

Chapter 3: The Unix File System

3.1. The File System

- oWhether it is a C program you execute, or a directory you use to house groups of such programs, they are all files in UNIX sense.
- oThis concept also extends to hard disk, printer, CD-ROM drive or terminal.
- oThe Shell is also a file, and so is the kernel.
- oThe main memory in your system is a file too.

- oA file is divided into three categories:
 - Ordinary file** - also known as **regular file**. It contains only data as a stream of characters.
 - Directory file** - A folder containing the names of other files and directories
 - Device file** - It represents all hardware devices

3.1.1 Ordinary File

- oThe traditional file is the **ordinary** or **regular** type. It consists of a stream of data resident on some kind of media.

- oYou can put anything you want into this type of file, including all data, source programs, object and executable code, all UNIX commands, as well as file created by the user.

- oCommands like **cat**, **ls**, and so forth are treated as ordinary or regular files.

- oThe most common type of ordinary file is the **text file**. This is just a regular file containing printable characters. The programs that you write are text files

.

- oThe UNIX commands that you use, or the C programs that you execute, are not.

- oThe characteristic feature of text files is that the data inside them are divided into groups of lines, with each line terminated by the *linefeed* (LF) character.
- oOn UNIX systems, the linefeed character is also known as the newline character.
- oIt is generated by the system when you press the *[Enter]* key and is not visible and doesn't appear in hard copy output.

3.1.2 Directory File

- oA **directory file** contains no external data but maintains some details of the files and subdirectories that it contains.
- oThe UNIX file system is organized with a number of such directories and subdirectories, and you can create them as you need.
- oYou often need to group a set of files pertaining to

- o This allows two or more files in separate directories to have the same filename.
- o A directory file contains two fields for each file—its name and identification number—Every file has a number called the **inode number**.
- o If a directory has, say, ten files, there will be ten such entries in the directory file.
- o You cannot write directly into a directory file; only the kernel can do that.
- o When an ordinary file is created or removed, its corresponding directory file is automatically updated by the kernel with the relevant information about the file.

3.1.3 Device File

- oThe definition of a file has been broadened by UNIX to consider even physical devices.
- oThis definition includes printers, tapes, floppy drives, CD-ROMS, hard-disks and terminals.
- oThe device file is special; it doesn't contain any data whatsoever—This file is not a sequence of characters.
- oAny output directed to it will be reflected onto the respective physical device associated with the filename.
- oWhen you issue a command to print a file, you are really directing the file's output to the file associated with the printer.
- oWhen you back up files on a disk, you are using the file associated with the disk drive.
- oThe kernel takes care of this by mapping these special files to their respective devices.

3.1.4 What's in a (File) name?

- o On most UNIX systems today, a filename can consist of up to 255 characters.

- o If you enter more than 255 characters when naming a file, the system will send an error message.

- o Files may or may not have extensions and can consist of any ASCII character except the /.

- o Just as you can have a filename beginning with a dot, you can have one which ends with a dot too. All these are valid filenames:

- index.last_time LIST.

- o You should mainly use the following characters when framing filenames:

- Alphabets and numerals

- The period(.)
- The hyphen (-)
- The underscore(_)

oUNIX imposes no restrictions on the extension, if any, that a file should have.

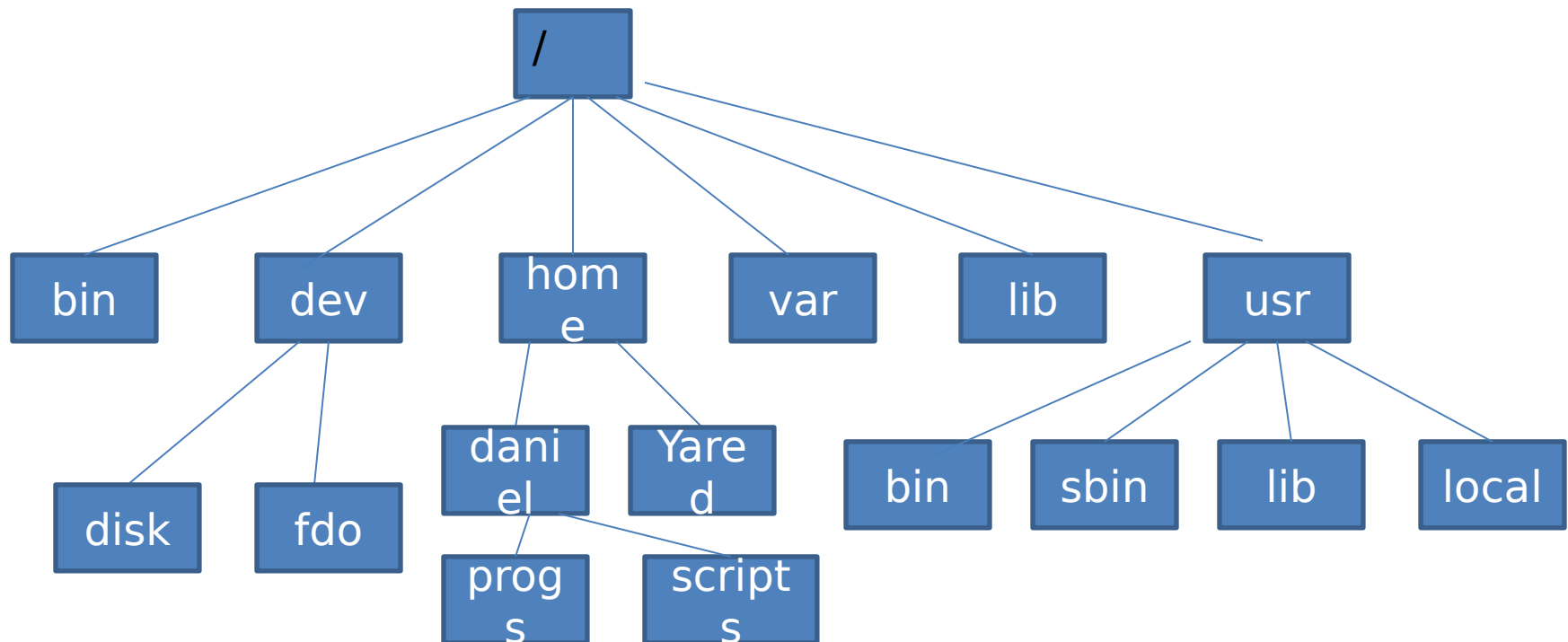
oNever use a - at the beginning of a filename.

When you use the file with a command it might be confused for an option.

3.1.5 The Parent-Child Relationship

oAll files in UNIX are “related” to one another. The file system in UNIX is a collection of all these related files organized in a hierarchical structure.

- oIn every UNIX file system, there is a supremo, which serves as the reference point for all files. This top is called **root**, and is represented by /(frontslash).
- oRoot is actually a directory file, and it has all the subdirectories of the system under it.
- oThese subdirectories, in turn, have more subdirectories and other files under them.



- oFor instance, bin and usr are two directories directly under root, while a second bin and lib are subdirectories under usr.
- oEvery file, apart from root, must have a parent.
- oIf the ancestry of a file can not be traced to root, the file is simply not part of the file system.
- oThus, the home directory is the parent of daniel, while root is the parent of home and the grandparent of daniel.
- oIf you create a file transport.cpp under the daniel directory, then daniel will be the parent of this file.

3.1.6 The UNIX file System

The following is a heavily trimmed structure of the standard UNIX file system.

o/bin and usr/bin—These are the directories where all the commonly used UNIX commands (binaries, hence the name bin) are found. Note that the PATH variable always shows these directories in its list.

o/sbin and /usr/sbin –These are the directories that store commands only the system administrator can execute

o/etc—This directory contains the configuration file of the system.

o/dev– This directory contains all device files.

o/home—All users are housed here. Daniel, for example, would have his home directory as /home/Daniel.

o/tmp—The directories where users are allowed to create temporary files. These files are wiped away regularly by the system.

o/var—The variable part of the file system. Contains all your print jobs, mail queues and incoming mail

o/lib –contains all library files.

3.1.7 The Home Directory

oWhen you log on to the system, UNIX automatically places you in a directory called the **home directory** (or **login directory**).

oIt is created by the system when a user account is opened. If you login using a login name daniel, you will probably end up in a directory having the pathname /home/daniel.

oThe shell variable HOME knows your home directory. To evaluate it, issue this command from the shell prompt:

```
$ echo $HOME  
/home/daniel
```

oThis home directory is decided by the system

3.1.8 Absolute Pathnames

o Many UNIX commands use file and directory names as arguments. These files and directories are presumed to exist in the current directory. For instance, the command

o `cat file1`

o Will work only if the file `file1` exists in your current directory. However, if you are placed in `/var` or another directory like `/ps` and you want to access `file1` in `/home/daniel`, you can't obviously use the above command. There are two ways of accessing this file from your current directory

o With an **absolute pathname** which uses the root directory as the ultimate reference for the file. All path references originate from the root.

o With a **relative pathname** which uses the current directory as point of reference and specifies the path relative to it.

oUsing an absolute pathname for “file1” means that you have to use this command line from your current directory to display the file.

```
cat /home/daniel/file1
```

o**cat** here uses an absolute pathname, where the location of file1 is specified with reference to root(the first /). When you have more than one / in a pathname, for each /, you have to descend one level in the file system.

oThus Daniel is one level below home, and two level below root.

3.1.9 cd: Changing Directories

oTo change to the bin directory from home directory, we issue the following command using **cd**.

```
$ pwd
```

```
/home/daniel
```

```
$ cd /bin
```

```
$ pwd
```

```
/bin
```

If we just type **cd**, it will take us to the home directory

3.1.10 Relative Pathnames(. and ..)

To move to the parent directory, one level up, you use **cd .. (two dots)**. Now, try this out:

```
$ pwd
```

```
/home/daniel
```

```
$ cd ..
```

```
$ pwd
```

```
/home
```

3.1.11 mkdir and rmdir: Making and removing directories and files

oAs we have seen earlier, directories are created with the **mkdir** command.

oWe can also create a directory tree with the following command.

```
$ mkdir ps ps/progs ps/data
```

oTo remove a directory, we use **rmdir** command

```
$ rmdir ps
```

```
rmdir: ps: Directory not empty
```

oTo remove the directory including its contents, use **rm -r *** from its parent directory.

orm command can also be used to remove files

```
rm chap01 chap02 chap03
```

oCan be used to delete more than one file at once.

oTo remove all files in the current directory, we can use:

```
$ rm *
```

```
$ _
```

3.1.12 cp: Copying files

oTo copy a file, note1, to a file, chap1, use the command:

```
$ cp note1 chap1
```

oThe first file is copied into the second. If the destination file doesn't exist, it will be created before copying takes place.

oWhat happens if chap1 exists and is a directory instead of an ordinary file?

oIn this case, note1 is copied into that directory(chap1) with the same name. It is like using

```
cp note1 chap1/note1
```

oTo copy files note1, note2 to the ps directory, you

oInteractive Copying (-i): The -I interactive option warns the user before overwriting the file. If unit1 exists, **cp** prompts for a response:

```
$ cp -i chap1 unit1
```

```
Cp: overwrite unit1? Y
```

oA y at this prompt overwrites the file: any other response leaves it uncopied.

oIt is also possible to copy an entire directory structure with -r (recursive) option. The following command copies all files and subdirectories in progs to newprogs:

```
cp -r progs newprogs
```

3.1.13 mv: Renaming files

To rename a file, note1 to chap1, use the following syntax:

```
$ mv note1 chap1
```

To move files, we also use the **mv** command. The following command moves two files to the progs directory

```
mv note1 note2 progs
```

Here progs must be a directory.

3.1.14 Creating Files

cat is also useful for creating a file:

```
$ cat > foo
```

A >symbol following the command means that the

Output goes to the filename following it. Cat used this

Now you can use the following command to display the contents of the file you just created:

\$ cat foo

3.1.15 file: know the File types

To know the type of the a single file,

\$ file note1

Note1: ASCII English text

To know the type of all files in a directory,

\$ file *

3.2 The VI Text Editor

oIn Unix, you will eventually write some C or Java programs or shell (perl) scripts.

oYou may also have to edit some of the system files at times.

oFor all this you must know how to use an editor and UNIX provides two very versatile ones- **vi** and **emacs**.

oWe will look at **vi**.

o**vi** is a full screen editor now available with all UNIX systems, and is widely acknowledged as one of the most powerful editors available in any environment. oIt was created by a graduate student-Bill Joy—later to become the cofounder of Sun Microsystems. **vi** made its first appearance in BSD UNIX but is now standard on all UNIX systems.

3.2.1 vi Preliminaries

oA **vi** session begins by invoking the command **vi** with a filename:

ovi index.html

oYou are presented a full empty screen, each line beginning with ~(tilde).

oThis is vi's way of indicating that they are nonexistent lines.

oFor text editing vi uses 24 of the 25 lines that are normally available on the terminal.

oThe last line is used for some commands that you will enter to act on the text.

oThis line is also used by the system to display messages.

oThe filename appears in this line with the message "index.html"

3.2.1 The Three Modes

oWhen you open a file with **vi**, the cursor is positioned at the top left-hand corner of the screen. You are said to be in **Command Mode**.

oThis is the mode where you pass commands to act on text.

oPressing a key doesn't show it on screen, but may perform a function like moving the cursor to the next line, or deleting a line.

oYou can't use the Command Mode to enter or replace text.

oThere are two Command Mode functions that you should know at this stage:

oThe *[Spacebar]* takes you one character ahead, while *[Backspace]* or *[Ctrl-h]* takes you a character back.

oBackspacing in this mode does not delete text at

oTo enter text, you have to leave the Command Mode and enter the **Input Mode**.

oThere are 10 keys which take you to this mode when pressed and whatever you enter then shows on the screen.

oBackspacing in this mode, however, erases all characters that the cursor passes through.

oTo leave this mode you have to press the *[Esc]* key.

oYou will have to save your work, leave **vi** or switch to editing another file.

oNeither of the two modes will quite do the work for you.

oYou have to use the **Last Line Mode** or **ex Mode**, where you enter the instruction in the last line of the screen.

oYou enter this mode by pressing ***[Shift]***: if you are in

3.2.3 Quitting vi- The Last Line Mode

- oWhen you edit a file using **vi**, or for that matter any editor, the original file isn't disturbed as much.
- oThe editor works with a copy of the file which it places in a **buffer**.
- oThis is simply a temporary storage area that is associated with a file on disk.
- oFrom time to time, you should save your work by writing the buffer contents to the disk file.
- oIt is the disk file that is permanent, not the buffer

3.2.3.1 Saving and Quitting (:wq and :x)

- oThe Last Line Mode offers two ways of saving and quitting--**:x** and **:wq**.
- oEither command returns you to the shell after saving your work (i.e. after writing the changed buffer to disk)

:x [Enter]

\$ _

3.2.4 Aborting Editing (:q)

It is also possible to abort the editing process and quit the editing mode without saving the buffer.

The **q**(quit) command takes you out of the editor

only if you don't have a changed buffer:

:q[Enter]

\$ _

ovi also has a safety mechanism that prevents you from aborting accidentally if you have modified the file in any way.

oNow try the same command again on a modified but unsaved file:

:q*[Enter]*

No write since last change (use ! To override)

oYou haven't saved the changes and are trying to quit; that is what the message says.

oIf you still want to abandon the changes, then use

:q!*[Enter]*

oThis will always return you to the prompt irrespective of the status of the buffer-no questions asked.

3.2.5 Inserting and Replacing Text

3.2.5.1 Inserting Text(i and I)

- oThe simplest type of input is insertion of text.
- oWhether the file contains any text or not, when **vi** is invoked, the cursor is always positioned at the first character of the first line.
- oTo insert a text at this position, press
i
- oThe character doesn't show up on the screen, but pressing this key changes the mode from Command to Input.
- oYou will see the words – INSERT – .
- oIf you don't see these words the type following in order to see the mode:
:set showmode
- oFurther key depressions now will show text on the screen as it is being entered.

oStart inserting few lines of text and follow each line with *[Enter]*

vi editor*[Enter]*

It is slow in getting started but is quite powerful*[Enter]*

It operates in three modes*[Enter]*

All the features of ex are also available*[Enter]*

You can even escape to the UNIX shell*[Enter]*

It maintains 26 buffers for storing chunks of text

~
~
~
~
~
~
~
~
~

- oThe lines containing text along with the “empty line” (actually nonexistent lines, shown with a ~ against each) approximate the screen shown here.
- oThe cursor is now positioned in the last character of the last line.
- oThis is known as the **current line**.
- oThe character where the cursor is stationed is known as the **current cursor position**.
- oIf you notice a mistake on this line, you can use *[Backspace]* to erase any inserted text –one character at a time.
- oAfter you have entered a few lines, you should press *[Esc]* to take you back to the Command Mode.
- oSince your work has not been saved, save your work.

- o You started insertion with **i**, which put text at the left of the cursor position.
- o If the **i** command is invoked with the cursor positioned on existing text, text on its right will be shifted further without being overwritten.
- o While **i** can be used to insert text anywhere in a line, **I** inserts text only at the *beginning* of a line.
- o Even if you are placed at the end of a line, you can move to line beginning with **I**
- o And then key in your text.
- o This represents a quick way of navigating to the beginning of the current line too.

the vi **e**ditor

ifull-screen*[Space][Esc]*

the vi full-screen editor

IThis is*[Space][Esc]*

This is the vi full-screen editor

3.2.5.2 Appending Text (**a** and **A**)

o There are other methods of inputting text. To append text to the right of the cursor position, use

a

o Followed by the text you wish to key in. This is slightly different from **i** as it places text on the right of the cursor.

o After you have finished editing press *[Esc]*

o **a** too has an uppercase counterpart which acts in completely opposite manner compared to **I**.

o To append text at the end of a line use

A

It is **s** slow in getting started but is quite powerful

a[Space]**very**[Esc]

It is very slow in getting started but is quite powerful

A[Space] **and flexible**[Esc]

It is very slow in getting started but is quite powerful
and flexible

3.2.5.3 Opening a New Line

o You can also “open” (insert) a new line by positioning the cursor at any point in a line and pressing

o

- oThis inserts an empty line below the current line.

- oTo open a line above the current line, use

O

It operates in three modes^s

oTo Insert text, you have to be in Insert Mode[Esc]

It operates in three modes

To insert text, you have to be in Insert Mode^e

OThese are Command Mode, Insert Mode, and Last Line Mode[Esc]

It operates in three modes

These are Command Mode, Insert Mode, and Last Line Mode

To insert text, you have to be in Insert Mode

3.2.5.4 Replacing Text(**r**, **R**, **s** and **S**)

oText is replaced with **r**, **R**, **s** and **S**. To replace one single character with another, you should use

r

oFollowed by the character that replaced the one under the cursor. You can replace only a single character this way.

o**vi** momentarily switches from the Command Mode to the Input Mode when **r** is pressed.

oit returns to the Command Mode as soon as the replacing character is entered.

oThere is no need to press *[Esc]* after using **r** (and the character) since **vi** expects a single character anyway.

oTo replace more than a single character, use
R
o followed by the text.

This is the vi full-screen editor
rt

this is the vi full-screen editor
RHere[Esc]

Here **e** is the vi full-screen editor

- oMany times, you will need to replace a single character with multi-character text.

- oTake your cursor to the point and press

S

- oNow enter as much text as you want and press *[Esc]*

- oThe text existing on the right is shifted further.

- o**S** replaces the entire line irrespective of the cursor position.

- oYou don't need to take your cursor to the beginning of the line to change the line's contents. Stay where you are and press

S

- oThe entire line vanishes from sight. Key in your text, and then press *[Esc]*.

You can even escape **t** to the UNIX shell

sby[Esc]

You can even escape by **o** the UNIX shell

SYou can use vi for writing codes[Esc]

You can use vi for writing codes.

3.2.5.5 Entering Control Characters

ovi uses *[Ctrl-v]* to precede any control character.

oFor instance, to enter *[Ctrl-h]*, you have to first press *[Ctrl-v]* and then *[Ctrl-h]*.

oYou will then see this on the screen

^H

oEven though you feel you are seeing a ^ (caret) and an H, there is only a single character out there.

oYou can position the cursor only on the ^ and not on the H; that's how you identify control characters.

The same technique can be adopted for entering the *[Esc]* character. Press *[Ctrl-v][Esc]*, and you will see the *[Esc]* character looking like this:

^[

This too is a single character. You can only place your cursor only on the ^.

It maintains 26 buffers for storing chunks of text^t

A[Ctrl-v][Ctrl-g][Space][Esc]

It maintains 26 buffers for storing chunks of text^G

3.2.6 Saving Text (:w)

oTo save a file and remain in the editing mode, use the **w** (write) command:

:w

“index.html” [New], 10L, 312C written

oThe message shows the name of the file, along with the number of lines and characters saved.

oWith the **w** command, you can optionally specify a different filename as well:

:w anotherfile

“anotherfile” [New] 10 L, 312 C written

oIn this case, the contents are separately written to anotherfile. However, your current file continues to be index.html even though **vi** shows anotherfile in the last line.

3.2.6.1 Writing Selected Lines

oThe **w** command can be used with one or two **line addresses** (one or two numbers representing one or more lines).

oIf used with one address, **w** writes the specified line to the file.

oWhen used with two, it writes the group addressed by the line numbers:

:5w lines

oThe above command writes the 5th line to the file lines.

:2,4w lines2

oThe command writes lines 2 through 4 to lines2 file.

o**vi** has two symbols that denote the current line and last line of the file. The **.** Represents the current line, while **\$** denotes the last line.

3.2.7 Recovering from Crash

It is not always possible to save a file.

Power glitches and blackouts or a kernel panic could leave some work unsaved.

However, since **vi** works with a buffer on disk, you can often recover much of your work.

vi may inform you of a catastrophe.

In that case, quit the editor and restart it with the **-r**(recover) option:

vi -r foo

3.2.8 Exit to the UNIX Shell

Sometimes, you need to make a **shell escape**(temporary exit) which brings the shell prompt for you.

You can do some work at the prompt and then come back to **vi**.

oThe following Last Line Mode command provides this escape facility.

:sh

\$_

oPressing **exit** or [Ctrl-d] brings you back to **vi**.

oYou can also run a single UNIX command without having a shell prompt in front of you.

oUse **:! With command name.**

:!date[Enter]

oSun Sept 13 10:30:23 PDT 2015

oPress Enter or enter a command to continue

oAn *[Enter]* here returns you to the editing screen you were in.

3.2.9 The Repeat Factor

One of the most notable feature of vi is the facility of prefixing a number to a command.

When this is done, most commands interpret the instruction to be repeated as many times.

For instance **3s** replaces the next three characters under the cursor.

This number prefixing a command is called the **repeat factor**.

If you want to insert a series of 20 asterisks in one line:

`20i*[Esc]`

3.2.10 The Command Mode

This is the mode you come when you have finished entering or changing your text. To enter this mode from the Input Mode, you have to press `[Esc]` key.

- oThis mode is meant for performing navigation and cut and past operations. Keys here are used both singly and in combination.
- oUnlike the Input Mode, when you press a key in the Command Mode, it doesn't show up on the screen, but simply performs its function.
- oThat is why you can see changes on the screen without knowing the command that has caused them.

3.2.11 Navigation

vi, in addition to the arrow keys, has its own set of commands that move the cursor in the four directions —**h**,**l**,**j** and **k**.

h moves cursor left, **l** right, **k** up, and **j** down.

You can use **5l** (moves 5 characters right) or **10k** (Moves 10 lines up).

3.2.12 The Beginning or End of a Line(0, | and \$)

- oMoving to the beginning or end of a line is sometimes essential.
- oThis is handled by **0** (zero), **|** and **\$**.
- oTo move to the first character of a line, use **00** or **|**
- oAnd to move to the end of a line, use **0\$**

3.2.13 Moving by Words (b, e, w)

- oWord-oriented movement is achieved with **b**, **e**, **and w** commands. Their functions are shown below:
- ob-moves back to beginning of word.
- oe-moves forward to end of word.
- ow-moves forward to beginning of word.

- oA repeat factor speeds up cursor movement.
- oFor example, **5b** takes the cursor five words back, while **3w** takes the cursor three words forward.
- oA word here is simply a string of alphanumeric characters and the **_** (underscore) character.
- oBash is one word and so is rd_profile.
- oTCP-IP is three words: the hyphen by definition becomes a word too.
- oThe commands **B**, **E** and **W** perform similar functions to those of their lowercase counterpart except that they skip punctuation.

3.2.14 Go to Line(G)

oIf you want to know the line number of your current cursor position, simply press *[Ctrl-g]*. You could see a line like this:

“index.html” [Modified] line 403 of 541 -74%-

o You come to know the filename, the current line number, the total number of lines in the file and the relative location of the page expressed as a percentage.

o You can use the **G** command to move the cursor to a specific line number. This is prefixed by the line number.

o To move to the 6th line, use

6G

o And to move to the beginning of the file, use

1G

o The end of the file is reached by simply using

G

3.2.15 Deleting ,Moving and Copying Text

3.2.15.1 Deleting Characters

oThe simplest text deletion is achieved is with the `x` command.

oThis command deletes the character under the cursor.

oMove the cursor to the character that needs to be deleted and then press `x`

o`x`

oThe character is removed from sight, and the text on the right shifts left to fill up the space.

oA repeat factor also applies here, so **`4x`** deletes the current character as well as three characters from right.

oThe unique feature of this command is that when you press **`x`** at the end of a line, you don't pull up the following line as most Windows editors do.

oInstead, **`vi`** starts removing characters *on the left*

To delete a text on the left of the cursor use
X

Unlike **x**, **X** doesn't delete the character under the cursor.

Here is the vi full-screen editor

x

Here is the vi fullscreen editor

6x

Here is the vi full editor

5X

Here is the vi editor

3.2.15.2 Joining Lines (J)

To join the current line and the line following it, use

J

J removes the **newline** character between the two lines

You can use the vi for writing codes

It maintains 26 buffers for storing chunks of text

J

You can use the vi for writing codes It maintains 26 buffers for storing chunks of text

3.2.15.3 Fine Deletion-Using the d Operator

If you want to delete all text from the present cursor position to the end of line, you can use the **d** operator.

oUsing the **d** operator, you can frame the command yourself. Using the command that moves you to the end of the line, **\$**, you can delete all text from the current cursor position to the end of a line.

d\$

oThe same operation can be done with **D**.

oSimilarly, since **G** takes you to the end of the file, you should use the following combination to delete all text from the current cursor position to the end of the file:

dG

oLikewise, to delete a word,

dw

oYou can also use the repeat factor and use **5dw** to delete five words.

oEntire lines are removed with the **dd** command.

Move the cursor to any line, and then press

dd

oTo delete a block of lines, use the repeat factor.

oThus to delete the current line and five lines below it,

use **6dd**

It operates in **t**hree modes

These are Command Mode, Insert Mode, and Last Line Mode

To insert text you have to be in Insert mode

All the features of ex are also available

dd

These are Command Mode, Insert Mode, and Last Line Mode

To insert text you have to be in Insert Mode

All the features of ex are also available

2dd

All the features of ex are also available

P

These are Command Mode, Insert Mode, and Last Line Mode

To insert text you have to be in Insert Mode

All the features of ex are also available

3.2.16 Moving Text (d, p and P)

oText movement is an extension of the deletion operation.

oYou just have to put the deleted text at the new location.

o**p** is used for putting back entire lines(those deleted with **dd**) below the current line.

o**P** places them above.

oWords are put to the left with **p** and to the right with **P**.

Moving Text between Two Files:

oYou can perform cut and paste operations between two files using a modified form of the above technique.

oTo move a block of text from one file to another just follow these steps:

1. Delete the text into a buffer a. If you are deleting three lines then use **“a3dd”**, where the normal delete command is preceded by a string “a.
2. Save the current file with **:w**.
3. Open a new file using the Last Line Mode command **:e foo**.
4. To put back text, navigate to the desired location, press “, the buffer name a and p to place the text below the current line – **“ap**.

4.1.5 Copying Text-Using the y Operator

oWords and lines are copied (yanked – **vi** terminology) with the **y** operator.

oThis means **yy** copies a line.

oTo yank five lines of text, move the cursor to the first of these and press

5yy

oNext, move the cursor to the new location below which you want to insert these lines and press

p

oto put the copied text below the current line.

oAdopting the same logic, one can say that **y\$** yanks text from current position to end of line, and **y1G** copies text from current position to the beginning of the file.

Here is the vi full-screen editor

It is very slow in getting started but is quite powerful and flexible

yy

Here is the vi full-screen editor

It is very slow in getting started but is quite powerful and flexible

It operates in three modes

These are Command Mode, Insert Mode, and Last Line Mode

To insert text you have to be in Insert mode

p

.

.

To Insert text you have to be in Insert Mode

Here is the vi full-screen editor

Copying Text between Two Files:

- oUsing the technique similar to the one used in deleting text, you can copy text between two files.
- oTo copy four words from one file to another, just use **4yw** instead of **4dd**;
- oEverything else remains the same except that you don't need to save the original file.

3.2.17 Changing Text

3.2.17.1 Changing with the c operator

To change text from the present cursor position up to the end of a line, use

c\$ or **C**

To change entire lines, use

cc

To change words, use

cw

3.2.18 Changing Case

vi uses ~ command to toggle the case of text.

To reverse the case of a section of text, move the cursor to the first character of the section and then press

~

The case changes: upper become lower, and lower become upper.

When you keep it pressed, the cursor moves forward and reverses the case of all subsequent text that it encounters.

If you have to do this for a string of twenty characters, use a repeat factor – **20~**.

3.2.19 The Dot: Repeating the Last Command

If you want to insert the same text or delete the same number of lines at several places, you have to use the actual insertion or deletion command only

oTo repeat the command at other places you have to use a simple `.(dot)` command.

oIf you copy a line of text with **yy** command, then to repeat the copying elsewhere, all you have to do is position the cursor at the desired location, and press `.`

o**3.2.20 Undoing Last Editing Instruction (u and U)**

oIf you have accidentally made an editing change that you shouldn't have, **vi** provides the **u** command for undoing the last change made.

oJust press

`u`

oTo redo some of the actions, you have to use *[Ctrl -r]*

3.2.21 String Search

3.2.21.1 Search for a Pattern

o You can search for a pattern (string or regular expression) by prefixing the string or expression with a / (frontslash).

o To locate the first occurrence of the string editor, simply enter

/editor[Enter]

o When you press the /, it is echoed in the last line of the screen. Enter the pattern and press enter.

o The cursor is then positioned on the first character of the first occurrence of this pattern, starting from the present location of the cursor.

o To search in the reverse direction, you have to use ? Instead of the /. The sequence

?editor[Enter]

o Searches backward for the most previous

3.2.21.2 Repeating the Last Pattern Search (n and N)

- oFor repeating searches **vi** uses the **n** and **N** commands.
- oFor repeating a search in the direction in which the previous search was made with / or ?, use **n**
- oFor repeating the search in the reverse direction, use **N**
- oThe cursor will be positioned at the beginning of the character forming the pattern.

3.2.21.3 Searching for a Character

- o**Vi** uses single-letter commands to move to a specific character in a line-**f** and **F**.
- oEither of them is followed by the character that has to be located.

fe Use F for backward search

- oNote that the search is restricted to the current line only.

- oYou can repeat the search using the commands ; and ,(comma).

- oTo search for the next occurrence of the previous character in the current line, use

; Use , for reverse search

3.2.21.4 Search and Replace (:s)

- o**vi** uses the Last Line Mode's **:s** command to perform substitution-replacing one string or expression with another.

- oLike **w,s** can also be preceded by an address, in which case the substitution affects only the lines addressed.

- oThe syntax can be understood by the following example which replaces every occurrence of the

oThe address 1, \$ represents all lines in the file. The patterns are separated by /s, and g makes the substitution **global**.

oIf the g parameter is not used, only the first occurrence of the pattern in each line is replaced; the rest are left alone.

3.2.22 Handling Multiple Files

oVi uses the Last Line Mode to handle multiple files and buffers. You can open as many buffers as you want and switch from one to the other.

3.2.22.1 Inserting File and Command Output

oTo insert the contents of one file at certain location in another file, use the command:

:r !note1

oTo insert the output of date command below the current line,

:r !date

3.2.22.2 Switching Files

oTo stop editing the current file and edit file foo
:e foo

oTo return to the original file use:
[Ctrl-^] or **:e#**

oTo stop editing the current file and edit file foo after abandoning changes made to current file
:e! foo

oTo load the last saved edition of current file
:e!

Using vi with Multiple Filenames:

- oWhen **vi** is used with multiple filenames in the command line, it loads the first file into the buffer.
- oYou can move to the next file or rewind the file list to start from the beginning:

:n	<i>Next file in command line</i>
:rew	Back to first file in command line

o**vi**'s protection mechanism prevents switch to another file if changes made to the current one remain unsaved.

oThe **!** is the universal overriding switch.

oUse it freely (as you did with **:q!** and **e!**) with any of the Last Line Mode commands whenever you want to ignore changes.

3.2.23 Marking Text

oYou can mark several positions in the file and later locate them. Unlike the bookmarks offered by browsers, these marks are invisible and disappear when you quit the editor.

oBoth the mark and locate commands have to be used with a letter to mark a location or access a unique mark.

oTo mark a location with the character q, move the cursor to the required location and press

mq

oAfter you have moved your cursor away from the portion of the marked text, you can move back to this mark by using

'q

oBy using the different characters of the alphabet, you can mark up to 26 location in the file in this way.

4.19 Filtering Text- The ! Operator

oWith the ! Operator you can sort the contents of your editing window without leaving the editor.

oFor instance, to sort line 2 through 5, move the cursor to the first character of line 2(with 2G) and then press the following sequence:

!5Gsort*[Enter]*

oNothing appears on the screen till the second address (5G) is entered. After you enter the word sort, the last line would look like this:

:.+.3!sort

oAfter you press *[Enter]*, the 20 specified lines are sorted and the output replaces the current contents