1.0 Introduction

The Engineering Computer Network (ECN) supports the **VI** (visual)editor and it is available on all of our major system computers.**VI** is a display oriented, interactive text editorwhich allows a user to create, modify, and store files on the computer via a terminal. Files commonly created by ECN users consist of suchinformation as program code, data, written reports, and letters. When using the **VI** editor, the screen of your terminal acts as a "window" to view the file you are editing.It is this viewable portion of the text that you can alterthrough a command sequence.The changes you instructed **VI** to performare reflected on the screen after completing the command sequence.

This document is intended to provide a general introduction to the**VI** editor for the beginning user. It does not provide a comprehensive coverage of all **VI** commands and features, but rather focuses on those commands andfeatures which will be most useful to the beginner.This tutorial discusses the use of many commandsand while they can all be useful,a new user would be wise to select only a core of commandsand use these until comfortable.This can help avoid that "overwhelmed" feelingthat learning something new on a computer often engenders.Later, reviewing the tutorial can lead to trying additional commands.

To use this tutorial,only a minimal familiarity with ECN computer usage is assumed:the user should be able to log in and should have a basic understanding of the UNIX file system.

All users are advised to read this tutorial on a machinecapable of showing the graphics (for example: a Sun workstationor a MacIntosh). The graphics provide visualinformation that is referenced in the text; therefore,a reader missing the graphics will also miss vital information.

()
() ()

ENTERING OR CALLING VI ()

()
() ()

2.1 Calling VI ()

To call the **VI** editor and begin an editing session, type at the UNIX prompt (%):

```
%  vi filename [RETURN]
```

where *filename* is the name of the file that you want to edit. **VI** automatically sets aside a temporary buffer as a working space for use during this editing session.

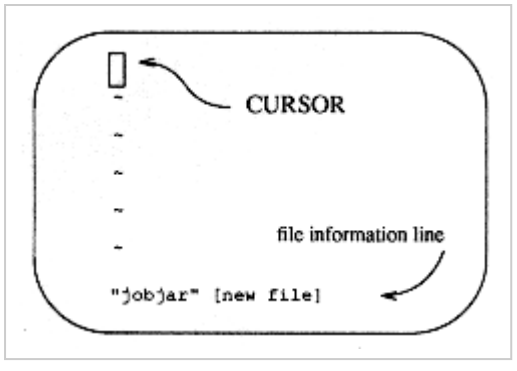
()
() ()

2.2 Editing A New File ()

If the file you specify when calling **VI** does not already exist in your current working directory, a new file will be created with the name that you have specified. For example, if you wanted to create a new file called *jobjar*, you would type:

```
% vi jobjar [RETURN]
```

The screen will flicker and clear and you will see:



When the **VI** editor creates a new file as in the previous example,the cursor will be placed on the screenin the home position (upper left-hand corner).The cursor on most terminals is a rectangle that completely covers the character space it is situated uponand reflects your current position within the buffer.Below the cursoralong the left side of the screen, is a tilde (~) characterpositioned in the first character space on each line. The tilde is used to indicate a void area in the file wherethere are no characters, not even blank spaces.At the bottom of the screen, the editor displays a file information linewhich presents the name of the file being edited, in this case *jobjar*, followed by "[New file]" to indicate that this is a new file and it has never been opened before.
()
() ()

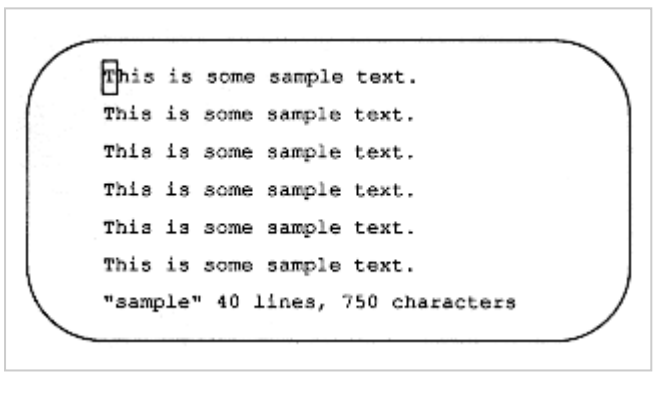
2.3 Editing An Existing File ()

The **VI** editor allows a user to modify and add materialto an already existing file.This is done by using the name of a current file when calling **VI**.**VI** will then fetch a copy of the contents of the specified file from disk storage, leaving the original undisturbed,and place the copy into the buffer.It is here, in the buffer, where **VI** permits you to do your editing.The idea of editing in the buffer is very beneficial,because if you really botch up your editing job (such as deleting a majorsection) the original file on the disk remains unaffected.

Upon opening a file,**VI** will display the first screenful of text starting at the top of the file, place the cursor in the home position,and print the file information line which displays the name of the file along with the number of lines and characters it containson the last line of the screen.For example, to edit the file *sample* which has 40 lines and 750 characters, call the **VI** editor with this command:

```
% vi sample [RETURN]
```

The screen will flicker and clear.Then you will see:



As shown in this example the cursor sits in the home position,it completely overlays the first character of the file.When the cursor is moved,this character will be exposed, unchanged.In general,most terminal screens will display only 20 to 24 lines of text.If a file has more text than will fit on one screen, the unviewed material remains in the buffer until needed.

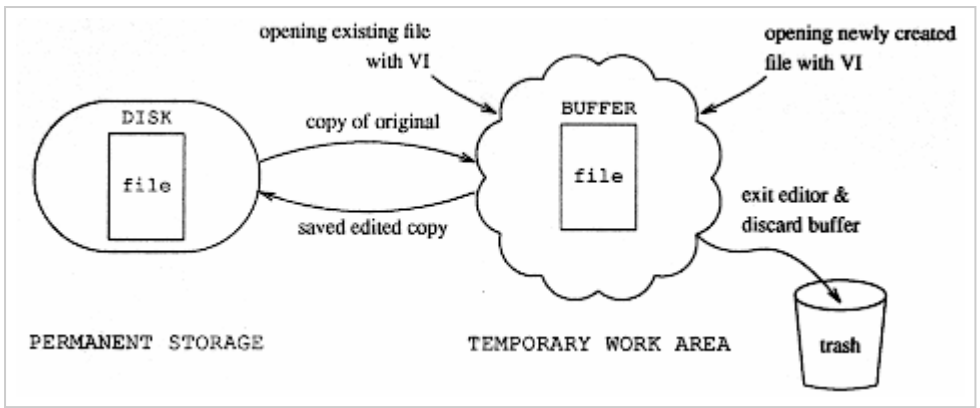
()
() ()

2.4 Saving or Storing A File ()

As mentioned before,when the **VI** editor is invoked a temporary working areais created called a buffer.This buffer or work space is only available for your use while you are using the editor.It is into this buffer that **VI** either places a duplicate copy of the file as it exists on the disk or creates a new file.When you exit the editor,this work area is discarded. Disk storage, on the other hand,can be thought of as a permanent storehousefor files until discarded by using the UNIX remove command.

Few things are more upsetting than to spend many hours keying information into a file, simply to have a careless keystroke close the buffer and destroy your work.Material you have keyed into the buffer file is only retained for future access when the file is "saved" on the disk.Material not "saved" on the disk is discarded when exiting

VIand the buffer is closed.The next figure depicts this relationship.

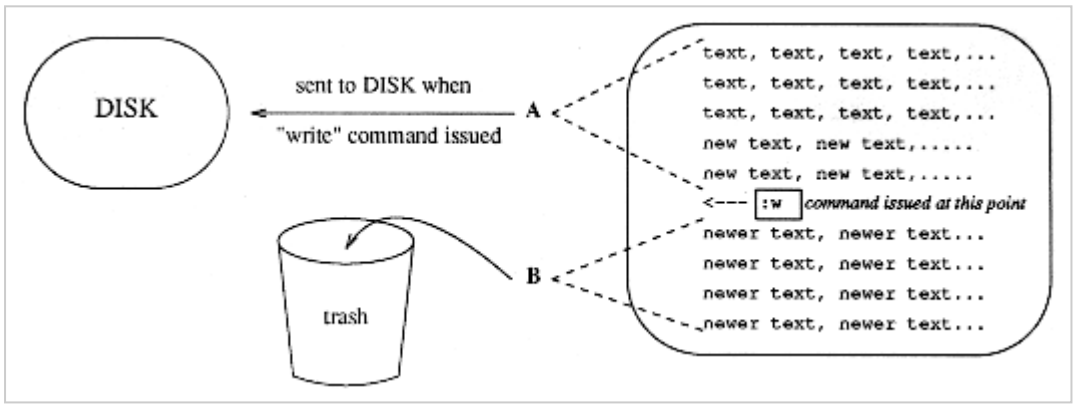


To avoid the frustration of rekeying material, you should periodically "save" the work done to that pointby "writing" the new material or alteration to the permanent disk storage. This is done by typing "**:w**"(write).As soon as you type "**:**"the cursor hops to the bottom of the screen and it is here that the complete "**:w**"command is echoed.**NOTE:** It is difficult in simple drawings to show the cursor and the command in boththe position where the command is issued plus the locationat the bottom of the screen where it is echoed;therefore for brevity, in the following examples you will see the commandpresented at the point in the text where it is issued.

When the user instructs the editor to save material with "**:w**", a copy of all material in the buffer is transported to the disk,overwriting the previous contents of the disk file.As a general rule,it is suggested that you "write" material to the disk every 15 to 20 minutes, or after you have keyed in enough information or corrections that it wouldirritate you if you needed to rekey it in all over again.

On the next page is an example of a file containing 3 lines of original textand the user keys in 2 additional lines followed with "**:w**".When the "**:w**" command is issued,the disk file copy will be overwritten with the 5 linesas they exist in the temporary work area.This material is referenced as section "**A**" in the next example.

What happens to your file if the system crashes or you make a dumbmistake in the midst of an editing session? Well, if your file looked like the one below when disaster struck,all material in section "**A**" would be safe because the "**:w**"would have caused everything to that point to be written to the disk.The material in section "**B**" might be completely or partially destroyed because it only exists on the temporary buffer.



An editing session can last for minutes or hours but the user can be assured that once material is written to the disk,the storage is permanent and the material will remain there until needed again.

3.0 Exiting VI

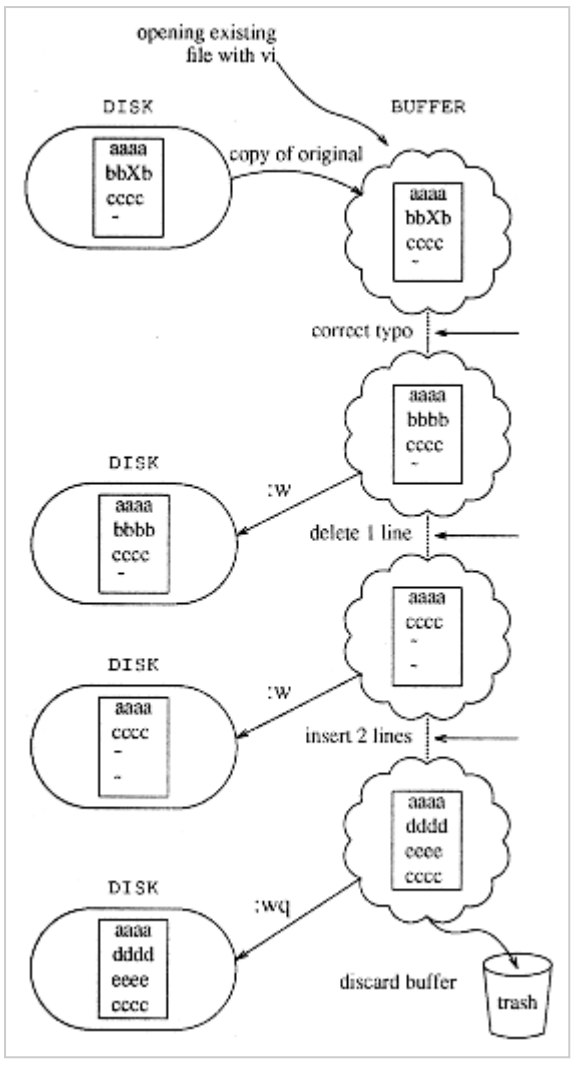
When you have finished your editing,you need to inform **VI** that you wish to "quit" the editorand return to your shell.This is done by typing "**:q**" (quit).**VI** will respond by updating the file information linewith the name of your file in quotes followed by thecurrent number of lines and characters.After exiting the editor,the UNIX prompt will again appear on your screen.
()
() ()

3.1 Exiting the Editor -- Retaining Changes ()

VI requires that the buffer be empty of newly edited materialwhen you type "**:q**".Thus if you have made any alterations to the file since the last time you typed "**:w**",**VI** will not know how you want these changes handled.The editor will print the statement:

```
    No write since last change (:quit! overrides)
```

at the bottom of the screen.If you decide you want to retain these changes,you must type "**:w**" before issuing a new "**:q**".Most system users get in the habit of combining these two commandsinto a single command "**:wq**" (write and quit).



The **VI** editor allows a user to repeatedly make alterations to a file.This is very handy when you are preparing a term paper, computer program,or report because each time you reread a section you conjure up better ways to word your thoughts.

Consider the simple file in this example,The file has 3 lines when opened and is then edited as follows:

1. The user corrects an error on line 2 that existed from a previous editing session.When the "**:w**"command is issued,all 3 lines on the disk file will be overwritten with the 3 lines as they exist on the buffer.

2. Next, the user decides to delete the 2nd line.When the "**:w**"command is issued all 3 lines on the disk filewill be overwritten with the 2 lines as they exist on the buffer.

3. Later, 2 new lines are inserted into the buffer copy between lines 1 and 2.When the "**:wq**"command is issued, the 2 lines on the disk filewill be overwritten with the 4 lines as they exist on the buffer.The contents of the buffer are discarded with the termination of the editing session.

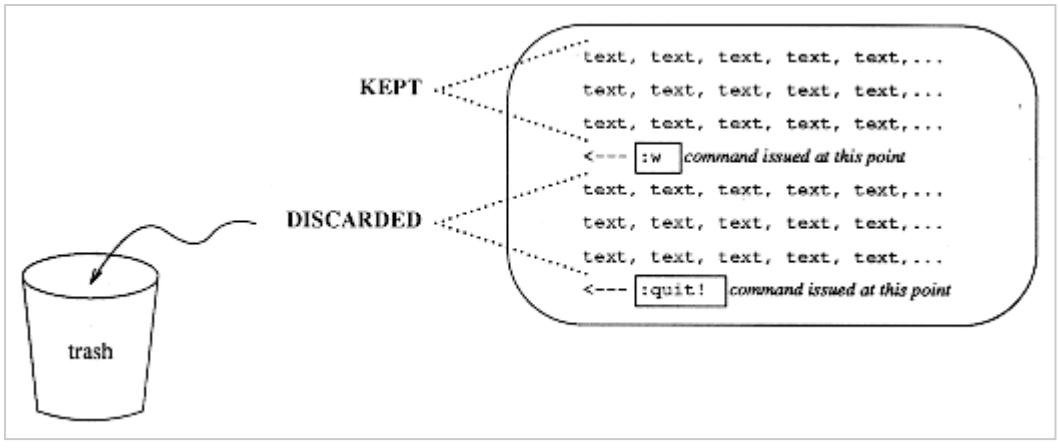
()

() ()

3.2 Exiting the Editor--Discarding Changes ()

Sometimes when working on a file,it is desirable to leave the editor without saving the modifications.This is accomplished by typing "**:quit!**".You might use this command if you started to edit a fileand do not like the way the changes are shaping up.When you use this command,**VI** will immediately discard all alterationsmade to your file since the last "**:w**".If you have not used a "**:w**"since opening the file,all changes since the beginning of the editing session will be abandoned.

The action of the "**:quit!**" command is illustrated below.Initially when the file was opened,it contained 3 lines.Some form of editing was done on these lines and the work was saved with the "**:w**" command.Additional material was keyed in,but the decision was made to discard the newer additions.When the"**:quit!**"command is issued, all text back to the "**:w**"is disposed of.



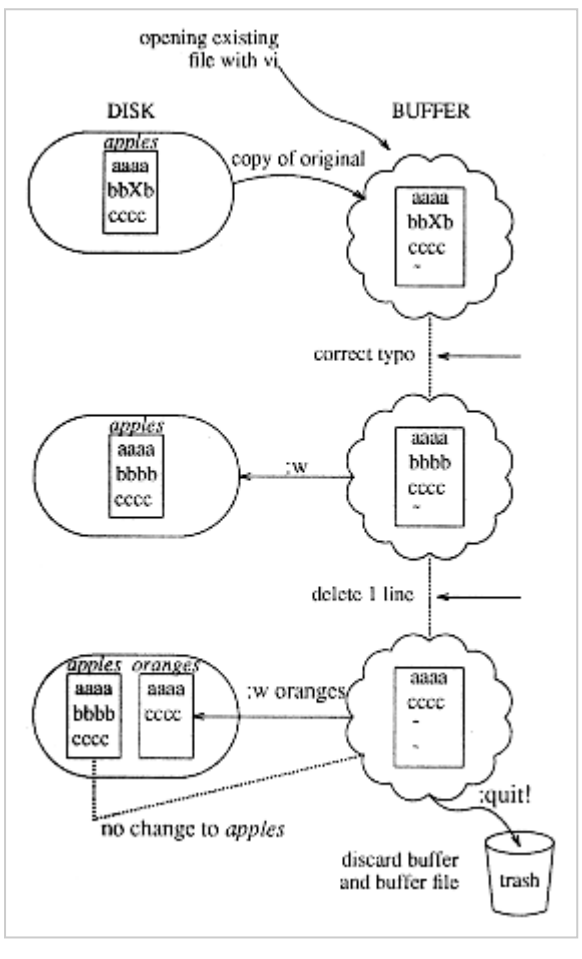
BE CAREFUL!!!When you use the"**:quit!**"command, the editor believes you know what you are doing and will immediately follow your instructions.You will not have a second chance to retrieve your workonce the buffer is closed and the file is discarded;so, key in this command with utmost caution.

()

() ()

3.3 Exiting VI--Keeping Changes and Original Text ()

There may be a time when you are editing a file and can not decideif you want to keep the current disk copy or retain the most recentchanges. **VI** offers you a way to "have your cake and eat it too".This is done by writing the current contents of the buffer into a new file.For example,if the original file is named *apples*,you could write the material from the buffer into a file named *oranges*then exit the editor discarding the buffer contents with "**:quit**".The file *oranges* contains the altered text and the file *apples*contains text as of the last issued "**:w**".



The sample file, *apples*, has 3 lines when opened and is then edited as follows:

1. The user corrects an error on line 2that existed from a previous editing session.When the "**:w**"command is issued,all 3 lines on the disk file will be overwritten with the 3 lines as they exist on the buffer.
2. Next, the user deletes the 2nd line,but can not decide to keep or discard the change,therefore; writes the contents of the buffer to a new file, *oranges* with"**:w oranges**".The buffer is then discarded with "**:quit!**"and the existing file, *apples*remains on the disk as it existed after the last "**:w**" was issued in step 1.

4.0 Organization of Vi

The **VI** editor is a sophisticated editor with a wide rangeof commands, yet the basic structure is simple.There are two modes of operation in **VI**,*command mode* and *text input mode*.*Command mode* is where you tell the editor what you want it to do.*Text input mode* is the portion of the editor where you key inmaterial (text, data, or program code) that you want retained in the file.*Text input mode* can only be accessed from *command mode*.The person who is able to visualize and separate the two modesof operation will very quickly master the control of this editor.The two modes of operation are illustrated in the block diagramat the end of this section.

()
() ()

4.1 Command Mode ()

Command mode is the initial state encountered when **VI** is invoked from the UNIX shell.It is in this state that **VI** receives instructions on what action is required, such as:

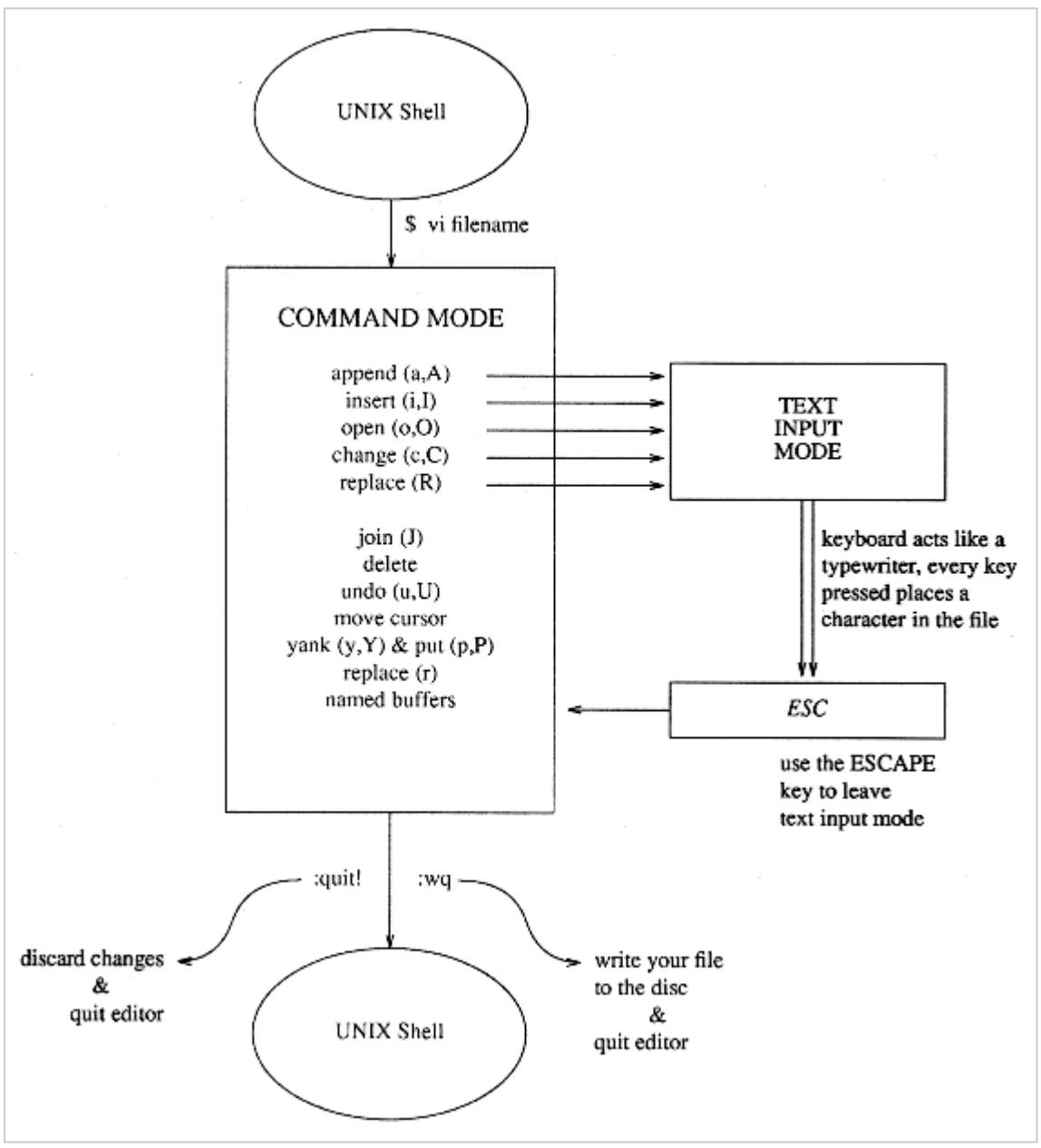
```
bringing text to the screen: scrolling, searching, using gotomoving the cursortransferring to text
input modetext insertion: append, insert, open, change, and replacereading in another filedeleting
(lines, words, or characters)changing (lines, words, or characters)copying or yanking (lines, wor
ds, or characters)undoing previous command actionusing named buffersjoining lines
```

It is important for the new user to realizethat when the majority of **VI** commands are keyed in,the editor does not echo the command back on the screen;rather, it simply executes the response the command requires.

In general, the steps given below will be repeated again and againeach editing session while in *command mode*:

1. place text in terminal window
2. position the cursor
3. give editing command

After a short time of using the editor,these actions will become second nature to youand performance of them will be done without concentration.



()
() ()

4.2 Text Input Mode ()

Text input mode is the second side of the **VI** editor.It is in this mode that you will key in the majority of text into your file.To enter *text input mode* a user would type,while in *command mode*, the editing command to append (a, A), insert (i, I), open (o, O), change (c,C), or replace (R).Without the initial command being echoed to the screen,the immediate response is for **VI** to transfer into *text input mode*. The user may see the cursor jump to a new locationdepending on the command issued.While in *text input mode*,all subsequent keystrokes will be

directly reflected in the file.The user may now key in as much information as desired;a single character, a chapter of a book, or more.The user leaves *text input mode* by depressing the"**ESC**"key.More information is provided on how to use the append, insert, open, changeand replace commands in section 7.

Most editing sessions include moving back and forth between *command mode* and *text input mode*.For example,a user may decide to change a word in one sentence (*text input mode*),move to the end of the file (*command mode*),add many additional pages of text (*text input mode*),backup half a page (*command mode*), correct some typos (*text input mode*),move again (*command mode*), and then inserta sentence in the middle of an existing paragraph (*text input mode*).This sounds complex, but don't worry,it is much easier to do than to read about.

5.0 Positioning Text on the Screen

As explained in the *Entering VI* section,when the **VI** editor is called,the opening screen will show thecursor located in the home position.If **VI** is opening an existing file,you will see the first 20 to 24 lines of the file on the screen.All additional lines copied from the disk will also be in the buffer, but not viewable on the screen.

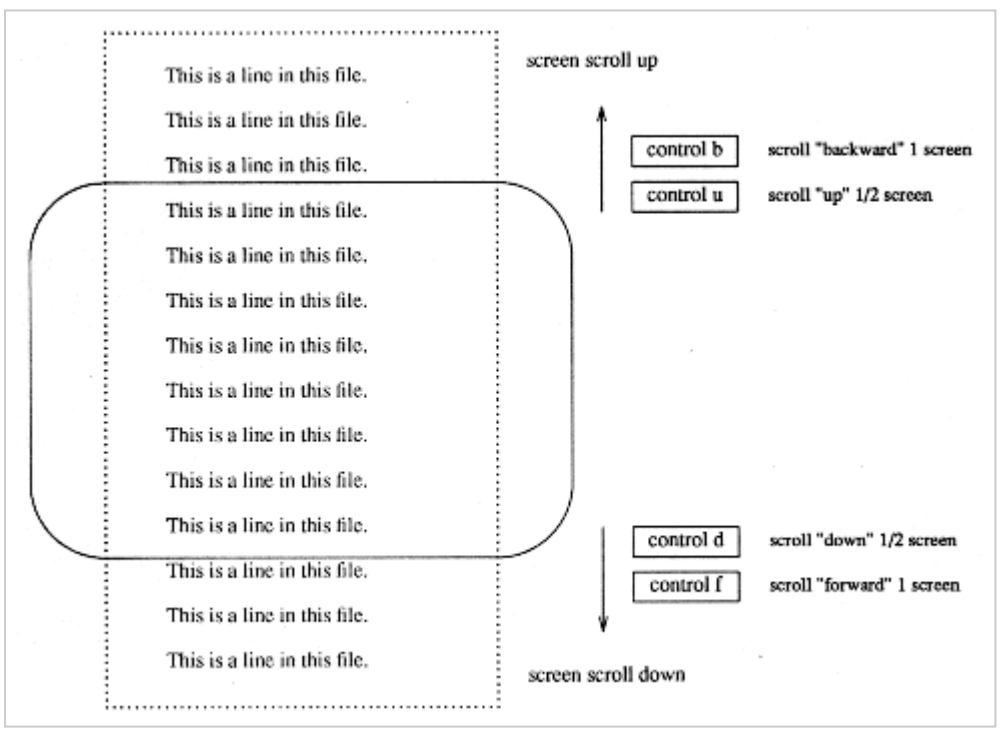
If the home position is the location from which you want to do your editing,then you are in great shape;unfortunately, most editing is performed at other locations within the file.So, your first course of action is to issue the proper text positioning command to cause the necessarytext to appear on the terminal screen.This can be done from *command mode* by scrolling the screen,using the goto command,or doing a pattern search.
()
() ()

5.1 Scrolling the Screen ()

To visualize scrolling,imagine that the file is one long, continuous page (like an ancient scroll)and that only a portion of the text is revealed on the screen at any one time.Thus, the only way to see another section of text is to realignthe terminal window to the file.

Depending on your point of reference,the user may imagine that the terminal screen floats up and down overa stationary text file,or that the text is pulled up and down with only a portion of it exposedto the stationary terminal screen.Either point of view arrives at the same result.

However, the designers of the **VI** editor have decided that the screenfloats up and down with respect to the file.As a result, the scrolling commands reflect the direction the screen moves past the text.These commands are mnemonic in nature.In other words,there exists a relationship between the command and its meaning.For example, when giving the command "**control d**",the terminal window moves "**d***ownward*" toward the end of the file.The command "**control u**",moves the terminal window "**u***pward*" toward the first line of the file.Thus, the window is "pushed" up or down revealing new text that eitherprecedes or follows the current screen.



()
() ()

5.2 Using the GOTO Command ()

Scrolling through one screen after another in order to reach the portion of the text you want to look atcan, at times, be laborious as well as boring.Sometimes it is easier to move the screen window directly to the location you are interested in viewing.If you know the line number or the general area you want to access,repositioning can be accomplished by using the "**G**" (goto) command.The goto command, preceded by a line number,such as "**250G**",will position the cursor on the first character space of line number 250.If the requested line is not currently on the screen,the screen will be redrawn with the requested line situated in the window.

Typing "**1G**"will move the cursor to the first line of the file and redrawthe screen if necessary. Not specifying a line number when typing "**G**",will move the cursor to the last line in the file.It is frequently handy to be able to pop to the beginning or end of a file;so "**G**" and "**1G**"are important commands to remember.

Discovering your current line numberand overall file size information can be gotten by typing"**control g**".The editor will then print:the file name, the line number,the number of lines in the file,and the percentage of the way through the file all at the bottom of the screen.Such a line might look like:

```
    "filename" line 17 of 122 -- 13%
```

Generally, new users find the "**G**"command easier to use if all lines are numbered.For more information on how you can have line numbers added to your files,refer to the *Miscellaneous Information* section.

()
() ()

5.3 Moving by Searching ()

Another method to reposition yourself within a fileis by identifying a word, phrase, or string of characters for the editor to locate.To institute a search,you type a forward slash mark (/) followed by a string of characters, terminated by a "**RETURN**".The cursor will hop to the bottom of the screenwhere the search command will be echoed.For example,if you wanted to find the word "catnip" in your file,you would type:

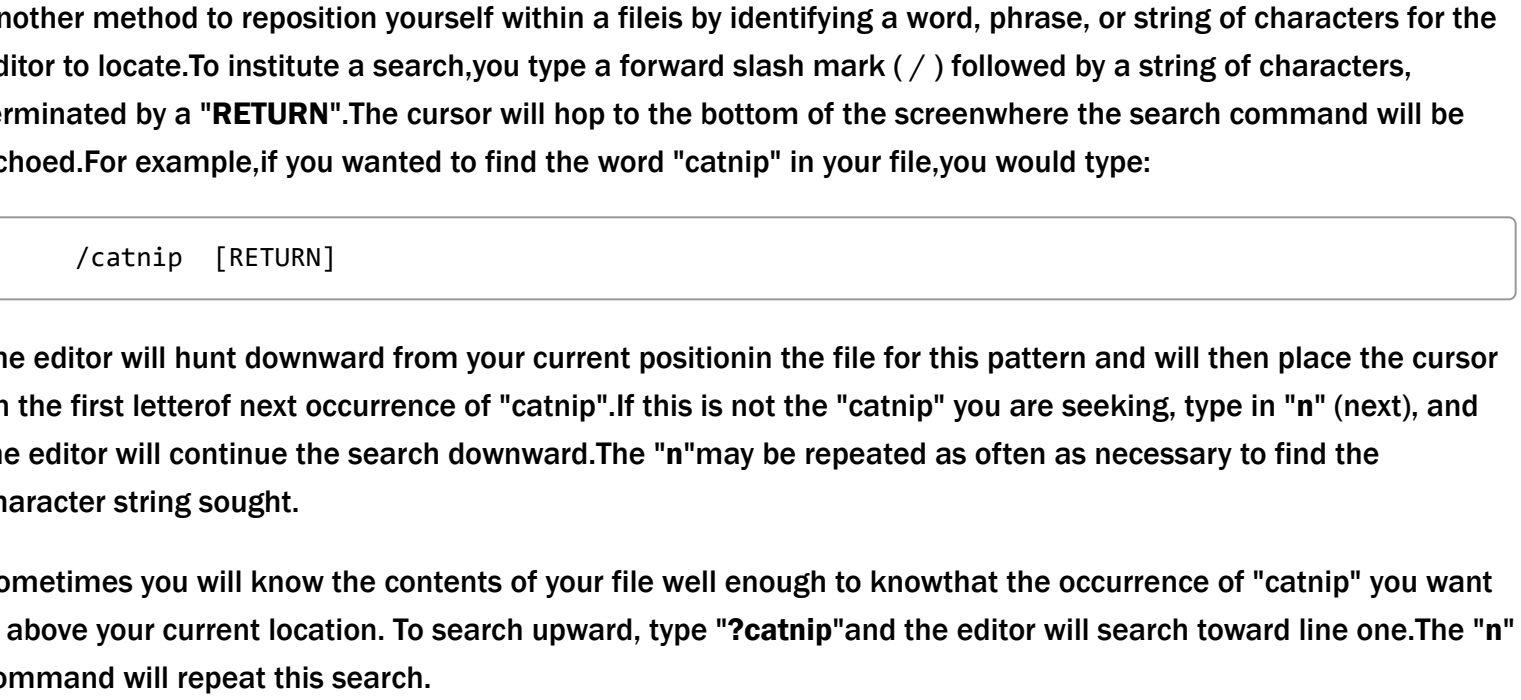
```
    /catnip  [RETURN]
```

The editor will hunt downward from your current positionin the file for this pattern and will then place the cursor on the first letterof next occurrence of "catnip".If this is not the "catnip" you are seeking, type in "**n**" (next), and the editor will continue the search downward.The "**n**"may be repeated as often as necessary to find the character string sought.

Sometimes you will know the contents of your file well enough to knowthat the occurrence of "catnip" you want is above your current location. To search upward, type "**?catnip**"and the editor will search toward line one.The "**n**" command will repeat this search.

Searching with the " / " or " ? " will ultimately process through the complete file.If you are searching upward and the pattern is not found before reaching the top of the file,the search will wraparound and the seeking will continue from thelast line of the file until the pattern is found.The reverse is true when searching downward.

After using the **VI** editor for a while,some users like to add the upper-case "**N**"command to their "bag of tricks".The "**N**"command reverses the direction of the search,i.e. if lower-case "**n**"is searching downward, upper-case "**N**"searches upward, and vice-versa.



If the pattern you have instructed **VI** to search for is notfound throughout the entire file,the editor will respond at the bottom of the screen with:

```
    Pattern not found
```

Remember, the editor will look per your instructions for an exactduplicate of the character string you have given it.If the editor does not locate your pattern,first check to see if you have mistyped the pattern you requestedthe editor to find.Note, spaces or absence of spaces can be important.If this is not the case,give thought to the possibility that the object in the file you aresearching for is misspelled.In that case, try the pattern search again but this time try using only a portion of the word,such as "tnip" rather than the full word "catnip".

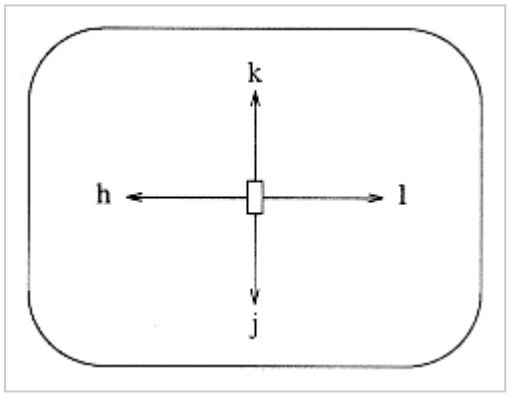
6.0 Positioning the Cursor

Congratulations, you have succeeded in getting the text you wantedto appear on the terminal screen.Now that you can see the target text,you can proceed to move in closer "for the kill".This will probably require a combination of major and minor cursor moves.Cursor movement commands are issued in *command mode*.
()
() ()

6.1 Major Screen Movements ()

Sometimes the cursor can be a substantial distance from the next editingposition on the screen.Large vertical jumps may be made with the use of the mnemonic commands:"**H**" for high or home, "**M**" for middle, or "**L**" for low or last screen line.When you type "**H**" "**M**" or "**L**"the cursor will immediately reposition to the first character space on thehighest, middle, or lowest screen line.

After using these three commands for a while,you may decide to become more precise in moving the cursorby adding a number to the command.For example, "**3H**"would move the cursor to the third line from the top of the screen.The command "**3L**"would likewise move the cursor to the third line from the bottom of the screen.As you may guess,the middle is always the middle and you can not fancy up the "**M**" command.



()
() ()

6.2 Minor Screen Movements ()

The most common means used to move the cursor is by use of "arrow" keys.Pressing an arrow key "**\(ua**" "**\(da**" "**->**" or "**<-**" will move the cursor one spacein the direction indicated.If the terminal you are using does not have arrow keys,you can use the "direction" keys "**h**" (left), "**j**" (down), "**k**" (up), and"**l**" (right)to move the cursor one space on the screen.Pressing direction keys in *command mode* will move the cursoras indicated in the figure to the right.**Watch out**, many people mix up the lower-case "**l**"with the lower-case "i" or the numeral "**1**".

The "**RETURN**" key is similar to the "**j**" key in that it moves the cursor down one line.However, the "**RETURN**"key always positions the cursor at the beginning of the next line down; whereas, the "**j**"key moves the cursor straight down from its present position,which may be the middle of a line.Moving several spaces may be accomplished by repeatedly pressing the"**RETURN**",direction or arrow key; such as,"**k**""**k**""**k**"to move upward 3 lines.You can also precede any of these keys with a number and achieve the same results,"**3k**".

Most speed typists prefer to move the cursor with the direction keysrather than arrow keysbecause they do not have to remove their fingers from the center of the keyboard.

()
() 6.3 Moving within the Line ()

Once you have located the cursor on the correct line,it may be necessary to fine tune the placement of the cursorstill further.You already possess the ability to move right and left by way of arrow or direction keys:"**l**" and "**->**" (right), or "**h**" and "**<-**" (left);and for a while this will probably be all you need to know.However, after you have worked with the **VI** editor for a while,it would be to your advantage to add the following line movement"scopes" to your command arsenal.Scope refers to the amount of text unit encompassed by a command key. ()

()

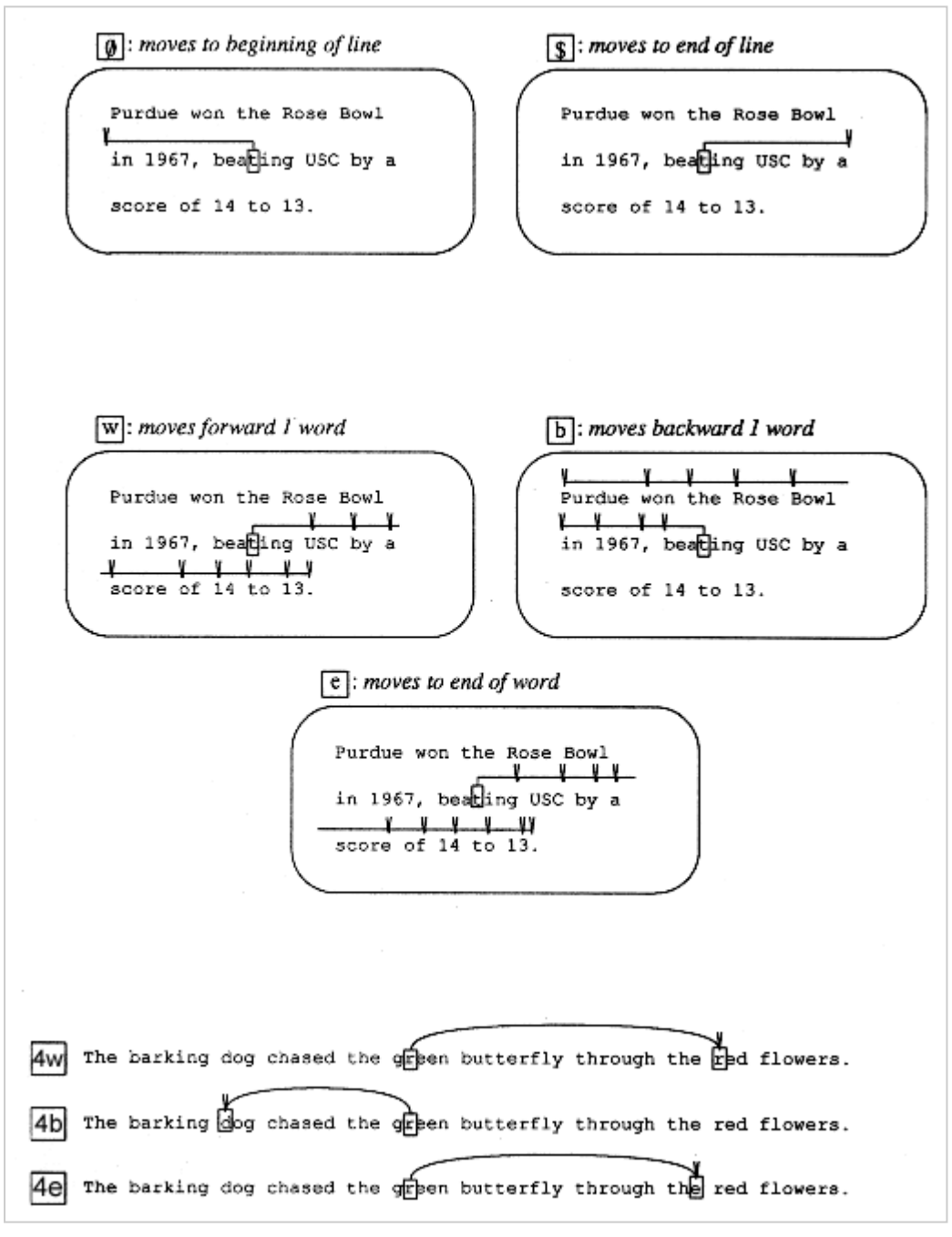
Scope	Text Unit Encompassed	Scope	Text Unit Encompassed
0	beginning of line (zero)	$	end of line
W	word right	w	word right
B	word left	b	word left
E	end of word right	e	end of word right

The figure on the next page provides an illustration ofhow the cursor would move using the scope keys.When you experiment with these keys on a terminal screen,you will notice that the "**0**" and "**$**"keys will jump to the beginning or end of the line the cursor is on,but will move no further.The "**w**", "**b**", "**e**"keys, on the other hand, will wraparound to the line below if moving right,or to the line above if moving left.The mini terminal screens on the next pageshow the path taken when a scope key is pressedrepeatedly causing the cursor to reposition again and again.It is important to remember that the cursor positionserves as a reference point for all scopes.Notice that when moving with the "**b**" or "**e**" key,if the cursor is sitting in the middle of a word,the remainder of that word is counted as a scope unit.Additionally, the lower-case scopes treat punctuation marksas a scope unit.In other words,a comma or period will be treated like a word in itself.

As with direction and arrow keys, the jumping power of"**w**", "**b**", or "**e**"can be multiplied by preceding the key with a number.At the bottom of the next page are three sentences illustratingthe effect of instructing the scope key to jump four units.In each example,the cursor is originally sitting on the "r" of the word "green".The arrow tip points to the destination location.

VI offers many variations for getting an editing job done.One of **vi's** "more-than-one-way-to-skin-a-cat" options,is the capitalization of the "**W**" "**B**" "**E**" keys.The action of the upper-case scope keys is the same as the lower-case scope keys,except the upper-case scope keys DO NOT see punctuation marksas a scope unit; therefore, DO NOT stop for punctuation marks.Use the form that works best for you...most people do not try toremember both variations.

The power of scopes is greatly increased when combined with operators.This concept is discussed further in the *Correcting Text* section.



7.0 Text Insertion

Every editor provides a method to place text into a fileand the **VI** editor offers you not one, but several convenient methods.For your sanity,it is suggested you read through this section,select one way of adding text,practice that method until you become proficient,and then begin to include the other methods as your skills increase.Attempting to learn to use all commands at once,while possible, leads to frustration and angry words.

The append, insert, and open commands all move the editor from*command mode* into *text input mode*.To use these commands efficiently,it is necessary to have a firm understanding of the meaning of thesetwo modes.More information about modes is contained in the*Organization of VI* section.

As you recall,while in *text input mode* the terminal keyboard acts just like a typewriter and every key pressed enters a character into yourfile.The most common error experienced by users when using **VI** isforgetting which mode they are in. This forgetfulness often leads totrying to issue a command while in *text input mode*. This results in an odd combination of characters being placed into the textwhich frequently causes the user to direct negative words at the terminalfollowed by a little clean-up editing.If you are ever in doubt about which mode you are in, press the "**ESC**" key.If you are in *command mode* when you press "**ESC**",the terminal will "beep".If you are in *text input mode*, all will be silent.Of course, pressing the "**ESC**"key while in *text input mode* has now placed you in *command mode*,but at least you know where you are.

A user may decide to place text into a brand new file with eitherthe insert or append command.The resulting action as seen by the user is the same with both commands. The open command can also be used on a new file,but differs from insert and append because open adds a blank line.Experiment with these commands to see their differences.
()
() ()

7.1 Append Command (a, A) ()

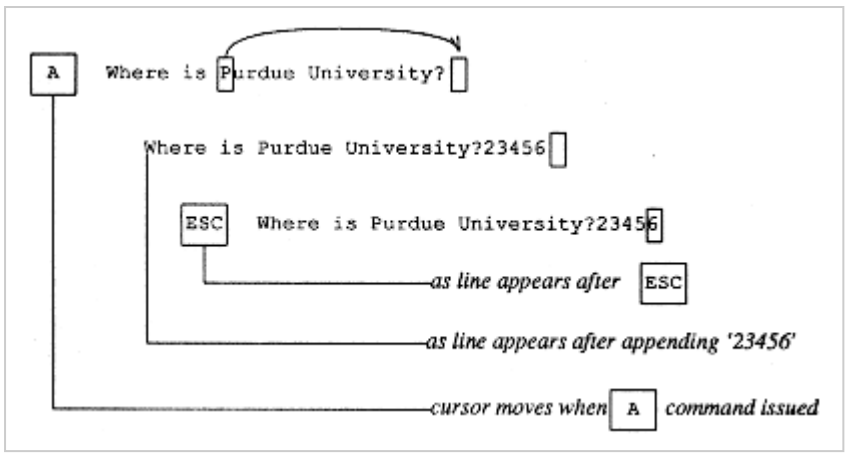
The append command means "to add after" and comes in two forms,the lower-case "**a**" and upper-case "**A**".This frequently used command allows you to place as much informationinto a new file or an existing file as you wantuntil you exit *text input mode* by pressing the "**ESC**"key and return to *command mode*.

Pressing the "**a**"key will cause the cursor to move one space to the right of the currentcursor position and await the new text.On most terminals,when you are appending text,the existing text seems to be overwritten.This is only temporary.As soon as you exit *text input mode* with the "**ESC**" key, the screen is redrawn and the material that was temporarily covered again appears.

The "**A,**"form of append works in much the same way,except when the command is issued the cursor jumps to the end of the lineand it is at this point that you enter *text input mode*and proceed to add text.Pressing "**ESC**"returns you to *command mode* and the screen is redrawn.

Below is an example of the lower-case "**a**"command appending new text after (to the right of) the cursor:

This is an example of the upper-case "**A**"command which appends after the last character in the line:



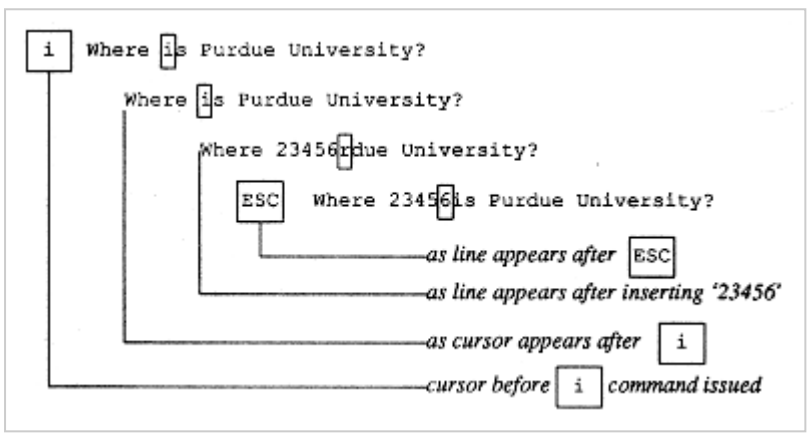
()

() ()

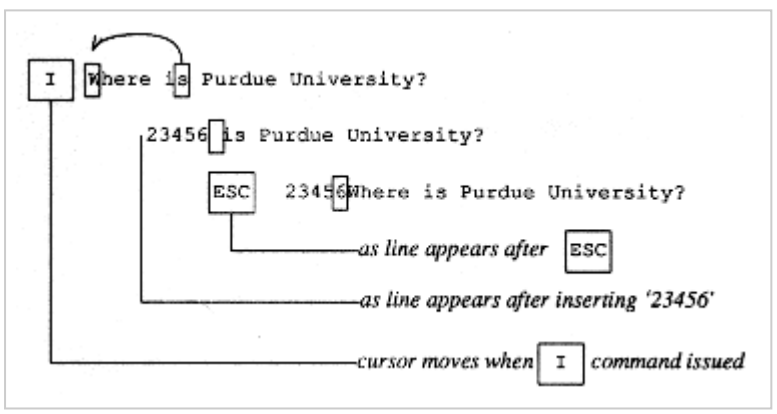
7.2 Insert Command (i, I) ()

The insert command is used to insert text into the file being edited.The lower-case "**i**"command inserts the new text you type to the left of the cursor.The upper-case "**I**" command inserts the text at the beginning of the current line.By giving the insert command,*text input mode* is activated,allowing you to enter as much text as you want.To stop entering text you must press the "**ESC**" key.On most terminals while you are inserting new material,old text seems to be overwritten.All this text will reappear unchanged after the newly inserted textwhen you exit *text input mode*and reenter *command mode* by pressing the "**ESC**" key.

The example below demonstrates how the lower-case "**i**"command functions inserting new text before (to the left of) the cursor:



This example illustrates the upper-case "**I**"command which inserts new text before (to the left of) the firstcharacter in the line:



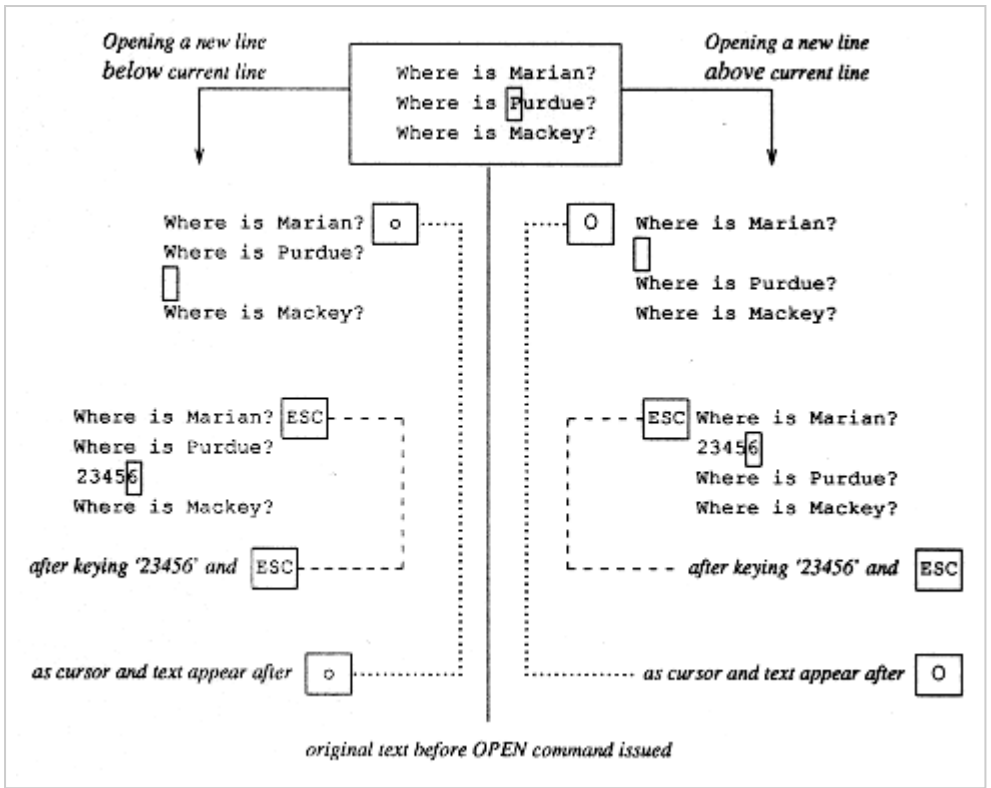
()

() ()

7.3 Open Command (o, O) ()

The open command is used to create a blank line in the file where additional text is to be typed in.The lower-case "**o**"command opens a new line *below* the line the cursor is onand the upper-case "**O**"command opens a new line *above* the line the cursor is on.By giving the open command,*text input mode* is activated and a blank line will appear on the screen, and the cursor will relocate to the first character space on this new line.You may now enter as much text as you want.To stop entering text you must press the "**ESC**" key.

The example below shows the use of the open commands.The left half of the example illustrates the lower-case "**o**"command opening a line below the current line, andthe right half illustrates the upper-case "**O**"command opening a line above the current line.



()

() ()

7.4 Read Command (:r) ()

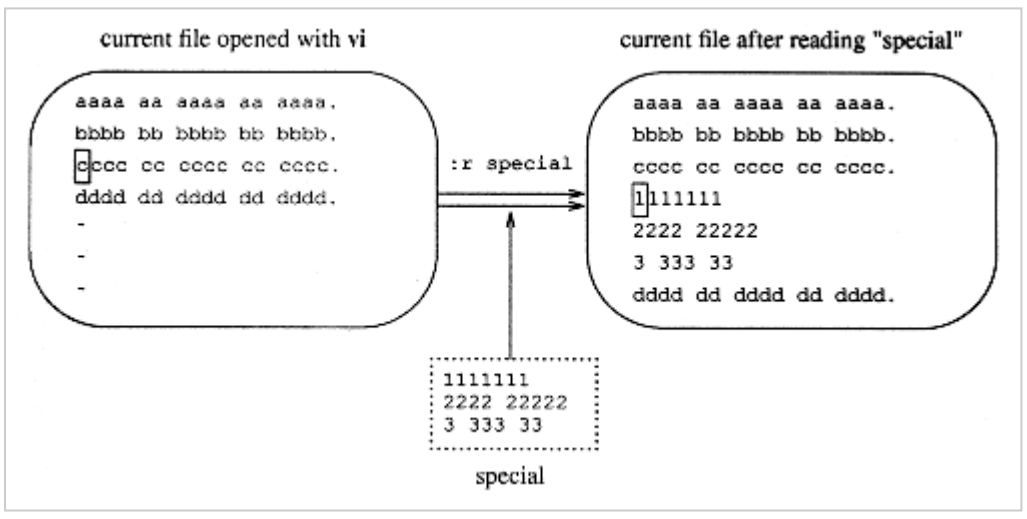
The read command is another method of placing information into a file.It is quick, easy and useful.After you have mastered the control of the **VI** editor,come back and reread this section and practice this command.It is too handy not to make use of it.

This command allows the user to place a copy of another file into the current file.For example, you might keep a special file containingthe heading required for all your lab assignmentsor a tricky, tough-to-type equation.In order to avoid retyping this information,all you do is tell the editor to "read" your special file into yourcurrent file, and like magic,this material from the special file becomes part of your current filewith no pain and no typos.

While in *command mode* andwith the cursor on the line above where you want the special fileread in, type:

```
:r filename
```

A copy of your special filewill appear directly below the line the cursor was sitting onand the cursor will reposition to the first character of the first linein the newly added text.Here is an example of the read command:



8.0 Correcting Text

Perfection is difficult to achieve;especially on the first try.It is probable that sooner or later (probably sooner) you willneed to change the material you have placed in a file.This section explains how to alter characters, words, and linesby combining the action of an operator (delete or change)with a scope to result in a more powerful command.

()

() ()

8.1 Delete Text (d, D) ()

The delete command is used in *command mode* to remove portions of text fromthe file being edited.The delete command is available for use in either the upper-case "**D**"format as a command, or in the lower-case "**d**"format as an operator.The upper-case **D**"command removes the text on the current line from the cursorto the end of the line.Most people tend to forget the "**D**"and instead use the "**d**" with the addition of scopes.

The lower-case "**d**"operator is very flexible because it can be used in conjunction with scopesto delete characters, words, and lines.The scope must be specified after the delete operator.Some of the most common scopes used with the delete operator shown in the next table.

Delete Operator & Scope	Resulting Action
dw	delete word forward
db	delete word backward
d$	delete from cursor to end of line (same as D)
dØ	delete from cursor to beginning of line
dL	delete from current line to end of screen
dG	delete from current line to end of file
d)	delete complete sentence forward
d(	delete complete sentence backwards
dd	delete complete line

It is important to remember that the current cursor position servesas the reference point for all of the scopes used with the delete operator.Notice that when the cursor is sitting in the middle of a word,the remainder of that word is counted as a scope unit.As with direction, arrow, or scope keys,the combination of "operator-scope" commands can be increasedby preceding them with a number.

Thus in the following example, if you type "**2dw**"with the cursor on the "f" of the word "finished", the result will be that the words "finished" and "this" are deleted.Notice, that if the cursor is sitting midpoint in a word,such as the "h" of "finished",only the remaining portion of that word is removed plus the next word, "this".

Original line	After 2dw command
To be finished this month:	To be month:
To be finished this month:	To be finismonth:

It will often be the case that you will want to delete whole lines.This can be accomplished by typing "**dd**".On most terminals when you delete a line,the editor will erase the line on the screen and replace it with an "@" character.This character symbolizes an empty line,much as the tilde (~) is used,and is not inserted into the text.This is done to save the cpu (central processing unit)the time necessary to repeatedly redraw the screen.If the "@" signs makes it difficult for you to read your file, type "**control r**"(some terminals require "**control l**")and the screen will be redrawn.For more information on redrawing the screensee the *Miscellaneous Information*section.As with any operator-scope combination,you can also delete more than one line at a time by preceding the"**dd**" command with a number.Typing "**2dd**" will delete two consecutive lines beginning with the current line.In the next example,if the cursor was sitting on the second line of a five line fileand the command "**2dd**" is issued, you would see two "@" symbols replace the deleted lines and these symbols would disappear when the screen was redrawn.

Original file	After 2dd issued	After file redrawn
To do this month:	To do this month:	To do this month:
a. see dentist	@	c. 15 page paper
b. buy gift	@	d. start project
c. 15 page paper	c. 15 page paper	
d. start project	d. start project	

()
() ()

8.2 Change Text (c, C) ()

The change command is used to replace portions of the text in the file being edited with new material that is keyed in.The change operator differs from the delete operator in thatthe *text input mode* is activated when issued.When you have completed keying in the new material to replace theidentified changed text, you are required to press the "**ESC**"key to return to *command mode*.
The "**c**"operator can be combined with the same scopes used with the delete operator,thus providing flexibility to change characters, words, and lines.Some of the most common scopes used with the change operator are:

Change Operator & Scope	Resulting Action
cw	change word forward
cb	change word backward
c$	change from cursor to end of line (same as C)
cØ	change from cursor to beginning of line
cL	change from current line to end of screen
cG	change from current line to end of file
c)	change from cursor to sentence start
c(	change from cursor to sentence end
cc	change complete line

It is important to remember that the current cursor position servesas the reference point for all of the scopes used with the change operator.Experimentation with scopes will show you that when the cursor is sitting in the middle of a word,the remainder of that word is counted as a scope unit.For example, if you use the command "**cw**"and the cursor is sitting midpoint in a word,only the remaining portion of that word is changed; while if the cursor is sitting at the beginning of a word,the entire word is changed.Careful positioning of the cursor permits you to change a prefix and suffixwithout altering the rest of the word.Any of the operator-scope commands can be repeated by precedingthem with a number.Typing "**2cw**"will replace the next two words with whatever is keyed in before the"**ESC**" is pressed.

To help you visualize the extent of the change that is going to take place,**VI** places a marker at the termination point of the forthcoming change command.This marker is a "$" and it overwrites the last character that is to be changed.Then as you type in the replacement text,the text on that line will begin to be overwritten.If the replacement text has fewer characters than the material being replaced,then the remaining old text between the final replacement character and the "$" maker will disappear when the "**ESC**"key is depressed.If the replacement text has more characters than the old text,then the new material will seem to overwrite the old text beyond the "$" marker.This is only temporary,and when the "**ESC**" key is pressed all the characters that were originally beyond the "$" marker will reappear.

Original Line	2cw Issued	New Text	After <ESC>
write the resume	write th$ resume	make	make resume
write the resume	write th$ resume	draft new	wrdraft new resume
write the resume	write th$ resume	ing sample	writing sample resume

This can be done with the "**cc**". command.Immediately after giving the line change command,the editor will erase the current line and leave it blank andawait replacement text.You may then proceed to key in a small amount of text or many paragraphs.You can change multiple text lines by preceding the"**cc**" command with a number.Thus typing "**9cc**"will change nine consecutive lines beginning with the current line.When you specify more than one line to be changed,the editor will immediately delete all the specified lines.On the screen the first line will be blanked with the cursor sitting on the first character space of that lineand all remaining lines to be replaced with "@" symbols.

Examine the sample text below where two lines are being changed with"**2cc**".

```
Original File      The cursor is on the "y" of "b. buy gift".After 2cc          The lines "1b.
buy gift" and "c. 15 page paper" are deleted.                    The cursor moves to the first cha
racter space on first removed line;                   "@" symbol indicates where the additional l
ine was removed.Replacement Text   New material keyed in "b. select classes"; and  "ESC" key pres
sed File Redrawn       Screen redrawn;  "@" symbol disappears.
```

Original File	After 2cc Issued	Replacement Text	File Redrawn
To do this month:	To do this month:	To do this month:	To do this month:
a. see dentist	a. see dentist	a. see dentist	a. see dentist
b. buy gift		b. select classes	b. select classes
c. 15 page paper	@	@	d. start project
d. start project	d. start project	d. start project	

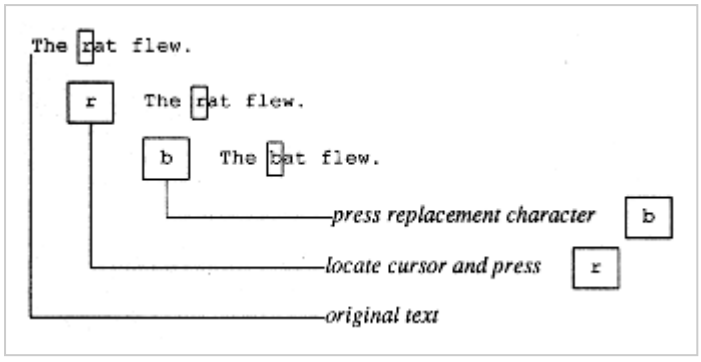
The upper-case "**C**"command changes the text on the current line from the cursor to the endof the line as does the command "**c$**".Many users disregard the "**C**"command and only remember the the lower-case "**c$**"command using the same format as previously learned for delete.

()

() ()

8.3 Replace Command (r, R) ()

The replace command is used to replace portions of text on the screenwith new characters in an overlay fashion.The lower-case "**r**" command replaces the single character with the new character you key in.To accomplish this exchange,move the cursor so it sits upon the character to be replaced,press the "**r**" key,then press the key you want to see on the screen.**VI** will make the exchange.The editor knows when you use the "**r**"key that only one character will be replaced;therefore,you remain in *command mode* throughout the action.



The upper-case "**R**"command replaces characters on the screen one at a time withcharacters you type in.Once you type "**R**",*text input mode* is activated permitting you to enter as muchtext as you want.It is important to remember that for each character you key inone is removed from the file until the end on the current line is reached;thereafter, any additional text will be inserted between the currentline and the line immediately below it.This way you can actually replace the end of one line with several linesof new text.To stop removing and replacing text you must press the "**ESC**" key.

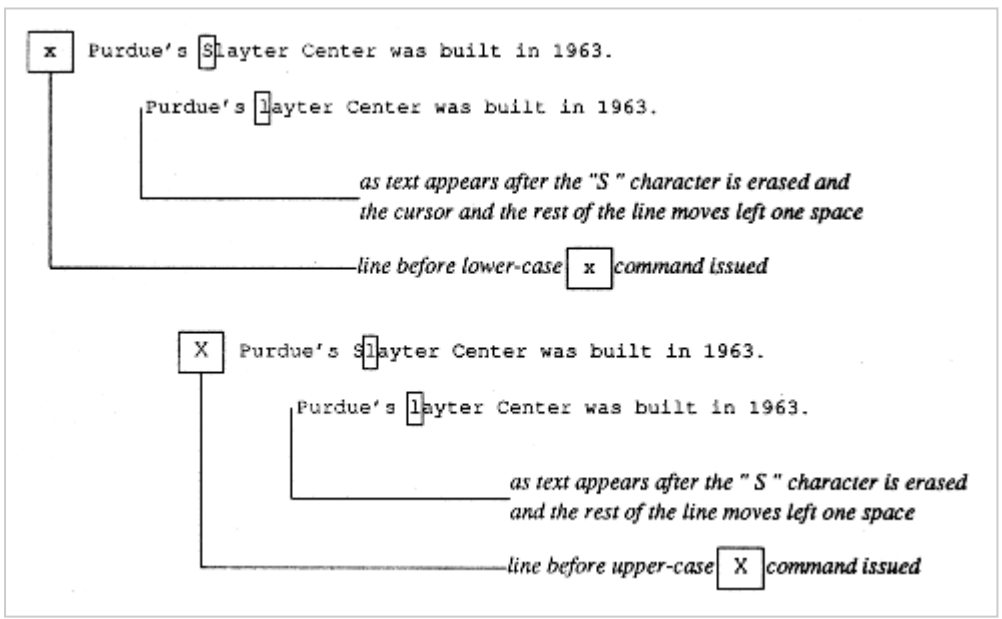
If you were to edit the next example using the "**R**"to replace the word " `classes` " with " `program` " followed by a"**RETURN**" and then continue keying in the extra line " `c. do speech` " and "**ESC**", the screen would look like the next table.

Original file	During Replacement	After <ESC> Pressed
To do this month:	To do this month:	To do this month:
a. see dentist	a. see dentist	a. see dentist
b. select classes	b. select program	b. select program
c. start project	c. do speechject	c. do speech
d. order tickets	d. order tickets	c. start project
		d. order tickets

()
() ()

8.4 Erase Command (x, X) ()

The erase command (also known locally as the "gobble" command) makes a character vanish.The upper-case "**X**"will cause the character to the left of the cursor to disappear.At the same time,the cursor and the remainder of the line move one character space to the left.The lower-case "**x**"command will erase or "eat" the character the cursor is sitting on.As with many **VI** commands,users will frequently elect to remember only one option.With the erase command it is usually the lower-case "**x**",because the cursor is highlighting the character that will disappear; plus, the lower-case "**x**"is easier to type because it does not require using the shift key.



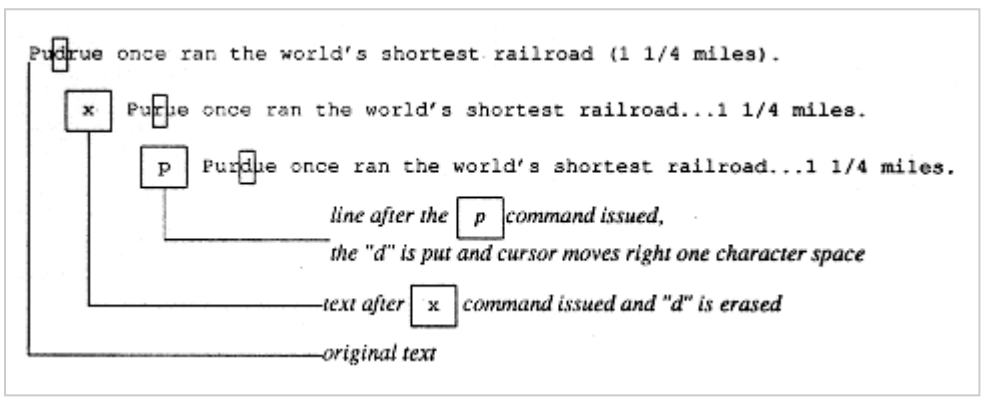
The erase command is wonderful,but unfortunately, it is easy to get into trouble using it.What happens is a user places the cursor on a linewhere a series of characters is to be removed andthen proceeds to eliminate the unwanted charactersby *constantly* holding down the "**x**".This is logical but, unbeknownst to the user, instructions to erase are being speedily sent (for example, ten instructions per second) to the editor while the editor erases

characters on the screen at a slower pace (maybe five instructions per second).The user then sees the intended last character disappear andthe erase key is released.Of course,the editor by now has a backlog of erase commandsand to the consternation of the user proceeds to "gobble" across the line.To avoid this problem,tap the erase key rather than holding it down.Oh yes,the good news is that "gobble" will only "eat" one screen width of information.If ever this happens to you,you will be very grateful for the last command in this section the "**U**" (undo).
()
() ()

8.5 Transposition Command (xp) ()

Almost every UNIX users will have days, sooner or later, where for some reason their "brain-hand" coordination short circuits and they spell familiar words in new and unique ways.Words like "the" appear on the screen "teh" and "their" becomes "thier".Transpositions occur when our fingers mix up the proper order of letterswithin a word even when the correct spelling is known.Correcting this common error is easy with the "**xp**" (transposition) command.

To use "**xp**",locate the cursor on the leftmost letter of the two letter combination,press "**x**" to erase the left character.As this action is being completed,the remainder of the line will move one character space to the left.The cursor will now be sitting on the character that had been to the rightof the erased character.Pressing the "**p**"will cause the deleted character to be put to the right of the cursor.Like magic without referencing either of the effected characters,simply moving the cursor to the proper location and typing the"**xp**" combination results in the two characters reversing order.



()
() ()

8.6 Undo Command (u, U) ()

The undo command is used to reverse or undo the effects of an already issued command that has changed the buffer.The lower-case "**u**"(undo) command undoes the effects of the last command (only one) that you issued.

After you make a number of changes to a single line,you may decide that you would rather have the original stateof the line back.The upper-case "**U**"undoes all of the changes that you have made to the current line.However, it does not remember changes you might have made to that line on previous visits.For example,if you make changes to line 12, move elsewhere, then return to line 12and make more changes, the "**U**"will only undo those changes made on the current visit to the line.

8.7 Replacing all ^M 's in a file

To replace the ^M 's on the screen in vi type

:1, $ s/\r/

What that says is: for lines 1 to end, replace \r with nothing . This will replace only one instance of ^M per line. If you have two on a line for some reason simply run the command a second time.

8.8 Global search and Replace

For a general global replace command use:

:%s/oldText/newText/g

This will replace all instances of 'oldText' with 'newText' throughout the file.

9.0 Rearranging and Duplicating Text

The **VI** editor allows the user to alter a file by way ofcopying or deleting a section of material to be inserted at another location.The user may select to complete the entire rearrangement with"yank and put" or "delete and put".In either action,this adjustment to the text is comprised of two separate command steps: the first part involves the original location and the second part deals with where the material is to be placed.**Warning:** yank & put and delete & put must be used in tandem.Inserting some other command action between the two steps,like correcting a typo,may cause the editor to become confused and produce unexpected results.A named buffer is another method to move or duplicate text.The advantage to using named buffers is that the delete or yank step can be done now and the put step handled later in the editing session.
()
() ()

9.1 Copying Text and Moving the Copy ()

In order to duplicate a section of text,you must first position the cursor at the material you want to copy.Next make a copy of the desired text by using the yank command.This places the copied text into a temporary buffer.Then immediately relocate the cursor at the point where you want thecopied text placed and instruct **VI** to put the copy.

Step 1: Copying Text with the Yank Command (y, Y)

The yank command is used to make copies of words, lines, and sectionsof text being edited and place theminto the unnamed buffer associated with the editor.The yank command is available for use in either the lower-case"**y**"format as an operator, or the upper-case "**Y**" format as a command.The lower-case "**y**"can be thought of as a yank operator which will combine withscopes to make an operator-scope command muchin the same way as the delete and change operators.The scope to be yanked must be specified after the "**y**" command.Some of the most common scopes used with "**y**"are summarized next:

Yank Operator & Scope	Resulting Action
yw	yank word forward
yb	yank word backward
y$	yank from cursor to end of line (same as Y)
yØ	yank from cursor to beginning of line
yL	yank from current line to end of screen
yG	yank from current line to end of file
y)	yank from cursor to start of sentence
y(	yank from cursor to end of sentence
yy	yank complete line

It is important to remember,as in previous examples of operator-scope commands,that the cursor serves as the reference point and the requested actionis initiated from the cursor backwards or forwards.Again the editor will permit you to address multiple scopes when usingthe yank operator;for example, the command "**3yw**"copies the three words to the right of the cursor.

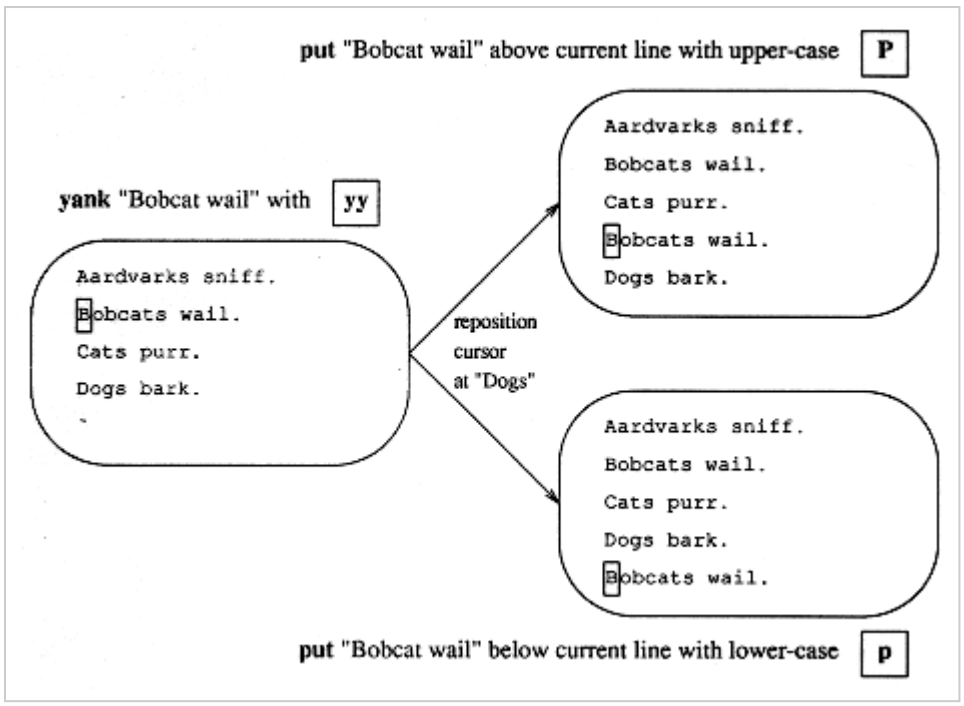
The function of the upper-case "**Y**"command is to copy whole lines of text into the unnamed buffer;however, the lower-case "**y**"operator can also be used to yank a whole line by typing it twice, "**yy**". You can yank more than one line by preceding either"**Y**" or "**yy**"with a number.Typing "**6Y**" or "**6yy**"copies the current line and the next 5 lines into the unnamed buffer.

Step 2: Relocating the Copy with the Put Command (p, P)

The put command is used to place the contents of the unnamed bufferback into the file being edited.Returning whole lines into the text is handled differentlythan word and sentence fragments.If the text contained in the unnamed buffer forms a line segmentor is a scope which partially spans more than one line,it will be placed within the current line after the cursor if you use lower-case "**p**",or before the cursor if you use upper-case "**P**".On the other hand,whole lines are returned to the file from the unnamed bufferwithout changing the current line.The lower-case "**p**"places the line or lines below the current line and the upper-case"**P**"places them above the current line.

A handy feature of yank & put is the ability to insertcopy repeatedly within the same file.The format for this action is yank, relocate cursor, put,relocate cursor, put, etc. until all needed copies have been placed.

See below for an example of copying a line and moving the copy.



()

() ()

9.2 Deleting Text and Moving It ()

In order to move a section of text from one location in a fileto another location,you must first position the cursor then delete the text to be movedinto a temporary buffer with the delete command.Immediately relocate the cursor to the point in the file where you wantto insert the previously deleted material and instruct **VI** to put the contents of the unnamed temporary buffer back into the file.

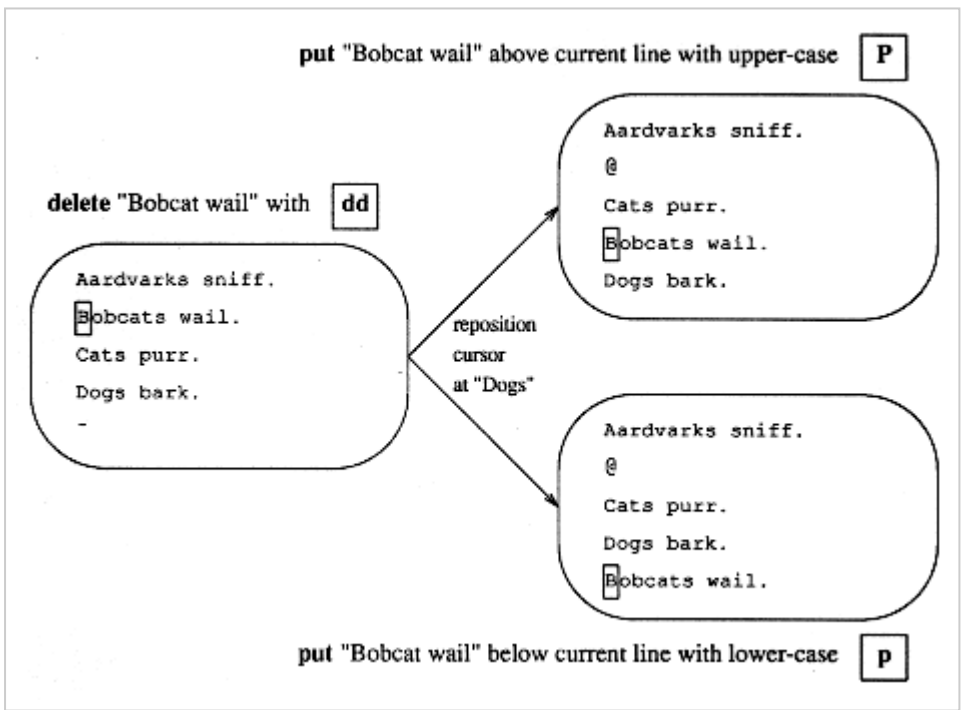
Step 1: Removing Text with the Delete Command (d, D)

Deleting a portion of text with the delete command does not simplycause the removed material to disappear into thin air; instead, the deleted text is placed in an unnamed editor buffer.Thus, all you have to do to move text within a file is delete itinto the unnamed buffer and put it back into the file where you want it.Since the delete command forms the first step in moving text,it is possible to move any portion of text which the delete commandcan delete, such as characters, words, and lines. A complete description of the delete command can be found in the *Correcting Text* section.

Step 2: Relocating the Deleted Text with the Put Command (p, P)

The put command works in the same manner with delete & put as it does with yank & put.Partial sentence fragments are placed after the cursor within the current line when the lower-case"**p**"is pressed and before the cursor when the upper-case"**P**"is used.Whole lines, on the other hand,are returned to the file from the unnamed buffer without changingthe current line.The lower-case"**p**"placing the line or lines below the current line and the upper-case"**P**"placing them above the current line.The cursor, while on the current line when the put command is issued,moves to the first character space on the first line of the freshlyplaced text.

Below is an illustration of how delete & put works when you deletea whole line and then put the contents of the buffer back into the file.



()

() ()

9.3 Named Buffers ()

Named buffers offer another way to copy (yank) or remove (delete) textand then reenter (put) it into the file.Usually this is a feature used by more advanced users.Every time you open a file with the **VI** editor,26 named buffers (a-z) are created for your use.The advantage of using named buffers to duplicate and rearrange textover the previous discussed yank & put and delete & put method is that you have theluxury of copying or removing now and resubmitting the text later in the editing session.Since the unnamed buffer associated with the editor only saves the last deleted or yanked text,you must put its contents back into the file when moving or copyingbefore performing some other editor command.The named buffer permits access to its contents at anytime during editing session.**Warning:** *VI's* 26 named buffers remain only for the life of the current editing session.If you **do not** use the material placed in a named buffer during the same session it is filled, the material is lost.

To use a named buffer with the yank and delete commands,you must first locate the cursor in the textat the point where the material to be placed in the buffer is situated.The **VI** editor is informed you intend to use a named buffer whenthe double quote (")is used followed by the name of the buffer (a-z)and then the command you want carried out.Thus typing:

```
    "g7yy           "g7dd
```

would tell the editor you are:

```
       "            calling forth a named buffer       g            its name is  gor      7yy
yanking 7 lines       7dd          deleting 7 lines
```

Anytime during a session you wish to append more information into anamed buffer,it is done by recalling the buffer with a capital letter for a name.To append information to the bottom of the buffer used above,you would type:

```
    "G3yy           "G3dd
```

Warning: These named buffers are not write protected.If a named buffer contains information and it is called a secondtime with its lower-case name,the original material is over-written.

Later when you are ready to make use of the content of the named buffer,you would type:

```
    "gp             "gP
```

telling the editor you are:

```
    "      calling forth a named buffer     g      its name is  gor   p      putting the content
s below the current line     P      putting the contents above the current line
```

The "putting" action of a named buffer may be exercised again and again during an editing sessionand a copy of its contents put into the file as often as needed.Different users select named buffers in different ways.Some users select a particular buffer because of the material to beplaced in it (z buffer for zebras, m buffer for money).Other users only use favorite buffers (like x, y, z or their initials).Whatever method you devise for buffer selection,you must remember its name.The editor provides no easy way to find out which buffer has which textsegment.For the user who gets totally mucked-up and can not remember in which bufferthat necessary file section was hidden,a possible (but painful) solution is to go to the end of the fileand begin emptying buffers one-by-one until the mystery buffer is rediscovered.

The procedure to accomplish this is:

1. go to the last line of the file
2. empty a buffer with " **buffer name p**
3. check buffer contents
4. remove dumped buffer contents with "**u**" (undo)
5. if not mystery buffer, precede to step 2 and repeat with next named buffer

---

10.1 Miscellaneous Information

The following section is a collection of information about the **VI**editor that you should know about for successful editing.Read this section,apply what is useful now,and tuck the rest away to return to laterafter you have used the editor for a while.
()
() ()

10.1 Creating Line Numbers ()

The developers of the **VI** editor designed **VI** to be used withoutline numbers,feeling that the average user would locate and manipulatetext by content rather than by line numbers.Additionally,because line numbers are constantly changing due to insertions and deletions,adjusting what you "*think*" is in a particular line can be dangerous.

The **VI** editor allows you to view line numbers in your file four ways:determining your current line number,a quick glance at line numbers for the complete file,insertion of line numbers for the current editing session only, and inclusion of line numbers for every session.

To determine the line number for the line the cursor is on, type"**control g**" as discussed in the *Positioning Text on the Screen* section.

To have a quick one time glance at line numbers,while you are in the file, type:

```
:%nu
```

This will cause line numbers to be assigned to all lines;unfortunately, this also causes the complete file to scroll past quicklystopping on the last screenful of text.Now if you are working with a short 20-line file, this is great;however, if your file contains 200 lines,you will see the first 180 lines whiz past.To stop the scrolling action,you must press the "**control s**"to "stop" screen movement and later "**control q**"to "quit" the frozen screen.Some people get very good at timing the "**control s**"in order to get the exact point of the file to stop on the screen,most do not.

Many users find it more convenient to have numbers added to a filefor the current editing session;knowing that the next time the editor is invoked,the numbers will not appear.To have line numbers inserted for the current session, type:

```
:set number
```

Immediately you will see the line numbers appear in your file and they will remain until you exit the editor or type:

```
:set nonu
```

The long-term line numbering option available with **VI** is to place a line numbering command in the *.exrc* file located in your HOME directory.The *.exrc* file is your personal control file to instruct all facesof the editor (**VI**, **EX**, and **EDIT**) on how you want it to perform when invoked.Not everyone creates a *.exrc* file.For those without this file,the automatic defaults of the editor are imposed, such as no line numbers.If you place the command "**set number**" in your *.exrc* file,the next time you invoke **VI** the editor will checkthis file for instructions andwill present the file to you with line numbers. Later, if you decide you prefer to forgo line numbers,open your *.exrc* file and delete the "set number" line.

Type the followingto get automatic line numbering each and every time you use the **VI**editor on all files in your *.exrc*:

```
%  cd     %  echo  'set number'  >>  .exrc     %
```

()
() ()

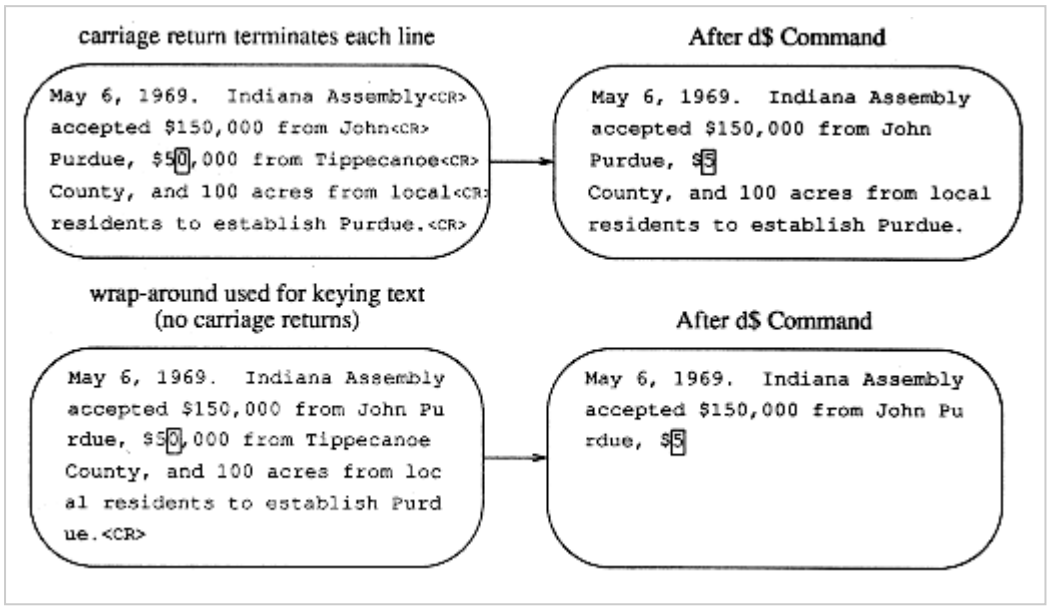
10.2 Lines and Sentences in VI ()

To be successful in your editing,it is necessary to understand what the editor considers a lineand a sentence.Just for clarity,a line and a sentence are different animals to the editor.To the editor, a line begins on the left of a screenand terminates at a carriage return .The carriage return is the invisible character placed in your file every

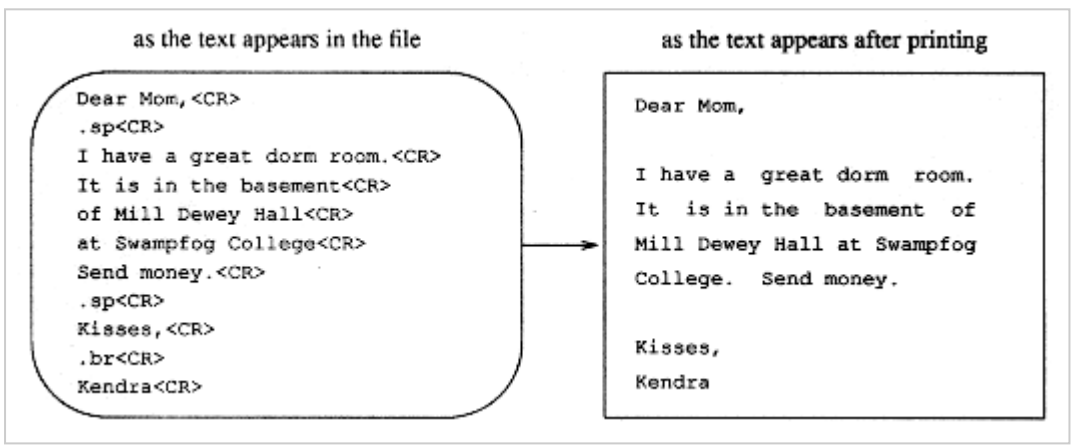
time you press the "**RETURN**"key.A sentence to the editor is a string of characters of unspecified length(a few characters to many lines)terminating with the punctuation marks " . ", " ? ", " ! " followed by either a carriage return or two blank spaces.

Technical typists, secretaries, and students who produce lotsof reports and papers find that editing is much easier to completeif as you are keying in text you make lines very short.The breaking up of sentences into many lines is helpful.Placing a carriage return after phrases and punctuationwill make editing words and lines less of a problem.

Some people like wraparound typing (straight typing without inserting carriage returns), **this is not recommended**.The next example demonstrates the use of the delete operator withthe end-of-line scope "**d$**"on text where the "**RETURN**"key follows phrases and punctuation versus using the same operator-scopecommand on wraparound typing.The results are drastically different.



When you print out your file using a text processor,the computer will connect lines and phrases plus insert spacing between sentences to make your material look presentable.For example:

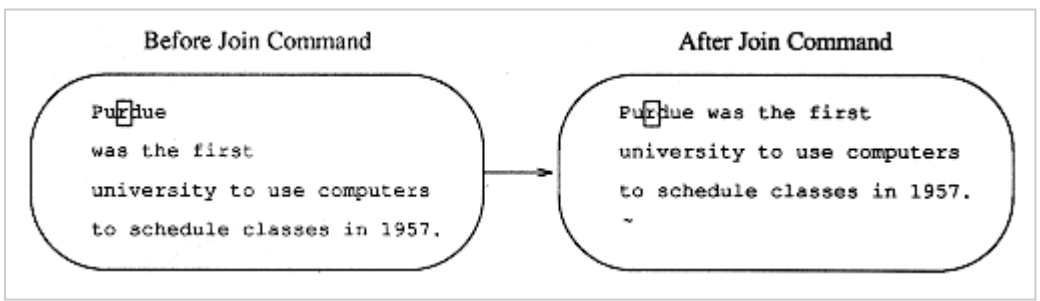


()

() ()

10.3 Joining Lines ()

As you are editing files,you will find it is desirable to combine or join lines.This is easily done using the "**J**" (join) command.An illustration of joining lines is given below.The cursor is located on the top line when the "**J**"command is issued.**VI** will move the lower line and butt it to the end of the upper line.The editor takes care of necessary spacing for you.



()

() ()

10.4 Redrawing the Screen ()

The **VI** editor requires cpu (central processing unit) time to function.Each command and action takes a minute amount of time.This interaction is not a problem if only one or two people are accessing the system.However,the Engineering Computer Network has thousands of usersand at any given moment many hundreds of users may be editing, running programs,and other system jobs,all competing for cpu time.

The **VI** designers foresaw that the editor could be a "time hog"and decided that one way to reduce some of the editing interactionwith the cpu was to minimize redrawing the screen.This is why when you are in the *text input mode*you do not see the screen being updated until the action is completedand you return to *command mode*.

With time conservation as the impetus,the screen is also not redrawn each time a line is deleted.Rather, a removed line is replaced with the "@" symbol to symbolize an empty line,much as the tilde (~) is used.Further editing of the text that remains on the screen is still possible.These "@" symbols remain viewable as long as editing continues on this screeneven though they are never inserted into the text.

Sometimes these "@" symbols are distracting and annoying.If a user would prefer not to see the "@" symbols, the screen may be redrawn and the "@" symbols eliminated by issuing the command"**control r**" or "**control l**".Experiment to see which command works on the terminal you are using.

()
() ()

10.5 Accessing the EX Editor ()

The **VI** editor has a powerful and useful companion editor, **EX**.In reality **VI** and **EX** are different faces of the same editor;both were developed from and use the same program base.**VI** is a screen oriented editor,while **EX** is a line oriented editor.Since both of these editors are built upon the same base,when one editor face is installed on a computer by default the other editor face is also there.This offers great advantages to the user because each face possesses individual strengths.Easy to use commands allow you to transfer from one face to another in order to increase your editing powerby permitting you to use the desirable features of both editorsand thereby better meet your editing requirements.The "global substitute" and "text marking" are two favorite**EX** commands.

All **VI** users, frequently without knowing it, make use of the dual face capability of this editor. For example, "**:w**", "**:wq**", or the "**:quit!**"are in reality **EX** commands.Also, the "read" command discussed in the *Text Insertion* section is an **EX** command!During an editing session,anytime you wish to invoke a single **EX** command,you must first make sure you are in *command mode* then type "**:**"followed by the command.The cursor will hop to the bottom of the screen and the command is echoed.When you press "**RETURN**",the **EX** command is executed and you are brought back to the *command mode* of **VI**.

Sometimes it is useful to issue a series of **EX** commands.This is done by typing "**Q**"while in *command mode* and you will enter and remain in the**EX** face until you type "**vi**"at the **EX** prompt (:),such as "**:vi**". Immediately you are returned to **VI** and can continuethe editing session using **VI** commands.

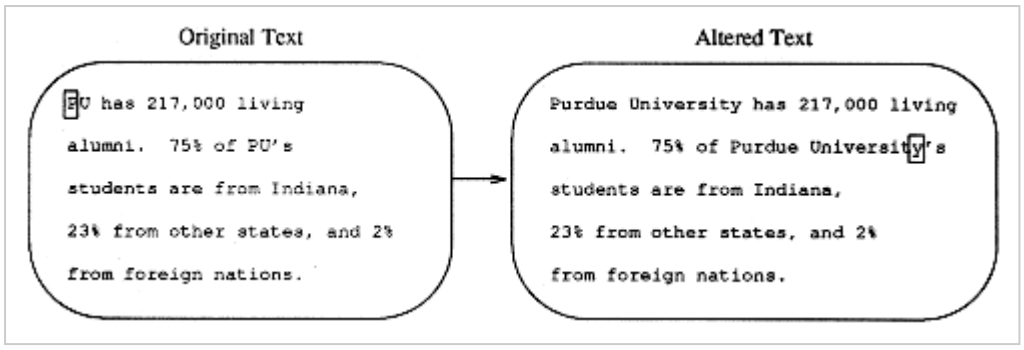
It is suggested that users do not try to learn both **VI**and **EX** at the same time.Learn **VI** well,then proceed to learn **EX**.Attempting to learn both faces of the editor at the same timeleads to frustration and confusion,as invariably the user mixes up which command to use when.

()
() ()

10.6 Repeating a Command ()

To make life a bit easier,**VI** allows text alteration commands to be repeated by using the"**.**" (repeat) command.A handy way to illustrate the repeat command is with the "**cw**"command replacing a single word with two new words throughout a paragraph.

In this example,the first occurrence of "PU" is located with the search command**PU**".Then with the cursor on the "P" of "PU", the "**cw**"command is issued followed with "Purdue University" and the"**ESC**".The "**n**"key is pressed to find the next occurrence of "PU".The cursor relocates on the "P" of the next "PU" and all that is requiredto change it to "Purdue University" is to type "**.**"



()
() ()

10.7 Temporarily Interrupting VI ()

The ability to access the UNIX shell while keeping the current file openwith **VI** is a true convenience.To do this,you should first "write" the current buffer contents to the disk filewith "**:w**".The current editing session may be interrupted by then typing"**:!**"followed by the desired command.When the requested command action is completed,the message:

```
    [Hit return to continue]
```

will appear at the bottom of the screen.After pressing "**RETURN**",you will be returned to the **VI** editor to the same location youwere at when you temporarily interrupted the editing session.
A common way to use this interrupt ability during a current editingsession is to read a recently received mail message, "**:! mail**".As you work more with UNIX,you will begin to see many ways to make this command work for you.

()
() ()

10.8 Editing Multiple Files Using VI ()

The **VI** editor provides an advanced featurewhich allows a user to invoke the editorand then edit multiple files by use of the "**:e**"(edit) command.This ability to access multiple files without leaving the editor permits a userto look up information in another file without exiting the editor.Additionally, because files are opened within the same editor invocationthey can share the same named buffers,thereby making the transfer of text possible between the files.The example on page 40 demonstrates how two linescan be "yanked" from the file *oranges*,placed into a named buffer "k",and then "put" into the file *apples*.

When **VI** is invoked,a work area called a buffer is created for editing purposes.It is into this work space that a copy of a specified disk file is placed.The editor permits only one file copy in this buffer space at a time.Thus after making changes to a file (delete, add, or change),you must inform the editor what you wish done to the current buffer contentsbefore you will be permitted to bring another file into this space.You do this by use of the "**:w**"(write current buffer contents to opened file),"**:e!\ newfile**"(toss current buffer contents, no update to opened file,and place a copy of newly called file in buffer), or"**:quit!**"(exit editor and toss buffer and buffer contents).The editor is smart enough to know thatif all you have done is copy or read from a file,it can dispose of the unneeded buffer copy without further instructionsfrom you when a new file is called.

When you have two files open,**VI** permits toggling between files by use of"**:e\ #**".This works because whenever **VI** sees the character "#"used in a command where a filename is expected,it substitutes the "#" with the name of the previous file.For example if you had been in *apples*then opened *oranges*,the command "**:e #**"would return you to where you were in the *apples* file.Repeat "**:e #**"and you would be back in *oranges*.

Another method to "cut" and "paste" between files is to use the *mark*command (m) in conjunction with named buffers (").Move the cursor to the first line you wish to copy from *oranges*and type "**ma**"(*m* to mark the line and *a* to specify which mark).Next move the cursor to the last line you wish to copy and type""**zy'a**".This command tells **VI** to open up "z (buffer named z)and *y* (yank) a copy of all the lines from the line marked '*a* to the current lineand place them in the aforementioned buffer.Return to *apples* with "**:e #**"and proceed to paste the contents of "z by moving to the desired location and*p* (put) the buffer contents by typing ""**zp**".

1. Open the original file with "**vi apples**".

2. Correct typo.

3. Write the contents of the buffer to the file *apples* using the"**:w**" command.

4. Issue the command "**:e oranges**"to open the second file.
5. Yank a copy of two lines from the file *oranges* and put them intothe named buffer "k" using the ""**k2yy**" command.
6. Recall the original file, *apples*, to the buffer using the"**:e apples**" or "**:e #**" command.
7. Contents of buffer discarded when *apples* is recalled to the buffer.The disk copy of *oranges* remains unaltered.
8. Position the cursor and put the lines from the buffer"k" into *apples* using the ""**kp**" command.
9. Write and quit the editor with the"**:wq**" command.
10. Buffer is discarded upon leaving the editor.

Operators and Scopes

Delete Operator & Scope	Resulting Action
dw	delete word forward
db	delete word backward
d$	delete from cursor to end of line (same as D)
dØ	delete from cursor to beginning of line
dL	delete from current line to end of screen
dG	delete from current line to end of file
d)	delete complete sentence forward
d(	delete complete sentence backwards
dd	delete complete line

Change Operator & Scope	Resulting Action
cw	change word forward
cb	change word backward
c$	change from cursor to end of line (same as C)
cØ	change from cursor to beginning of line
cL	change from current line to end of screen
cG	change from current line to end of file
c)	change from cursor to sentence start
c(	change from cursor to sentence end
cc	change complete line

Yank Operator & Scope	Resulting Action
yw	yank word forward
yb	yank word backward
y$	yank from cursor to end of line (same as Y)
yØ	yank from cursor to beginning of line
yL	yank from current line to end of screen
yG	yank from current line to end of file
y)	yank from cursor to start of sentence
y(	yank from cursor to end of sentence
yy	yank complete line

()