers 36 buffers. (26 named buffers from a to z, 9 numbered buffers from 1 to 9 and an unnamed buffer) By default, the copied or deleted text gets stored in the unnamed buffer. Users are given the flexibility to save contents into multiple buffers. The following table summarizes the yank operations and paste operations with repeat factors

Command	Usage
y	Copy a single character of current cursor position
4y	Copies 4 characters the current cursor position
yw	Copies a word
yy or Y	Copies entire line of the cursor position
4yy or 4Y	Copies four lines including the current line
y$	Copies from cursor position to the end of line

p (Lowercase)	Puts the copied or deleted text (contents of buffer) after or below the cursor position
P (Uppercase)	Puts the copied or deleted text (contents of buffer) before or above the cursor position
4p	Puts the contents of buffer four times.

The deleted text can be pasted using p or P commands. It is similar to cut-paste in windows. It must be noted that the deleted contents are also stored in buffer so that they can be pasted anywhere in the editor. Operations like copy-paste and cut-paste do support multiple files, so the contents of one file can be copied to another file. If the user wants to copy multiple blocks from one file to another file, he may use named buffers. For instance, the following steps can be performed to copy 2 blocks of 2 lines and 4 lines each from one file to another file using command mode

a) In the first file place the cursor position at appropriate position and issue the command "a2yy

b) Go to another desired block and issue the command "b4yy

c) In the second file issue the command "ap

d) In the second file issue the command "bp

The above steps copy 2 lines in buffer named 'a' and 4 lines in buffer named 'b'. The contents of these two buffers can be placed at the appropriate positions in another file using p command preceded by the name of the buffer. It should be noted that "(double quotes) are a part of the command.

Note: Vim provides the facility to edit multiple files at any instance, it is advisable to learn the to avoid the confusion in learning vim editor authors would recommend the readers to explore the feature by themselves.

Changing the Case

The ~ (tilde) command can be used to toggle the case. Repeat factors can be used for multiple characters. For example when 10~ is issued in command mode, it will change the case of 10 characters following the cursor position, moreover the cursor moves ahead by 10 position. This command is applicable only to alphabets. To convert a string into upper case, the command is "gU" and some movement whereas to convert to lower case, the command is "gu" and some movement.

Suppose the cursor is at the first character of the word "hello"

- Toggle case "Hello" to "hello" with 'g~'then a movement.
- Uppercase "hello" to "Hello" with 'gU' then a movement.
- Lowercase "Hello" to "hello" with 'gu' then a movement.
- For entire word, use 'g~w' or 'guw' or 'gUw'
- For entire line, use 'guu' to convert into lower case while 'gUU' to convert into upper case

Undoing and Repeating

The effects of previously executed command can be undone by using u (lowercase) in command mode. Vim does allow repeat factors with u command. Another command U (uppercase) allows undoing in the current line. If a user issues 4u, it undoes the previously executed 4 commands.

The '.' (dot) command repeats the previous operation. It can repeat the operations performed in command mode or input mode. For example if a user wants to type printf("Hello") 5 times, he can simply type it once and issue "4." in the command mode, the .(dot) command will repeat the contents 4 times. Similarly 5dd can be performed by a single dd command and "4." command.

3.6 Searching & Replacing

Most of the Linux administrators use search and replace facilities to configure various services in a network. Servers use large configuration files, often there are requirements to modify certain configurations in such files, the searching and substituting facilities offered by Vim relieves the Linux professionals from performing tedious tasks. Vim facilitates forward and backward searching, it also allows substitution (replacement) of words from selected text or from the entire file. The following section discusses few techniques widely used by the Linux professionals.

The strings that are to be searched are known as patterns. There can be many variations while searching for a particular pattern. The search process in vim is case-sensitive by default. However vim provides facilities to ignore case in search process. Vim uses special characters like '/' or '?' to perform forward and backward search in the normal mode. The last line of the editor is used while searching for a particular pattern.

Note: By default the search in Linux is case-sensitive. If a user wants to ignore case in pattern matching, the following command is to be issued in ex mode

```
:set ignorecase   or   :set ic

The reverse of above command is

:set noignorecase   or  :set noic
```

Command	Function
/searchstring	Searches in the forward direction for the next occurrence of the

	pattern. The cursor gets positioned under the first character of the pattern in the editor
?searchstring	Searches in the backward direction for the next occurrence of the pattern. The cursor gets positioned under the first character of the pattern in the editor
n	Repeats the last search command
N	Repeats the last search command in opposite direction of previous search

Few variations can be made while searching for a pattern or a sub-string. A user may either wish to search lines starting or ending with a particular pattern. User may be willing to search words starting or ending with a given pattern or may wish to search for words containing sub-strings of a given pattern. The following table illustrates few ways and means to search for a given pattern, it uses regular expressions. More about regular expressions is discussed in the separate module.

Pattern	**Meaning**
/^pattern	Search for the lines beginning with the pattern
/pattern$	Search for the lines ending with the pattern
/ \<pattern	Search for all strings beginning with the pattern
/pattern\>	Search for all strings ending with the pattern
/ \<pattern\>	Search for whole word (pattern); substrings containing pattern will be discarded
/[a-h]pattern	Search for patterns starting with a,b,c,...h
/ [^a-h]pattern	Search for patterns that does not start with either a,b,c,...h
/pattern*	Search for patterns ending with any characters

Substitution is a very powerful feature of vim editor. Apart from searching patterns the editor allows replacement; replacements can be made within the entire editor or to a particular block. Imagine a programmer who needs to modify the variable name for a given program; the programmer has the liberty to specify the line numbers in which modifications are to be made. Most of the word processor do not permit such flexibility in search and replace features.

The search and replace functionality is widely known as substitute feature. Vim follows a particular syntax to substitute a given pattern with a new one. Command 's' is used for substitution. In general the syntax is:

```
address/old_pattern/new_pattern/flag
```

Where address represents the starting or ending lines in the editor, old pattern indicates what is to be replaced by the new pattern; flag indicates whether to perform replacement from the entire range and also allows the user to confirm before replacements are done.

Command	**Function**
:s/old/new	Substitutes the first occurrence of old string with new string
:s/old/new/g	Substitutes all occurrences of old string with new string
:m,n s/old/new/g	Substitutes all occurrences of old string with new string from lines m to n
:1,$ s/old/new/g	Substitutes all occurrences of old string with new string from the first line to end of file
:1,. s/old/new/g	Substitutes all occurrences of old string with new string from the first line to the current line
:.,$ s/old/new/g	Substitutes all occurrences of old string with new string from the current line to end of file
:/old/new/gc	User selectively substitutes all occurrences of old string with new string with the confirmation from the user (vim waits for yes (y) or no (n) from the user and performs substitution accordingly)

Vim provides rich set of features for substitution; the readers may explore more features by referring the manuals or online documentation. For example if a programmer wants to comment certain part of code, say from line number 10 to 20, he need not comment each line rather issue a single command as shown below:

```
:10,20 s/^/#/g
```

Features like search and substitute allows a Linux professional to configure various files rapidly. For instance, configuring 50 machines in a computer lab hardly requires 2-3 minutes if the user is well versed! Although similar tasks can be achieved using word processor, more effort is required to achieve the functionality. It will be a matter of joy in exploring and using new features of Vim editor. Most of the GNU/Linux distros provide tutorials on vi. To go through the tutorials, type the following command:

```
$vimtutor
```

4 Overview of Nano Editor

GNU Nano is a simple easy-to-use text editor. It can be compared with the Microsoft edit program found in MS-DOS prompt. Nano editor aims to replace Pico editor that used to be the default editor included in the Pine e-mail program's message composer. Nano editor comes with some additional features like "search and replace" and "go to line and column number" that were not found in Pico.

Syntax:

```
nano [OPTIONS] [+LINE,COLUMN] [FILE] ...
```

Example:

```
$nano sample.txt
```

The above command will open a blank screen to create a file sample.txt. Navigating the screen can be accomplished by arrow keys or by keyboard shortcuts provided by the editor. The list of keyboard shortcuts can be viewed from the Help option by using Ctrl + g shortcut. Some of the basic options are summarized in the following table

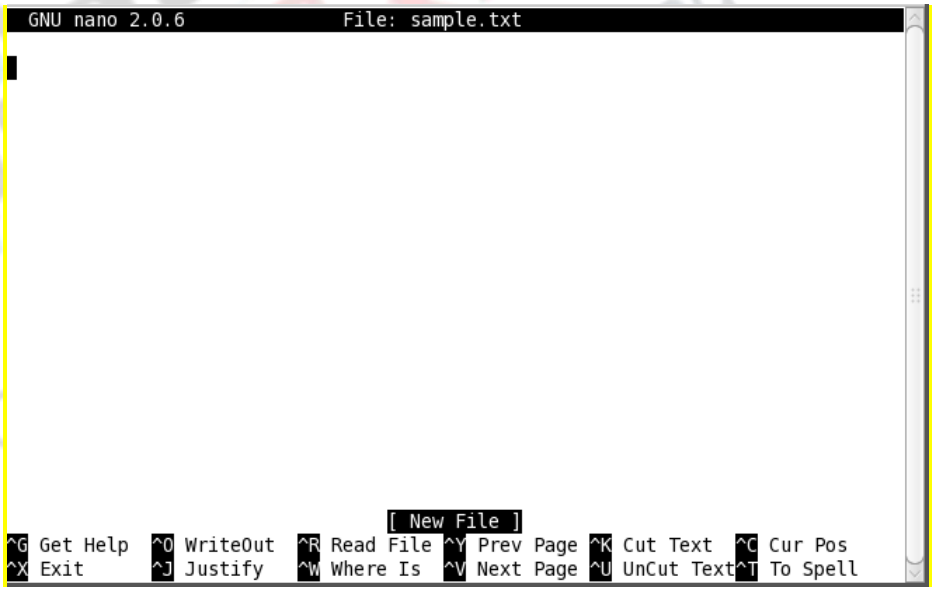

Figure 7: Default Screen of Nano editor

Options	**Particulars**
-m	Enable mouse support

-v	View the file in read-only format

Note: Students must avoid the use of Nano editor unless they have mastered vim editor. It is important to note that not all Linux systems come with Nano editor. It is advisable that a student must know vi editor.

5 Overview of Emacs Editor

Vim gained lot of popularity but there is a community who prefers Emacs editor. This editor is licensed under GNU GPL. The two most famous editors in Linux/Unix users are vi and Emacs. Emacs is mostly used by the programmers working on LISP. GNU Emacs is a free, portable, extensible text editor, portable means that it runs on many machines under many different operating systems and extensible means that it can be customized for fonts, colors, key bindings, mouse, menus etc. Emacs is an extremely successful editor particularly good for programmers. This editor makes it easy to edit code in almost all languages, providing context sensitive indentation and layout. It also allows to compile programs inside emacs, with links from error messages to source code; facilitates debugging, manage change logs; jump directly to a location in the source by symbol (function or variable name); and interact with the revision control system. Emacs also provides mail readers, news readers, World Wide Web, gopher, and FTP clients, spell checkers.

Emacs in general is the name of a family of text editors that are either descended from or inspired by one another. The first Emacs for Unix machines was Gosling Emacs and later got commercialized into Unipress Emacs. GNU Emacs was written by Richard Stallman and got released in 1976. GNU Emacs is the most popular flavor of Emacs editor and is included in most of the Linux Distributions. Figure-8 is a welcome screen of the editor. Emacs editor can be invoked using the following command from Shell prompt:

```
$emacs <filename>


$emacs Fundamental.txt
```

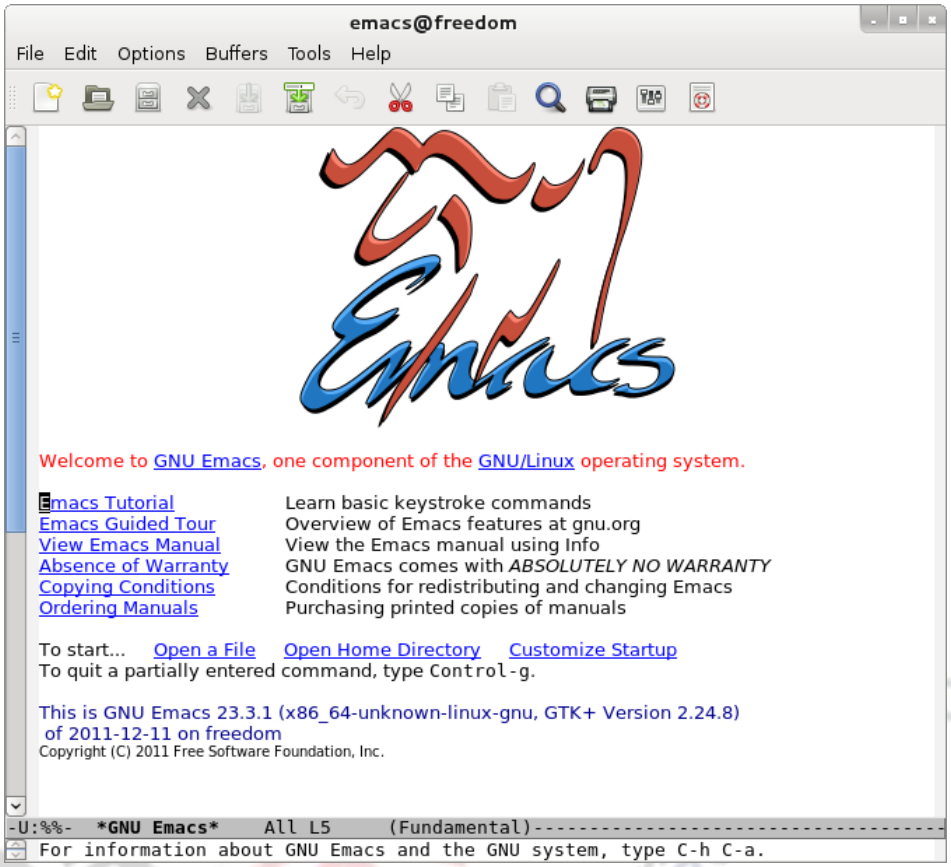

Figure 8: Welcome Screen of GNU Emacs editor

GNU Emacs is not a WYSIWYG word processor. Unlike Vi, Emacs is a "mode-less" editor. It does not require to change modes while performing various text editing functionality, Emacs is always in the Input mode. As there are no modes in emacs, all the functions of emacs are performed using combination of Control keys and Meta keys. Meta keys are usually either Alt or Esc key of the keyboard depending the type of machine.

Emacs commands are denoted as combinations of C-x and M-x which indicates that the Control and x keys should be held down together, the x character can be replaced by any other key, similarly M-x indicates that the Meta and x keys should be held down together. Emacs commands are abbreviated with C or M for Control and Meta Keys respectively. For example C-f is used to move one character forward, M-f moves forward by one word. For example, to exit Emacs, the command is C-x C-c (save-buffers and quit-emacs). It will offer to save all your buffers and then exit. Emacs documentation comes with hundreds of different combination to perform text editing functions. A comprehensive list of GNU/Emacs commands is given in extra reading document. It is advisable to get mastery over either Vi editor or Emacs editor.

Few other editors like sed and document processing system like latex are discussed in separate modules.

Summary

- Text editors (rather than word processing programs like MS Word or Writer) are used quite often in Linux. They are used for tasks such as for creating or updating system configuration files, writing scripts, developing source code, etc.
- nano is an easy-to-use text-based screen editor.
- gedit, kwrite, kate etc are graphical editors very similar to Notepad inWindows.
- The vim editor is available on all Linux systems and is very widely used. Graphical extension versions of vim (gvim) are widely available as well.
- vim has three modes: Command, Insert, and Line; emacs has only one but requires use of special keys such as Control and Escape.
- GNU emacs is available on all Linux systems as a popular alternative to vi. It can support both a graphical user interface and a text mode interface.
- To access the vi tutorial, type vimtutor at a command line window.
- To access the emacs tutorial type Ctlr-h and then t from within emacs.
