

# First steps with vim

Ajit J. Thakkar

<http://www.unb.ca/chem/ajit/>  
ajit@unb.ca

## Contents

- 1 Don't confuse vim with vi
- 2 Survival skills
- 3 Basics
- 4 Visual block operations
- 5 Searching and replacing
- 6 Editing more than one file

## 1 Don't confuse vim with vi

*vim* is a powerful text editor created by Bram Moolenaar. It is a vast improvement over the UNIX editor *vi*. If you know how to use *vi*, then you can use *vim* in compatibility mode right away. However, not knowing *vi* or not wanting to learn *vi* is no reason to shy away from *vim*. Getting started with *vim* is easy — *vim* is not *vi* and does not share the latter's steep learning curve.

This tutorial shows you an easy way to begin productive work with *vim* using a small selection of *vim* commands that are more than adequate to perform a wide variety of tasks. More *vim* commands can be

learnt as the need arises, and as your desire to use *vim* more efficiently grows.

- 1 Not many people learn to ride a bicycle by reading how to do it. Learning a
- 2 text editor is not different in that respect. Such skills are acquired by practice. Start
- 3 using *vim* to edit a junk file or a copy of a valuable one as you read these notes.
- 4 Before you begin, check, or have someone check, that the *.vimrc* file has the line:
- 5 `set nocp bs=2 ru is gd ic scs`. It sets the *vim* options assumed by this tutorial.
- 6 Section 2 contains enough information to use *vim* in a primitive fashion. The full tutorial will get you to a competent novice level. Many of the *vim* commands described here will not work with *vi*; this article is not about *vi*. After you have mastered this material, look into *vim*'s text objects, regular expressions, Ex commands, key mappings and scripts.

## Notation

The prefix **C-** denotes a **Ctrl** key combination. For example, **C-x** means press **x** while holding the **Ctrl** key down. **C-x C-l** or **C-x-l** means press **x** and then **l** while holding the **Ctrl** key down. **C-w o** means first do a **C-w** combination, release the **Ctrl** key and then press **o**.

## 2 Survival skills

### Modes

Text editors need to distinguish whether a key is intended as input to the file being edited or as a command. Some text editors reserve `Ctrl` or `Alt` key combinations for commands, and normal, unmodified keys for input. Other editors have a command line from which commands can be issued. *vim* uses both these methods as well as explicit mode switching.

The modes you need to be concerned with most are (a) insert mode, (b) normal mode, (c) visual mode, and (d) command-line mode. Insert mode is for adding text to the file; since mistakes can be made when entering text, insert mode also has facilities for minor corrections. Simple editing commands are issued in normal mode. Visual mode is for performing editing operations on selected blocks of text. Complex commands require command-line mode.

### Insert and normal modes

- To begin editing the file *filename* with *vim*, issue the command: `vim filename`. You will begin in normal mode.
- The screen will show lines from the *filename* file. If the file ends midway in the screen, the screen lines below the file show a `~` in column one.
- The last line is a message line. To the right of the line, a pair of numbers (like 83,6) show the current cursor position in “line,column” form. At the extreme right you see a percentage which indicates how far in the file your cursor is.
- You can move around the file using the arrow keys: `←`, `→`, `↑`, `↓`.
- Try not to press keys thoughtlessly while you are in normal mode. Most letter keys issue editing commands.
- Don’t panic if you inadvertently issue commands in normal mode. If you are not in normal mode or you are not sure, return to normal mode by pressing the `Esc` key twice. Then, press the `u` key (no `Enter` required) as many times as required to *undo* all the unwanted changes.
- Most *vim* commands are case sensitive. For example, in normal mode, the key `U` corresponds to a different undo command: undo all changes made to the current line provided that the cursor has not been moved off the line since the changes were made.
- Pressing `i` puts you into *insert* mode at the position of the cursor, whereas pressing `o` puts you into insert mode with the cursor on a new line below the one you were on. The message line will show `-- INSERT --` as long as you remain in insert mode.
- In insert mode, you can add text at the cursor position by typing it in, delete the character at the cursor position with the `Del` key, and delete the preceding character with the `Backspace` key. You can move around the file with the cursor keys, and return to normal mode by pressing `Esc`.
- To end the *vim* session, switch to normal mode. Then the command `ZQ` (note upper-case letters are required) will let you quit without saving the changes you have made, whereas `ZZ` will exit after saving your file if changes have been made.

## 3 Basics

### Moving around

All the keys listed below move the cursor in insert, normal and visual modes. The first six keys also work in command-line mode.

Key	Cursor moves to
→	Next character
←	Previous character
C-→	Next word
C-←	Previous word
Home	Start of line
End	End of line
↓	Next line
↑	Previous line
C-PageDown	Bottom of page
C-PageUp	Top of page
PageDown	Next page
PageUp	Previous page
C-Home	Top of file
C-End	End of file

### Adding text

Creating new text is a primary purpose of a text editor and *vim* has many *insert mode* features to make this task easier. Insert mode also has facilities for minor corrections.

- C-w deletes the previous word, Del deletes the character under the cursor, and Backspace the previous character.
- The Insert key is a toggle between inserting characters and typing over existing ones; the message line shows -- REPLACE -- when the toggle is in the typeover state.
- Suppose you need to type a long word, such as “perturbation”, that you have already used in your file. Type in the first few letters, say **per**, and then

press C-p and *vim* will offer a word completion. If it is the one you want, simply continue typing. If *vim* has found the incorrect completion, say “periodic”, then just keep pressing C-p until *vim* finds the correct completion.

- C-n works just like C-p except that it attempts completion to a matching word that comes later in the text.
- In a computer code or list, the line you are typing may contain bits that appear in the previous or next line. The C-y key copies the next character from the line above, and the C-e key copies the next character from the line below.
- The key sequence C-x C-l attempts to complete the current line to a copy of one that occurs earlier. If the completion is right, continue typing. If not, press C-p until *vim* finds the right one. This feature is most useful for editing a computer program or a table.
- C-t indents the line by prepending a fixed number of spaces to the beginning of the line. C-d is the reverse and performs an unindent. They work regardless of your position on the line.
- C-a inserts the most recently inserted text again. For example, if you insert **bye** somewhere in the file, move to some other location, and press C-a, then **bye** will be inserted again.
- C-o lets you execute a single normal mode command and return to insert mode. For example, C-o :w Enter will save the file and return you to insert mode.

## 4 Visual block operations

Block operations are carried out in visual mode and require at most five actions.

1. Switch to normal mode.
2. Move to the start of the block.
3. Press a key to switch to visual mode and begin marking.
4. Move to the end of the block. The block will be highlighted as you move the cursor.
5. Press a key to perform the block operation.

There are three types of blocks.

1. A stream block is a set of contiguous characters.
2. A line block is a set of adjacent lines.
3. A box block is a rectangular region.

### Marking blocks

- To mark a stream block, position the cursor on the first character of the block, press `v`, and move to the last character of the block. The message line will show `-- VISUAL --`.
- To mark a line block, position the cursor on the first line of the block, press `V`, and move the cursor to the last line of the block. The message line will show `-- VISUAL LINE --`.
- To mark a box block, position the cursor on one corner of the block, press `C-v`, and move the cursor to the diagonally opposite corner. The message line will show `-- VISUAL BLOCK --`.
- `Esc` will cancel the marking and return you to normal mode.
- The start of a block can be changed after you have reached the end by pressing `o` and moving the cursor.

### Operating on blocks

- The table below shows the keys used for some common block operations.

Key	Block operation
<code>d</code>	delete
<code>y</code>	yank
<code>U</code>	change to uppercase text
<code>u</code>	change to lowercase text
<code>~</code>	toggle case of text
<code>&lt;</code>	unindent (full lines)
<code>&gt;</code>	indent (full lines)

- The same key may issue a different command in a different mode. For example, `u` is “undo” in normal mode but “change to lowercase” in visual mode.
- To copy a block, first *yank* it to an unnamed<sup>1</sup> register by pressing `y`, move the cursor to the position where you want the copy to be inserted, and then *put* the block there by pressing `p`. If additional copies are needed, simply move the cursor to each of the desired locations and press `p` there.
- To move a marked block, first *delete* it by pressing `d` causing *vim* to put the block into the unnamed register, move the cursor to the position where you want the moved block to be inserted, and then press `p`.
- To fill a box (not line or stream) block with copies of a single character, press `r`, and then press the key for the desired character.

---

<sup>1</sup>There are also registers named `a`, `b`, `c`, .... The command `"ay` yanks to the `a` register, `"ad` deletes to the `a` register, and `"ap` puts the contents of the `a` register. These are handy if you need to save several separate blocks of data for later insertion.

## 5 Searching and replacing

### Command-line mode

Search, replace and other commands are given in command-line mode. Some general features are summarized below.

- The commands are started from normal mode with one of the keys: (a) /, (b) ?, (c) :, or (d) ! which switch *vim* to command-line mode.
- The command appears on the message line as you type it and is not executed until you press **Enter** giving you the opportunity to edit it using the ←, →, Backspace, Del, Home and End keys.
- You can abandon a command you have started and return to normal mode by pressing Esc.
- Previous command-lines can be recalled with the ↑ and ↓ keys.

### Searching for patterns

- Search commands begin with either a / or a ?. The search starts at the cursor.
- If you want to find the next occurrence of “cricket”, begin typing `/cricket` slowly and note how *vim* carries out an incremental search using what you have typed at any given moment. If you pause after you type `/c` you will see that *vim* has found the next occurrence of “c”. As you add more letters and reach `/cric`, *vim* may already have found the next “cricket” because no other earlier strings matched “cric”. Then you can conclude the search by pressing **Enter**. In the worst case, you will have to type in the full search pattern and then hit **Enter**.

- The search does not distinguish between upper- and lower-case characters.
- The command `/tall` finds the next occurrence of the string ‘tall’ regardless of whether it is the word “Tall” or contained in words such as “forestall”, “Tallest”, or “installation”.
- You can restrict the search to complete words by sandwiching the word between `\<` and `\>` as in the command `/\<tall\>`.
- If the search pattern contains uppercase characters as in `/Moolenaar`, then the search will respect case.
- To search backwards, start the command with a ? instead of a /.
- The normal-mode command `n` finds the next occurrence of the previous search pattern whereas `N` finds the previous occurrence.
- Patterns can have blanks. A pattern will be found only if it is contained wholly on one line. `/cross section` will not find instances where “cross” is the last word on a line and “section” is the first word on the next line.
- In a search pattern, the character `.` stands for any character. `/gr.y` will find grey, gray, gr6y and even gr[y].
- In a search pattern, the character `*` means that the preceding character may occur zero or more times. So `/gr*y` will find gy, gry, grry, grrry, etc.
- If you need to search for `.` or `*` as such, then use `\.` and `\*` in the search pattern. The command `/a\.b` finds “a.b”.

- If the cursor is on an opening {, [, or (, then the normal-mode command % will attempt to find the matching closing }, ], or ). % will also take you from a closing bracket to the opening one.
- In normal mode, \* searches for the next instance of the word under the cursor. # searches backwards.

## Replacing text

- The general substitute command is:  
:range s/pattern/string/options
- The range can be line numbers separated by a comma. The symbol . specifies the current line, \$ the last line of the file, and % the entire file; see the examples below.

range	Meaning
%	Entire file
3,97	lines 3–97
.,86	current line to line 86
3,\$	line 3 to end of file
'<,>'	visually marked block

- All instances of “wave function” in the file are changed to “orbital” by :%s/wave function/orbital/. The command is case-insensitive and will change “Wave function” as well. The command will respect case if the option I is added after the final slash in the command.
- :1,13s/\<all\>/none/ changes “all” to “none” in the first 13 lines but does not make changes like “tall” to “tnone”.
- The first occurrence only, on the current line and each of the following lines, of “Hermitian” can be replaced

by “self-adjoint” with the command  
:.,\$s/Hermitian/self-adjoint/g

- To selectively change some occurrences of “day” to “night”, use the command :%s/day/night/c. It takes you to each instance of “day” in your file and asks, on the message line, whether you want the change made.
- The range can be a visual block. Mark the block and then press: and you will see the block’s range appear on the command-line as :’<,>’ after which you can type s/pattern/string/.

## 6 Editing more than one file

### Editing two files

Sometimes it is useful to edit two files at once — particularly when you need to copy or move a block from one file to the other.

- If you are editing file *main* with *vim*, then the command :e *sub* leaves the file *main* in an invisible editing buffer, and puts the file *sub* in another editing buffer that is displayed on the screen.
- You can switch the screen view from one buffer to the other by pressing C-6.
- You can mark and yank a block in one file, switch to the other file with C-6, and then do a put.

### Split-screen editing

- The two files can be viewed simultaneously in two half-screen horizontal windows with the C-w C-6 command. C-w o returns you to a full-screen view.
- You can switch from one window to the next with the command C-w w.