The vi Editor

To start vi

The basic syntax is: vi [*file*…]

Command Mode and Input Mode

Editing Buffer:	When you work with vi, all the data is kept in the editing buffer. It means that when you use vi to edit an existing file, you are not working with the actual file.
Command Mode:	The characters you type are interpreted as commands.
Input Mode:	Everything you type is inserted into the editing buffer.

As you work with vi, you frequently change back and forth between command mode and input mode. In command mode, there are a number of commands that can be used to change to input mode.

When you are in input mode, you press **`<Esc>`** key to change to command mode.

Some versions of vi have an option named showmode. Once you set this option (*:set showmode*), vi will display a short message at the bottom right-hand corner of the screen when you are in input mode.

How do you add some data to the middle of a file?

1. When you start vi, you are automatically in a command mode.
2. The cursor shows your current position in the editing buffer.
3. You move the cursor to the place where you want to add the data. (There are 40 different “simple” commands just to move the cursor.)
4. You type one of the 12 different “simple” commands to change to input mode.
5. You start typing. Everything you type is inserted into the editing buffer.
6. When you finish typing, you press <Esc> key to change back to command mode.
7. Save your file and quit from vi.

Starting vi as a Read-Only Editor:

vi –R [*file…*] (R: read-only)

or

view [*file…*]

Recovering Data After a System Failure:

vi –r *filename* (r: recover)

Stop vi

`ZZ`	Save your work, then stop.
`:q!`	Quit without saving (You need to press <Return>).

Redisplay

`^L`	Redisplays everything.

Enter control characters

`^V` followed by the control character you want to enter.

Examples:

`^V^C`	results a **`^C`**
`^V^V`	results a **`^V`**
`^V^G`	results a **`^G`**

Moving the Cursor

Move the cursor by one position

`h`	Move cursor one position left
`j`	Move cursor one position down
`k`	Move cursor one position up
`l`	Move cursor one position right

The above four keys are easy to press with you right hand. The following keys are the alternatives that are easier to remember:

←	Move cursor one position left
↓	Move cursor one position down
↑	Move cursor one position up
→	Move cursor one position right
`<Backspace>`	Move cursor one position left
`<Space>`	Move cursor one position right

Move the cursor within a line

`-`	Move cursor to beginning of previous line
`+`	Move cursor to beginning of next line
`<Return>`	Move cursor to beginning of next line
`0`	Move cursor to beginning of current line
`$`	Move cursor to end of current line

Move the cursor by a word

`w`	Move cursor forward to first character of next word
`e`	Move cursor forward to last character of next word
`b`	Move cursor backward to first character of previous word
`W`	Same as `w`; ignore punctuation
`E`	Same as `e`; ignore punctuation
`B`	Same as `b`; ignore punctuation

Move the cursor in a larger range

`)`	Move forward to beginning of next sentence
`(`	Move backward to beginning of previous sentence
`}`	Move forward to beginning of next paragraph
`{`	Move backward to beginning of previous paragraph
`H` (high)	Move cursor to top line
`M` (middle)	Move cursor to middle line
`L` (low)	Move cursor to last line

The art of moving the cursor is to get where you want in as few keystrokes as possible.

Repeat Count

Whenever it makes sense, you can have vi automatically repeat a cursor movement command by typing a number, repeat count, before the command.

Examples:

`20w`	Move forward 20 words
`30j`	Move down 30 lines
`30+`	Same as `30j`

As a general rule, you can repeat any vi command, not just cursor commands, by typing a number in front of it, as long as doing so makes sense.

Moving Through the Editing Buffer

vi will display as much of the editing buffer as will fit on your screen.

To display different parts of the editing buffer:

`^F`	Move down (forward) one screen
`^B`	Move up (back) one screen
`^D`	Move down a half screen
`^U`	Move up a half screen

Examples:

`10^F`	Move down 10 screens
`7^B`	Move up 7 screens

However, when you type a number in front of ^D or ^U, it is used for something different: it sets the number of lines that either of these commands should jump.

Example:

10^D Will jump 10 lines and tell vi that all subsequent **^D** and **^U** commands should also jump 10 lines until you reset it.

Searching for a Pattern

You also can move around the editing buffer by jumping to a line that contains a particular pattern.

/*pattern*<Return>

vi will search forward for the next occurrence of the ***pattern***.

?*pattern*<Return>

vi will search backward for the previous occurrence of the ***pattern***.

n vi will repeat the last **/** or **?** command in the same direction.

N vi will repeat the last **/** or **?** command in the opposite direction.

In fact, ***pattern*** is not just a string, it can be a “Regular Expression”.

Special Characters to Use in Regular Expressions

`.`	Match any single character except newline
`*`	Match zero or more of the preceding characters
`^`	Match the beginning of a line
`$`	Match the end of a line
`\<`	Match the beginning of a word
`\>`	Match the end of a word
`[ ]`	Match one of the enclosed characters
`[^ ]`	Match any character that is not enclosed
`\`	Take the following symbol literally

Examples of using regular expression:

`/t.t.l\>`	Will search for next 5-letter word whose first and third letter is t and the last letter is l (i.e., total).
`/th[ae]`	Will search for the next word whose first letter is t, the second letter is h, and the third letter is either a or e (i.e., the, that, these, …).
`?^The\>`	Will search for the previous line that starts with the word The.

Using Line Numbers

Internally, vi keeps track of each line in the editing buffer by assigning it a line number. You can either see these numbers by entering

```
:set number
```

or get rid of these numbers by entering

```
:set nonumber
```

There are two important uses for line numbers:

1. You can use them with many of the ex commands.
2. You can use G (goto) command to jump to a specific line.

   Examples:

`50G`	Jump to line 50
`1G`	Jump to the beginning of the editing buffer
`G`	Jump to the end of the editing buffer

Inserting Data into the Editing Buffer

Commands for entering new data:

`i`	Change to input mode: inset before cursor position
`a`	Change to input mode: inset after cursor position
`I`	Change to input mode: inset at start of current line
`A`	Change to input mode: inset at end of current line
`o`	Change to input mode: open below current line
`O`	Change to input mode: open above current line

`i`: insert `a`: append `o`: open

Making Changes to the Editing Buffer

vi Replacement Commands:

`r`	Replace one character
`R`	Replace by typing over
`s`	Replace one character by insertion
`C`	Replace from cursor to the end of line

`cc`	Replace entire current line by insertion
`S`	Replace entire current line by insertion
`c`*move*	Replace from cursor to *move* by insertion

Example 1: This is a test.

Move the cursor to a and type **`s`**, you will see This is $ test.

Type not a<Esc>, you will get This is not a test.

Example 2: This is a bad example.

Move the cursor to b and type **`cw`**, you will see

This is a <u>b</u>a$ example.

Type very good<Esc>, you will get

This is a very good example.

Example 3: This is not a bad example.

Move the cursor to n and type **`c3w`**, you will see

This is <u>n</u>ot a ba$ example.

Type a very good<Esc>, you will get

This is a very good example.

Replacing a Pattern

If you want to replace a particular pattern with something else, you can use the ex command named :s (substitute):

`:s/`*`pattern`*`/`*`replace`*`/`	substitute, current line
`:`*`lines`*`/`*`pattern`*`/`*`replace`*`/`	substitute, specified line
`:`*`line`*`,`*`lines`*`/`*`pattern`*`/`*`replace`*`/`	substitute, specified range
`:%s/`*`pattern`*`/`*`replace`*`/`	substitute, all lines

Examples:

```
:s/television/TV/
```

will replace the first "television" on the current line with "TV".

`:s/television/TV/g` **g**: global

will change all "television"s on the current line to "TV"s.

`:s/television/TV/c` **c**: confirm

vi will ask your permission before making the change.

```
:s/television//g
```

will remove all "television"s on the current line.

```
:57s/television/TV/
```

will change the first "television" to "TV" on line 57.

```
:57,60s/television/TV/cg
```

will change all "television"s to "TV"s between line 57
and line 60 with your permissions.

```
:.,$s/television/TV/g
```

will change all "television"s to "TV"s from the current line
to the end of the editing buffer.

`:1,$s/television/TV/g` and `:%s/television/TV/g`

will change all "television"s to "TV"s in the editing buffer.

Undo

`u`	Undo the last command that modified the editing buffer
`U`	Restore the current line

Example: If you entered `:%s/television/TV/g` and found that it was a mistake. You can undo it by typing a `u`.

Note:

1. `u` can undo itself.
2. `u` can undo `U`.
3. `U` only works as long as you stay on the line with the changes.

Repeating a Change: `.`

The `.` command will repeat the last insertion, substitution, change, or deletion.

For example, if you need to insert the name Krzyzak at several places in the editing buffer and want to find a way to make it easier. You can insert

Krzyzak`<Esc>`

at the first place. Then move to the place where you want to make the next insertion and type a `.`. The insertion will be repeated for you.

You can use the `.` command as many times as you want.

Changing the Case of a Letter: `~`

Controlling the Length of Lines

Breaking a Long Line: `r`

If you want to break a long line, you can move the cursor to the space following the last character you want to break, then type:

```
r<Return>
```

`r` will replace the space with a newline character.

Join Two Lines: `J`

If you want to join two lines, you can move the cursor to the first line and type `J`. vi will join that line and the next line into one line.

You can use the repeat number in front of `J` to join more than two lines into one line. However, `2J` will likely do the same thing as `J` does.

`:set wm` (you also can use `wrapmargin` to replace `wm`)

You can ask vi to break lines for you:

`:set wm=`*n*	Auto line break within n positions of the right margin

To turn off the automatic margin control:

```
:set wm=0
```

Deleting Data from the Editing Buffer

There are several ways to delete data from the editing buffer:

`x`	delete character at cursor
`X`	delete character to left of cursor
`D`	delete from cursor to end of line
`d`*move*	delete from cursor to *move*
`dd`	delete the entire current line
`:`*line*`d`	delete specified line
`:`*line*`,`*line*`d`	delete specified range

Examples:

`dw`	delete one word
`d8W`	delete 8 words (ignore punctuation)
`d5}`	delete 5 paragraphs
`dG`	delete from current line to end of editing buffer
`d1G`	delete from current line to beginning of editing buffer
`8dd`	delete 8 lines from the current line
`:50d`	delete line 50
`:50,60d`	delete lines 50 through 60
`:1,.d`	delete from the beginning of the editing buffer to the current line
`:.,$d`	delete from the current line to the end of the editing buffer
`:1,$d`	delete the entire editing buffer
`:%d`	delete the entire editing buffer

Copying the Last Deletion

vi always keeps a copy of the last thing that you deleted. You can copy this deletion to any place in the editing buffer by using the **`p`** and **`P`** commands.

`p`	Insert the last deletion after the current position of the cursor, or below the current line.
`P`	Insert the last deletion before the current position of the cursor, or above the current line.

Example:

This is a for line testing.

You move the cursor to the space before for and type **`deep`**, you will get

This is a line for testing.

`de`	delete the space and the following word
`e`	move forward to the end of the next word
`p`	insert the deletion after the cursor

This is a for line testing.	move cursor after a and type de
This is a line testing.	type e
This is a line testing.	type p
This is a line for testing.	

Other vi command combinations:

`xp`	transpose two characters
`ddp`	transpose two lines

Copying and Moving Lines

You can use the ex commands **:co** (copy) and **:m** (move) to copy or move lines.

The ex copy and move commands are:

:*line***co***target*	copy specified line; insert below target
:*line***,***line***co***target*	copy specified range; insert below target
:*line***m***target*	move specified line; insert below target
:*line***,***line***m***target*	move specified range; insert below target

Examples:

`:4co12`	copy line 4, insert below line 12
`:2,5co12`	copy lines 2 through 5, insert below line 12
`:4m12`	move line 4, insert below line 12
`:2,5m12`	move lines 2 through 5, insert below line 12
`:1,.m$`	move lines 1 through current line to bottom
`:.,$m0`	move current line through the last line to top

Entering Shell Commands

There are several ways to enter regular shell commands within vi.

:! *command* pause vi, execute *command*

Example:

```
:! date
[No write since last change]
Wed Jan 14 11:36:06 CST 2015

Press ENTER or type command to continue
```

`:!!`	Pause vi, execute the last shell command.

If you would like to enter a number of shell commands, you can start a new shell:

`:sh`	Pause vi and start a new copy of your login shell. Now, you can enter as many commands as you want. To exit from this shell, you can use either ^D or exit.

You also can type

`:!csh`	to start a C-Shell
`:!sh`	to start a Bourn shell
`:!ksh`	to start a Korn shell

Reading Data into the Editing Buffer :r

To read data from an existing file into the editing buffer, use the **`:r`** command:

`:`*line***`r`** *file*	insert contents of file after specified line
`:r` *file*	insert contents of file after current line

Examples:

`:10r` info	insert the contents of the file info after line 10
`:0r` info	insert the contents of info at the beginning of the editing buffer
`:$r` info	insert info to the end of the editing buffer
`:r` info	insert info after the current line

You also can insert the output of a shell command directly into the editing buffer by using **`:r`**

`:`*line*`r !`*command*	insert output of *command* after specified line
`:r !`*command*	insert output of *command* after current line

Example:

`:r !`uptime	insert of the output of uptime after the current

line

Note:

You can use **`u`** to undo the result of **`:r`** (or **`:r !`**command).

Using a Shell Command to Process Data

Using the `!` and `!!` commands, you can send lines from the editing buffer to a regular shell command. The output of the command will replace the original lines.

n`!!`*command*	execute *command* on n lines
`!`*move command*	execute *command* from cursor to *move*

Example: We have 5 lines in our editing buffer

```
of
time
brief
a
history
```

Then we move the cursor to the first line and type: **`5!!`** *sort*

The original 5 lines will be replaced by

a
brief
history
of
time

fmp is a command that can make your message look nice without changing the paragraph breaks.

Examples:

`10!!`fmp	format 10 lines (from the current line)
`!}`fmt	format all lines to the end of paragraph
`1G!G`fmt	move the cursor to the first line, then format the entire editing buffer

Writing Data to a File

`:w`	write data to original file
`:w` *file*	write data to specified file
`:w>>` *file*	append data to specified file

Examples:

`:w`	will work as a save
`:w` backup	save the contents of the editing buffer to a file named backup
`:w>>` info	append the contents of the editing buffer to info
`:10w` memo	write 10 line to a file named memo
`:2,5w>>` info	append lines 2 through 5 to file info

Changing the File You Are Editing

If you want to edit a different file, you don't have to quit and restart vi. You can use the `:e` and `:e!` commands:

`:e` *file*	edit the specified file
`:e!` *file*	edit the specified file, omit automatic check

Examples:

`:e` file2	Will begin editing a file named file2 (vi will check to see if you have saved your data. If there is unsaved file, vi will not let you change to file2).
`:e!` file3	Will begin editing a file named file3 (overriding the automatic checking protection).

Using Abbreviations

By using the :ab command, you can create abbreviations for frequently used words or expressions.

`:ab` *short long*	set *short* as an abbreviation for *long*
`:ab`	display current abbreviations
`:una` *short*	cancel abbreviation *short*

Example:

`:ab` tuw The University of Winnipeg

From now on, whenever you type tuw as a separate word (in input mode), vi will automatically replace it with The University of Winnipeg.

Using the .exrc File to Initialize vi

When vi starts, it looks for a file named .exrc in your home directory. If such file exists, vi will read and execute any ex commands in that file. This allows you to initialize your working environment automatically. A sample .exrc file:

```
“ set the options
        set wm=6
“set abbreviations
        ab pami Pattern Analysis and Machine Intelligence
        ab tuw The University of Winnipeg
“define the g macro
        map g 1G
```