I  Introduction to vi

The UNIX command cat can be used to create and write a file, but it is too limited a tool for all except the simplest cases. Usually you will use a text editor which has an extensive range of commands to create and modify files. UNIX has several text editors (such as ed, sed, vi, emacs). These notes refer to the visual editor, vi.

If there are so many other editors available, why do we insist that you learn this complicated one called vi? Well, vi is a very powerful editor, when you become experienced, and unlike some other popular editors, it is available in nearly identical form on nearly every UNIX system. This means that once you have mastered this editor, you will be able to create and modify files whatever UNIX machine you are using.

vi is called a visual editor because you can see a copy of the file you are creating or modifying on the screen while you are editing. vi is a powerful and widely-used editor. Here, you will be introduced to only a few of its many commands and capacities.

The vi editor has two distinct modes — command mode (with its own set of commands), and insert mode. In command mode, most commands (key strokes) either move the cursor around the editor buffer, or enable you to alter the text (change, move or delete it) in the editor buffer. Other commands (including i, a, o) cause the editor to change to insert mode. All characters typed in insert mode are added to the file. Typing the esc key causes the editor to leave insert mode and return to command mode.

You should also know that when you enter the command: vi filename, a copy of that file filename is placed into an editor buffer, and the editing that you do is performed on this copy, not on the original file. The original file is only replaced by the changed text when you write (save) the editor buffer using the :w (or ZZ) command. If filename is a new file, the editor buffer is empty until you insert text, and the file is not created until you save or write the buffer.

Occasionally you will make such a mess with your editing that you will want to revert to the original without saving the alterations. the command :q! throws away the text in the editor buffer and leaves the original file unaltered. Alternatively you could use the command :wq filename2 and write your edited file to a new filename, filename2, leaving the original file unaltered (i.e. not over-written).

Here are two further items that you need to keep clearly in mind, to get you out of trouble:

- If you find strange and extraneous characters on your screen, or if the display is somehow distorted, it is possible that you have you have not set your terminal type correctly for the terminal you are currently using. To correct this, you need to exit from vi (type esc to exit from insert mode, then :q! to quit without saving), set your term variable correctly, and then resume editing. (Refer to the section UNIX Troubleshooting to see how to do this.)

- If you become confused as to what mode you are in (command mode or insert mode) while using vi, e.g. if it does not execute your commands correctly, press the esc key to return to command mode. It is always safe to press the esc key; it may sometimes be unsafe to press other keys.
1 vitutor

The program vitutor copies a text file on to your directory working area, and provides a relatively short introduction to vi. You should now access and proceed through this program, by entering the command:

\texttt{vitur}\texttt{or}

\begin{table}
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Basic vi Commands (for absolute beginners)} & \\
\hline
h, j, k, l & Move one character left/down/up/right \\
x & Delete the character currently under the cursor \\
i, a & Insert characters before/after cursor \\
:j! & Quit without writing the editor buffer \\
:wq or ZZ & Write editor buffer (overwriting original file), then quit \\
ESC & Exit from insert mode and return to command mode \\
\hline
\end{tabular}
\end{table}

\begin{table}
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{More Basic vi Commands} & \\
\hline
\texttt{nSPACE}, \texttt{nBACKSPACE} & Move \textit{n} characters forward/backward \\
\texttt{nRETURN}, \texttt{n-} & Move to start of \textit{n}'th next/preceding line \\
\texttt{nh}, \texttt{nj}, \texttt{nk}, \texttt{nl} & Move \textit{n} characters left/down/up/right \\
\texttt{nw}, \texttt{nb} & Move \textit{n} words forward/backward \\
\texttt{nG}, \texttt{(G)} & Move to line number \textit{n} (default: last line) \\
\texttt{^F}, \texttt{^B} & Scroll one screen forwards/backwards in your file \\
\texttt{ndl}, \texttt{ndw}, \texttt{ndd} & Delete \textit{n} chars/words/lines \\
\texttt{nx} & Delete \textit{n} characters (cf. ndl) \\
I, A & Insert characters at beginning/end of line \\
o, O & Open line for insertion below/above cursor \\
\hline
\end{tabular}
\end{table}

When you have finished the vi tutorial program, you should read again the brief description of vi given above; it should be more meaningful to you now. You might also like to go through the tutorial program again, to help consolidate your new learning.

2 Summary

Once you have completed this section, including going through vitutor, you should be able to commence using the vi editor to create and modify your files.

- Do \textit{not} expect to remember all the commands you covered in vitutor. Remember the basic ones given in the previous two tables to start with, and remember the sorts of things that \textit{can} be done.

- If you find you need to do something, but can't remember how, refer to the table at the back of this manual, or look at the file created by vitutor, tutor.vi, in your home directory. This file contains all the text of the tutor program, so you can look up anything in there.
Some of the more advanced features of vi are discussed in the next section of the manual. It is not necessary to cover these topics in first year Computer Science, but you may find them useful anyway.
J More about vi

This section of the manual discusses some of the more advanced features of vi. In first year, it is not necessary to know these features, but you may find some of them useful, so it will be well worth your while to read through this section at least once.

1 High level commands

Writing, quitting, editing new files

NOTE: any command starting with a colon (:) or slash (/) must be completed by pressing ENTER or RETURN. For other commands you do not need to press either of these keys, unless otherwise specified.

So far we have seen how to enter vi and to write out a file using either ZZ or :w. The first exits from vi, (writing if changes were made), the second writes and stays in vi.

If you have changed vi’s copy of the file, but do not wish to save your changes, either because you messed up the file or decided that the changes are not an improvement, then you can give the command :q! to quit from vi without writing the changes. You can also re-edit the same file (starting over) by giving the command :e!. These commands should be used only rarely, and with caution, as it is not possible to recover the changes you have made after you discard them in this manner.

You can edit a different file without leaving vi by giving the command :e name. If you have not written out your file before you try to do this, then vi will tell you this, and delay editing the other file. You can then use :w to save your work, and give :e name again, or carefully give the command :e! name, which edits the other file and discards the changes you have made to the current file.

Escaping to a shell

You can get to a shell to execute a single command by giving a vi command of the form :! cmd. The system will run the single command cmd. When the command finishes, vi will ask you to “hit return to continue”. When you have finished looking at the output on the screen, you should hit ENTER or RETURN and vi will redraw the screen. You can then continue editing. You can also give another : command when it asks you for a return; in this case the screen will not be redrawn.

If you wish to execute more than one command in the shell, then you can use the command :sh. This will give you a new shell. When you terminate the shell, by typing a ‘\:D, vi will clear the screen and continue.

If vi has been invoked from csh, ‘Z will suspend vi and return to the (top level) shell. When vi is resumed, the screen will be redrawn.

Marking and returning

The command ‘ ‘ (two back quotes) returns to the previous place after a motion of the cursor by a command such as /, ? or G. You can also mark lines in the file with single letter tags
and return to these marks later by naming the tags. Try marking the current line with the command `m x`, where you should pick some letter for x, say ‘a’. Then move the cursor to a different line and hit ‘a’. The cursor will return to the place which you marked. Marks will last only until you edit another file.

When using operators such as d and referring to marked lines, it is often desirable to delete whole lines rather than deleting to the exact position in the line marked by m. In this case, you can use the form ‘x’ rather than ‘x’. Used without an operator, ‘x’ will move to the first non-white character of the marked line; similarly, ‘’ will move to the first non-white character of the line containing the previous context mark ‘’.

Adjusting the screen

If the screen image is messed up because of a transmission error to your terminal, or because some program other than vi wrote output to your terminal, you can hit a `L to refresh the screen.

On a dumb terminal, if there are ‘@’ lines in the middle of the screen as a result of line deletion, you may get rid of these lines by typing `R to cause vi to retype the screen, closing up these holes.

Finally, if you wish to place a certain line on the screen at the top, middle or bottom of the screen, you can position the cursor to that line, and give a z command. You should follow the z command with ENTER or RETURN if you want the line to appear at the top of the window, with . if you want it at the centre, or with -- if you want it at the bottom.

2 Special topics

Recovering lost lines

You might have a serious problem if you delete a number of lines and then regret that they were deleted. Despair not, vi saves the last 9 deleted blocks of text in a set of registers numbered 1–9. You can get the n’th previous deleted text back in your file with the command " np. The " says that a buffer name is to follow, n is the number of the buffer you wish to try (use the number 1 for now), and p is the print command, which puts text in the buffer after the cursor. If this doesn’t bring back the text you wanted, hit u to undo this and . (period) to repeat the put command. In general, the . command will repeat the last change you made.

As a special case, when the last command refers to a numbered text buffer, the . command increments the number of the buffer before repeating the command. Thus a sequence of the form "1pu.u.u. will, if repeated long enough, show you all the deleted text which has been saved for you. You can omit the u commands here to gather up all the text in the buffer, or stop after any . command to keep just the then-recovered text. The command P can also be used rather than p to put the recovered text before instead of after the cursor.

Recovering lost files

If the system crashes, you can recover the work you were doing to within a few changes. You will normally receive mail when you next login, giving you the name of the file which has
been saved for you. You should then change to the directory you were using when the system
crashed and give the command

\texttt{vi -r \textit{name}}

replacing \textit{name} with the name of the file which you were editing. This will recover your work
to a point near where you left off.

\textit{Do NOT use ZZ to write the recovered file unless you are sure you have changed it since it was
recovered, or vi will simply discard the file.}

You can get a listing of the files which are saved for you with the command

\texttt{vi -r}

If there is more than one instance of a particular file saved, \texttt{vi} gives you the newest instance
each time you recover it. You can thus get an older saved copy back by first recovering the
newer copies.

\textbf{Continuous text input}

When you are typing in large amounts of text, it is convenient to have lines broken near the
right margin automatically. You can cause this to happen by giving the command

\texttt{:set \textit{wm}=10}

This causes all lines to be broken at a space at least 10 columns from the right hand edge of
the screen.

If \texttt{vi} breaks an input line and you wish to put it back together you can join lines with \texttt{J}. You
can give \texttt{J} a count of the number of lines to be joined, as in \texttt{3J}, to join 3 lines. \texttt{vi} supplies
white space, if appropriate, at the juncture of the joined lines, and leaves the cursor at this
white space. You can kill the white space with \texttt{x} if you don't want it.

\textbf{Features for editing programs}

\texttt{vi} has a number of commands for editing programs. The thing that most distinguishes editing
of programs from editing of text is the desirability of maintaining an indented structure in the
body of the program. \texttt{vi} has an \textit{autoindent} facility for helping to generate correctly indented
programs. To enable this facility give the command

\texttt{:set ai}

Now try opening a new line with \texttt{o} and type some characters on the line after a few tabs. If
you now start another line, \texttt{vi} supplies white space at the beginning of the line which lines it
up with the previous line. You cannot backspace over this indentation, but you can use \texttt{^D}
to backtab over the supplied indentation.

Each time you type \texttt{^D} you back up one position, normally to an 8 column boundary. This
amount is settable; \texttt{vi} has an option called \texttt{shiftwidth} which you can set to change this
value. Try giving the command

\texttt{:set \textit{sw}=4}

and then experimenting with autoindent again.

For shifting lines in the program left and right, there are the operators < and >. These shift
the lines you specify right or left by one `shiftwidth`. Try `<<` and `>>` which shift one line
left or right, and `<L` and `>L`, which shift the rest of the display left and right.

If you have a complicated expression and wish to see how the parentheses match, put the
cursor at a left or right parenthesis and hit `%`. This will show you the matching parenthesis.
This works also for braces `{` and `}`, and brackets `[` and `]`.

If you are editing C programs, you can use the `[[` and `]]` keys to advance or retreat to a line
starting with a `{`, that is, a block at a time. When `]]` is used with an operator it stops after
a line which starts with `}`; this is useful with `y]]`.

### 3 vi and ex

vi is actually one mode of editing within the editor ex. When you are running vi you can
escape to the line oriented editor of ex by giving the command :q. All of the : commands
which were introduced above are available in ex. Just give them without the ;, and follow
them with a `ENTER` or `RETURN`. Likewise, most ex commands can be invoked from vi using :

In rare instances, an internal error may occur in vi. If this happens, you will get a diagnostic
and be left in the command mode of ex. You can then save your work and quit, by giving
the command `x` after the ex prompt ;, or you can re-enter vi by giving ex the command vi.

There are a number of things which you can do more easily in ex than in vi. Systematic
changes in line-oriented material are in this category. The section below gives an introduction
to some simple ex commands1. Experienced users often mix their use of ex command mode
and vi command mode to speed the work they are doing.

#### ex commands used in vi

This is a brief introduction to some of the ex commands often used within vi.

Since ex is a line oriented editor, the commands given must specify which line or lines to
affect. Lines are specified in the following ways:

- **Current line** The current line is referred to as line dot, ie. `.`; and is officially the last line
  you did anything to. In vi this is the line the cursor is on.

- **Line numbers** In vi the command `^G` can be used to find out the line number of the current
  line.

- **Last line** When specifying line numbers the symbol `$` refers to the last line in the buffer.

- **Context search** Lines can be specified by giving a pattern to search for. The first line that
  matches the given string is the line on which the command is applied.

Where commands are shown with two lines specified, the operation is applied from the first
to the second inclusively. However it is not necessary to always specify two lines. If one line
is given the comma is omitted. If no line is specified, the current line is assumed.

1If you need more information refer to the Advanced Editing on UNIX or the Ex Reference Manual
with the editor `ed`, but all of the commands discussed apply exactly to ex.
Print

The *ex* print command is not very useful in *vi*, but it is given here because it is a harmless command to practice line references with. The print command has the following format:

:line1, line2 p

Some examples are:

.: .,+2p

which prints the current and next 2 lines;

:10,$-1p

which prints line 10 down to the second last line of the file; and

: /the/, /and/p

which prints all the lines between the first occurrence of the word ‘the’ and the first occurrence of the word ‘and’.

Substitute

The format is:

:line1, line2 s/string1/string2/

The effect is that wherever string1 is found between the lines specified, string2 is substituted for it. String1 can be a regular expression. For example:

:s/er/or/

changes er to or on the current line. Also

:s/x.y/xy/

gets rid of the character between x and y irrespective or what character was present.

Note that if string1 is not found nothing happens but a warning message appears on the bottom of the screen.

In string2 there are only two special characters:

\ (backslash) and \& (ampersand)

The backslash \ has the same effect as in string1. Ampersand \& in string2 refers to the string that was matched by string1. For example:

:s/computer/&s/

pluralises the word ‘computer’ with minimum typing, and

:s/.*/(&)/

puts parentheses around the whole line.

All of the other regular expression special characters have no extra-ordinary affect in string2. For example:

:s/x.y/x.y/

replaces any character found between x and y with a literal dot.
Global substitute

In the above examples given for the substitute command, the normal vi mode commands would have been just as effective in making the changes. However the substitute command becomes much faster than using vi commands when repeated changes over several lines, or the whole file have to be made.

For example, if you misspelt someone’s name several times in the first ten lines of a letter, the command

:1,10s/Lyn/kne/

will change any ‘Lyn’s found on the first ten lines to ‘Lynne’.

However if more that one occurrence of ‘Lyn’ is found per line, only the first occurrence will be changed. To get around this put a g on the end of the command:

:1,10s/Lyn/kne/g

To specify that you want to make the change in all the lines of the file you could use

:1,$s/Lyn/kne/g

or

:g/Lyn/s/Lyn/kne/g

The second one says to globally search for the string ‘Lyn’ and where it is found apply the substitution. This can also be written as:

:g/Lyn/s//&ne/g

The empty string1 after the s indicates that it is the same as the string that was just searched for.

When complex substitutions are needed it is often good to quickly check the lines that were changed to make sure that it did what you thought it would. The print command can be combined with the substitute command to display all of the affected lines. Just put a p on the end of the command. That is:

:g/Lyn/s//&ne/gp

Delete

The delete command has the format:

:line1, line2 d

This command is useful in vi if you want to delete more than one screenful of text. For example:

::,$d

will delete from the current line to the end of the file in one step.

Write

The write command w was introduced in the section “Writing, quitting and editing new files”, but this command can also take line numbers and file names. For example:
:1,/"Dear/-1 w > heading
will make a copy of the heading section of a letter in the file called heading. The line /"Dear/-1 is the line before the one beginning with 'Dear'.

Read

The read command is used to read a file into the buffer. If there is a file in the buffer already the effect is to add the text of the new file to what is already there. The text of the new file is placed after the current line. This command is used by positioning the cursor at the appropriate position and issuing the command:

:r filename