



Vim 101 HACKS

Practical Examples for Becoming
Fast and Productive in Vim Editor



Ramesh Natarajan
www.thegeekstuff.com

Table of Contents

| | |
|--|----|
| Introduction | 8 |
| About the Author..... | 9 |
| Copyright & Disclaimer..... | 10 |
| Foreword | 11 |
| Version | 12 |
| Chapter 1: Vim Basics | 13 |
| Opening a File | 13 |
| Saving a File | 13 |
| Closing a File | 14 |
| Types of Vim Modes | 14 |
| Moving around a File..... | 16 |
| Vim Configuration File (~/.vimrc)..... | 17 |
| Vim Version | 19 |
| Vim Installation | 20 |
| All About Vim Help | 21 |
| Use Vimtutor for Practice | 22 |
| Chapter 2: Basic Navigation..... | 23 |
| Hack 1. Scroll Full Page or Half Page | 23 |
| Hack 2. Word Navigation..... | 24 |
| Hack 3. Position cursor at specific location within a line | 25 |
| Hack 4. Paragraph, Section, Sentence Navigations..... | 26 |
| Chapter 3: Advanced Navigation..... | 27 |
| Hack 5. Screen Navigation..... | 27 |
| Hack 6. Redraw Screen with Current Line on Top, Bottom or Middle | 27 |

| | |
|--|-----------|
| Hack 7. Navigate to Top and Bottom of the File | 28 |
| Hack 8. Navigate to N th Character, N th Percentage of a File | 28 |
| Hack 9. Line Number Navigation | 29 |
| Hack 10. Source Code Navigation | 30 |
| Hack 11. Navigate From Insert Mode | 31 |
| Chapter 4: Expert Navigation | 32 |
| Hack 12. Jump Using CTRL-O and CTRL-I | 32 |
| Hack 13. Navigate Within a Very Long Line | 33 |
| Hack 14. Vim Command Line Navigation | 34 |
| Hack 15. Create Local Bookmarks inside file using Marks | 35 |
| Hack 16. Create Global Bookmarks inside Vim File | 37 |
| Hack 17. How to Display all Bookmarks | 38 |
| Hack 18. Navigate any Source Code effectively using Ctags | 41 |
| Hack 19. Convert Vim Editor to Beautiful Source Code Browser for Any Programming Language | 44 |
| Chapter 5: Basic Text Manipulation | 49 |
| Hack 20. Insert or Append Text | 49 |
| Hack 21. Replace Text | 50 |
| Hack 22. Substitute Text | 51 |
| Hack 23. Change Text | 52 |
| Hack 24. Join Lines Using nojoinspaces Option | 52 |
| Chapter 6: Advanced Text Manipulation | 54 |
| Hack 25. Copy One Character, or Word, or Line, or up to a Position | 54 |
| Hack 26. Paste before or after the copied Line / Word / Others | 55 |
| Hack 27. Delete Single Char, or Word, or Line | 56 |
| Hack 28. Inserting Content from Clipboard Buffer | 57 |
| Hack 29. Insert Content to Clipboard from File | 57 |
| Hack 30. Write Part of File to another File | 58 |
| Hack 31. Swap Adjacent Characters | 58 |
| Hack 32. The power of the . (dot) Command | 59 |
| Hack 33. Visual Mode Commands | 60 |

| | |
|---|-----------|
| Hack 34. Editing with :g | 62 |
| Chapter 7: Expert Text Manipulation | 65 |
| Hack 35. Copy Lines to Named Buffer for Later Use | 65 |
| Hack 36. Convert Inserted Text to Normal Mode Commands..... | 65 |
| Hack 37. Abbreviation and Unabbreviation | 66 |
| Hack 38. Automatic Spelling Correction | 67 |
| Hack 39. Record and Play Using Macros | 69 |
| Hack 40. Sort File Content | 74 |
| Hack 41. Recover Deleted Text | 75 |
| Hack 42. Add Automatic Headers to Files Using..... | 76 |
| Chapter 8: Vim as a Programmers Editor | 81 |
| Hack 43. Make Vim Highlight Your Code Smartly | 81 |
| Hack 44. Smart Indentation | 82 |
| Hack 45. Access Unix Man page for Functions from Vim | 83 |
| Hack 46. Jump to Variable Declaration | 84 |
| Hack 47. Align the Variable Assignment | 84 |
| Hack 48. Increment and Decrement Number Using CTRL Keys | 85 |
| Hack 49. Execute One Vim Command in Insert Mode | 86 |
| Hack 50. View Current File Details..... | 86 |
| Hack 51. Take Control of the Vim Status Bar | 87 |
| Hack 52. Change Case | 88 |
| Hack 53. Spell Check | 90 |
| Hack 54. Setup Quit Confirmation..... | 91 |
| Hack 55. Use :up and Avoid :w | 91 |
| Hack 56. Edit Current Buffer Content | 92 |
| Hack 57. Tabs and Spaces | 92 |
| Chapter 9: Vim Command Line Hacks | 94 |
| Hack 58. Open File in Read Only Mode | 94 |
| Hack 59. Recover Swap File Explicitly..... | 94 |
| Hack 60. Execute any Vim Command when opening a file | 97 |
| Hack 61. Execute Commands Stored in a File | 97 |

| | |
|---|------------|
| Hack 62. Skip Loading Plugins Temporarily | 98 |
| Hack 63. Enter Restricted Mode in Vim | 98 |
| Chapter 10: gVim Hacks | 99 |
| Hack 64. Display and Hide gVim Menu and Toolbar | 99 |
| Hack 65. Adding a Custom Menu or Menu Items to gVim | 100 |
| Hack 66. Change Font in gVim..... | 102 |
| Chapter 11: Vim Look and Feel, Tabs, and Windows | 103 |
| Hack 67. Split Windows Horizontally and Vertically | 103 |
| Hack 68. Change Window Title | 105 |
| Hack 69. Change Vim Colors | 106 |
| Hack 70. Edit Multiple Files in Tabs | 107 |
| Chapter 12: Additional Features in Vim Editor..... | 109 |
| Hack 71. Repeat an Operation N number of times | 109 |
| Hack 72. Undo and Redo Action | 109 |
| Hack 73. Open the File whose Name is under the Cursor | 110 |
| Hack 74. Edit Multiple Files Using the Traditional Method | 112 |
| Hack 75. Saving Files Automatically | 113 |
| Hack 76. Encrypt File in Vim | 114 |
| Hack 77. Save and Resume Vim Sessions | 114 |
| Hack 78. Execute Unix Shell Command Inside Vim | 116 |
| Hack 79. Review the Differences between Files using Vimdiff | 117 |
| Hack 80. Vim Map Command..... | 118 |
| Hack 81. Make Bash Shell work like Vim Editor | 121 |
| Hack 82. Set Vim Options..... | 122 |
| Hack 83. Unset Vim Options | 122 |
| Hack 84. Default registers and their uses | 123 |
| Hack 85. Numeric Registers and Recovering Deletes | 124 |
| Hack 86. Vim Directory Operation | 124 |
| Chapter 13: Power of Search | 127 |
| Hack 87. Navigation by Search | 127 |

| | |
|---|------------|
| Hack 88. Go to Next / Previous Occurrence of the Current Word ... | 128 |
| Hack 89. Search for a Character within a Line | 129 |
| Hack 90. 12 Powerful Find and Replace Examples | 129 |
| Hack 91. Search across Multiple Files using vimgrep | 137 |
| Hack 92. Highlight Search Results with Color | 138 |
| Hack 93. Vim Incremental Search | 139 |
| Hack 94. The Power of :match | 140 |
| Chapter 14: Automatic Completion..... | 142 |
| Hack 95. Automatic Word Completion..... | 142 |
| Hack 96. Automatic Line Completion | 143 |
| Hack 97. Automatic Filename Completion | 144 |
| Hack 98. Dictionary Completion | 145 |
| Hack 99. Thesaurus Word Completion..... | 146 |
| Hack 100. Automatically open a Pop-up menu for Completion | 149 |
| Hack 101. Automatically offers Word Completion as you type | 152 |
| Chapter 15: Bonus Hacks | 155 |
| Bonus Hack 1. Add Bullet Point Style to List of Items..... | 155 |
| Bonus Hack 2. Set Vim as Universal Default Editor using update- alternatives | 157 |
| Bonus Hack 3. Make Vim as Default Editor | 157 |
| Bonus Hack 4. Format a Paragraph..... | 158 |
| Bonus Hack 5. Edit Macros for Reuse | 158 |
| Bonus Hack 6. Indent Code Block | 159 |
| Bonus Hack 7. Power of Combination..... | 160 |
| Bonus Hack 8. Identify the changes done to a file..... | 161 |
| Bonus Hack 9. Refresh the Screen..... | 161 |
| Bonus Hack 10. Insert Non Keyboard Characters | 161 |
| Bonus Hack 11. Vim ex Mode | 162 |
| Bonus Hack 12. Place the cursor at the end of the match..... | 163 |
| Bonus Hack 13. View ASCII value of a character | 163 |
| Bonus Hack 14. Edit Binary files in Vim Editor | 164 |
| Bonus Hack 15. Folding - View Only Required Part of Code | 164 |

| | |
|--------------------------------|-----|
| Your Feedback and Support..... | 167 |
|--------------------------------|-----|

Introduction

“Productivity is being able to do things that you were never able to do before.”

--Franz Kafka--

If you are spending significant amount of your time on Unix or Linux environment, you may have to use Vi / Vim editor frequently. Mastering the Vim editor fundamentals and knowing how to use it effectively will instantly boost your productivity.

This book contains 101 Vim hacks (examples) that will help you to become fast and productive on the Vim editor.

All the hacks in this book are explained with appropriate Vim editor command examples that are crisp and easy to follow.

This book contains 15 chapters.

- **Chapter 1** explains basics of Vim editors for newbie.
- **Chapters 2 - 14** contain all the 101 hacks.
- **Chapter 15** contains additional bonus hacks. We'll be adding more hacks to this section in upcoming editions of this book.

Conventions used in this book:

- **CTRL-A** - Press the CTRL key and the A key at the same time.
- **10j** - Enter these characters in the sequence in the normal mode
- **:set nu** - Enter this command in the command line mode.

About the Author



I'm Ramesh Natarajan, author of The Geek Stuff blog thegeekstuff.com and this eBook.

I have done extensive programming in several languages and C is my favorite. I have done a lot of work on the infrastructure side including Linux system administration, DBA, Networking, Hardware and Storage (EMC).

I have also developed passworddragon.com — a free, easy and secure password manager that runs on Windows, Linux and Mac.

I'm also the author of free Linux 101 Hacks eBook - <http://www.thegeekstuff.com/linux-101-hacks-free-ebook/>

If you have any feedback about this eBook, please use this contact form - <http://www.thegeekstuff.com/contact> to get in touch with me.

Copyright & Disclaimer

Copyright © 2009 - Ramesh Natarajan. All rights reserved.

No part of this book may be reproduced, translated, posted or shared in any form, by any means.

The information provided in this book is provided "as is" with no implied warranties or guarantees.

Foreword

There are a lot of editors, most of which offer only modest functionality and little comfort. Well, all of these tools have their place, but a professional user needs professional tools. You have opted for Vim - a very good choice.

Vim was created when system resources were limited. These times have passed, but the result is a highly stable editor that runs on virtually any platform and has an exceptional command concept. It is certainly one of the best editors available for programmers. If you are adept with it, you can reach an incredible level of productivity. Vim offers just about everything you could ever want from an editor. What remains can be handled with macros, plug-ins and command line utilities.

Mastering an advanced editor is no small investment. However, after many years of experience, I can assure you that the effort bears interest richly. And do not forget, learning how to wield its power well can be a lot of fun, along the way!

The best that can happen is when an experienced user shows you the way and accompanies you as you learn. This book does exactly this. The learning curve is quite steep. After you have mastered the basics, you will learn step by step more advanced techniques. In order to become a master yourself, you only need three things: practice, practice, practice. This book will guide you for weeks or months there.

Now there is much to do, but you will have a very interesting time and gain much by it. Enjoy using Vim effectively. It's best to start immediately.

--Prof. Dr. Fritz Mehner, FH Südwestfalen, Germany

(Author of several [Vim plugins](#), including [bash-support vim plugin](#))

Version

| Version | Date | Revisions |
|---------|-------------|---------------|
| 1.0 | 21-Oct-2009 | First Edition |
| | | |

Chapter 1: Vim Basics

Before we begin to review the 101 hacks, let us understand a few basics of the Vim editor.

Opening a File

There are two methods to open a file. The following examples will open the `/etc/passwd` file.

Method 1: Open from command line as explained above.

```
$ vim /etc/passwd
```

Method 2: Open from the Vim editor after launching it.

```
$ vim
```

```
:e /etc/passwd
```

Saving a File

The following are methods to save a file.

| Save Methods | Description |
|--|-------------------|
| <code>:w</code> (or) <code>:write</code> | Save working file |
| <code>:up</code> (or) <code>:update</code> | Save working file |

| | |
|-------------------------------|---|
| <code>:w newfile.txt</code> | Save as newfile.txt |
| <code>:up newfile.txt</code> | Save as newfile.txt |
| <code>:w! newfile.txt</code> | Save as newfile.txt (with overwrite option) |
| <code>:up! newfile.txt</code> | Save as newfile.txt (with overwrite option) |

Closing a File

The following are methods to close a file and exit out of the Vim editor.

| Quit Methods | Description |
|------------------|--|
| <code>:x</code> | Save working file and exit |
| <code>:wq</code> | Save working file and exit |
| <code>ZZ</code> | Save working file and exit |
| <code>:q!</code> | Exit without saving working file |
| <code>:qa</code> | Exit all open files in the current Vim session |

Types of Vim Modes

There are several modes in Vim. For easy understanding let us separate these into two categories - basic modes and advanced modes.

Basic Vim Modes

It is absolutely essential to understand these three basic modes to use the Vim editor effectively.

| Mode | Description |
|--------------|--|
| Normal | The Vim editor starts in this mode, where you can execute all editor commands. |
| Insert | This mode is for inserting text. |
| Command Line | This mode is for executing ex commands at the bottom of the editor. For example, :wq |

Let's assume you want to create a helloworld.txt with the text "Hello World!". The following steps explain how you do this with the help of these three modes.

Step 1: Normal mode. Open the new file in Normal mode.

```
$ vim helloworld.txt
```

Step 2: Insert Mode. Go to Insert mode, and type Hello World!

```
i
```

Step 3: Command Line mode. Go to command line mode and type :wq to save the file and exit the editor.

```
<ESC> :wq
```

Advanced Modes

While these are not essential for a typical usage, it is good to understand all available Vim modes.

| Mode | Description |
|--------|--|
| Visual | You can select text (Using v, V or CTRL-V) in visual mode and execute Vim commands. For example, you can select a column of text and delete in visual mode. |
| Select | From :help vim-modes-intro <i>"Typing a printable character deletes the selection and starts Insert mode. In this mode, "-- SELECT --" is shown at the bottom of the window."</i> |
| Ex | From :help vim-modes-intro <i>"Like Command-line mode, but after entering a command you remain in Ex mode"</i> |

Moving around a File

In most applications you may be using the up, down, left and right arrow keys to do basic navigation. In the Vim editor, following are the basic navigation keys:

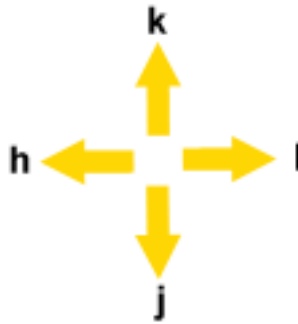


Fig: Basic Navigation Keys

| Navigation Key | Description |
|------------------|--------------------------|
| j | Down one line |
| k | Up one line |
| h | Move right one character |
| l (lower-case L) | Move left one character |

As a historical note the h, j, k and l keys had arrows on the ADM-3A terminal, which was commonly used with early UNIX systems. This is why those keys were chosen for navigation.

Note: If you are not comfortable with j, k, h and l keys, you can still use arrow keys to move around.

Vim Configuration File (~/.vimrc)

Local Vimrc

All configuration options that you define inside Vim are valid only for that particular Vim session.

For example, if you do `:set number` to display line numbers inside Vim, this will apply only in that particular Vim session. If you exit and start the Vim editor, the line number display will not be present anymore.

If you want to make your configuration settings permanent for future Vim sessions, you should add it to the `~/.vimrc` file as shown below.

```
$ vim ~/.vimrc

set number
set list
```

Location of local vimrc file:

| OS | Location |
|------------|---|
| UNIX/Linux | <code>\$HOME/.vimrc</code> Example: <code>/home/ramesh/.vimrc</code> Note: On Unix there is a <code>.</code> (period) before vimrc |
| Windows | <code>\$HOME/_vimrc</code> Example: <code>C:\Documents and Settings\ramesh_vimrc</code> Note: On Windows there is a <code>_</code> (underscore) before vimrc |

Global Vimrc

Global Vimrc is for sysadmins to add system-wide Vim configuration options that will be effective for all users of that system. Typically you should be modifying only the local vimrc file.

Location of global vimrc file:

| OS | Location |
|------------|--|
| UNIX/Linux | <code>\$VIM/.vimrc</code> Example: <code>/usr/share/vim/.vimrc</code> |
| Windows | <code>\$VIM_vimrc</code> Example: <code>C:\Program Files\Vim_vimrc</code> |

Vim Version

The latest stable release of Vim is 7.2 as of the first publishing of this book. All hacks mentioned in this book are tested against the latest stable version.

Execute `:version` from the Vim editor to identify the version of your Vim editor.

Vim version on Ubuntu:

```
$ vim

:version

VIM - Vi IMproved 7.2 (2008 Aug 9, compiled Mar 19 2009
15:27:51)
```

```
Included patches: 1-79  
Compiled by builddd@rothera.builddd  
Huge version with GTK2-GNOME GUI.
```

Vim version on Windows:

```
C:> vim  
  
:version  
  
VIM - Vi IMproved 7.2 (2008 Aug 9, compiled Aug 9 2008  
18:46:22)  
MS-Windows 32-bit GUI version with OLE support  
Compiled by Bram@KIBAALE  
Big version with GUI.
```

Vim Installation

Vim is the default editor on almost all Unix distribution. If you don't have latest Vim editor on your system, follow the instruction below to install it.

Install Vim on Windows:

- Go to vim.org -> Download -> PC: MS-DOS and MS-Windows -> Self-installing executable -> gvim72.exe
- Direct Download Link: <ftp://ftp.vim.org/pub/vim/pc/gvim72.exe>
- Download gvim72.exe and install it.

Install Vim on Ubuntu Linux:

```
$ sudo apt-get install vim-full
```

All About Vim Help

Type `:help` to view the built in help documentation that comes with the Vim editor.

```
$ vim  
  
:help
```

While browsing the Vim help document keep the following in mind:

- Anything within `| |` is a link
- Move your cursor to any character between `| |` and press **CTRL-]** to go to that particular help section.
- For example, `|quickref|` is a link.

| Help | Description |
|--------------------------------|--|
| <code>:help (or) :h</code> | Vim built in help documentation |
| <code>:helpgrep pattern</code> | Search help using pattern. For example, <code>:helpgrep saveas</code> Tip: Use <code>:cn</code> to jump for next occurrence of the pattern. |
| <code>:help 'option'</code> | Help on a Vim set option. For example: <code>:help 'list'</code> will give help about <code>:set list</code> |

| | |
|----------------------------------|--|
| <code>:help CTRL-X</code> | Help on Vim CTRL-X command. Use the same concept to get help on other CTRL- Vim commands. |
| <code>:help :x</code> | Help on Vim :x command Use the same concept to get help on other : Vim commands. |
| <code>:help<CTRL-D></code> | Help on auto completion. For example: <code>help<CTRL-D></code> will show all commands that starts with help. |

Use Vimtutor for Practice

The Vimtutor program has built in tutor file that contains step-by-step instructions to learn the Vim editor.

When you launch vimtutor, it copies the original tutor file and opens it automatically. You can modify this file and play around with it as you wish.

```
$ vimtutor
```

By default vimtutor opens the English tutor file. To open a tutor file in your specific language, give the language code at the end.

For example, the following opens the tutor file in Spanish.

```
$ vimtutor es
```

Chapter 2: Basic Navigation

There are three chapters dedicated to Navigation - Basic, Advanced and Expert Navigation.

If you only use the h, j, k, l characters to navigate, you will soon realize this is very painful and time-consuming.

Hacks in the navigation chapters will help you navigate file content very effectively with less key strokes.

Hack 1. Scroll Full Page or Half Page

On a large file, using j, k, h and l keys to scroll down pages is not effective.

Use the following page navigation keys.

| Navigation Key | Description |
|----------------|-----------------------|
| CTRL-F | Scroll down full page |
| CTRL-B | Scroll up full page |
| CTRL-D | Scroll down half page |
| CTRL-U | Scroll up half page |

Instead of using j and k, you can also use CTRL keys to scroll one line at a time as explained below.

There is a slight visual difference between using j, k keys and this CTRL keys. Try it out yourself to see the difference.

| Navigation Key | Description |
|----------------|----------------------|
| CTRL-E | Scroll down one line |
| CTRL-Y | Scroll up one line |

Hack 2. Word Navigation

Using h and l key to navigate horizontally is very painful and time consuming.

You can navigate words effectively using the word navigation keys mentioned below.

| Navigation Key | Description |
|----------------|--------------------------------------|
| w | Go to the beginning of next word |
| W | Go to the beginning of next WORD |
| e | Go to the end of current word |
| E | Go to the end of current WORD |
| b | Go to the beginning of previous word |
| B | Go to the beginning of previous WORD |

word Vs WORD

word consists of a sequence of letters, digits and underscores. **WORD** consists of a sequence of non-blank characters, separated with white space.

- For example, 192.168.1.3 contains seven words. But the whole 192.168.1.3 is considered as one WORD.

- If you are at the beginning of "192.168.1.3 devserver" and press **w** (to go to next word), you'll go to the first . (period). Because 192 is considered as a word.
- If you are at the beginning of "192.168.1.3 devserver" and press **W** (to go to next WORD), you'll go to d in "devserver". Because the whole 192.168.1.3 is considered as a WORD.



Fig: word Vs WORD

Hack 3. Position cursor at specific location within a line

Instead of just using **l** and **h** to navigate within a line, you can use the following to position cursor at various locations within the same line.

| Navigation Key | Description |
|-------------------------|---|
| 0 (zero) | Go to the starting of current line |
| \$ (dollar sign) | Go to the end of current line |
| ^ (caret sign) | Go to the first non blank character of current line |
| g_ | Go to the last non blank character of current line |

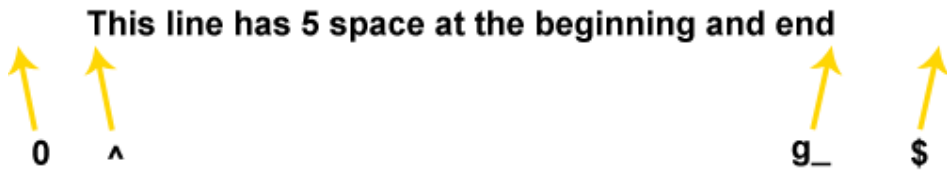


Fig: Line beginning and end navigation

Hack 4. Paragraph, Section, Sentence Navigations

Use the following keys for paragraph, section and sentence navigation.

| Navigation Key | Description |
|----------------|--|
| { | Go to the beginning of current paragraph |
| } | Go to the beginning of next paragraph |
| [[| Go to the beginning of the current section |
|]] | Go to the beginning of next section |
| (| Go to the beginning of previous sentence |
|) | Go to the beginning of next sentence |

Chapter 3: Advanced Navigation

Hack 5. Screen Navigation

Move cursor to top, center and bottom of the screen as explained below.

| Navigation Key | Description |
|----------------|---|
| H | Go to the first line of current screen. A mnemonic for H is "home" position - "0,0" a.k.a. the upper left corner of the screen |
| M | Go to the middle line of current screen |
| L | Go to the last line of current screen |

Hack 6. Redraw Screen with Current Line on Top, Bottom or Middle

You can redraw the screen with current line under the cursor at top, bottom or middle as shown below.

| Navigation Key | Description |
|--|---|
| z<ENTER> | Redraw the screen with the current line under the cursor at the top of the screen. |
| z- (lowercase z followed by hyphen -) | Redraw the screen with the current line under the cursor at the bottom of the screen. |

| | |
|-------------------------------------|---|
| z. (lowercase z followed by period) | Redraw the screen with the current line under the cursor at the middle of the screen. |
|-------------------------------------|---|

Hack 7. Navigate to Top and Bottom of the File

You can jump to the beginning and end of the file quickly as shown below.

| Navigation Key | Description |
|----------------|---|
| :0 | Go to the top of the file - method 1 |
| gg | Go to the top of the file - method 2 |
| 1G | Go to the top of the file - method 3 |
| :\$ | Go to the bottom of the file - method 1 |
| G | Go to the bottom of the file - method 2 |

Hack 8. Navigate to Nth Character, Nth Percentage of a File

You can navigate to Nth character or Nth percentage of a file as shown below.

| Navigation Key | Description |
|----------------|---|
| 50% | Go to the 50th percentage of file. Jump to the middle of the file. |

| | |
|------------|--|
| 75% | Go to 75% of the file. Jump to 3/4 th of the file. |
| 100l | Navigation key is: 100 followed by l. Go to the 100 th character from current position |
| 100<space> | Navigation key is: 100 followed by empty space Another way to go to the 100 th character from current position |
| :goto 25 | Go to 25 th character from the start of file |
| 25 | Navigation key is: 25 followed by pipe symbol Go to 25 th character in the current line |

Hack 9. Line Number Navigation

The following are command for line number setting inside Vim editor.

| Command | Description |
|----------------------------|-----------------------------|
| :set number :set nu | Display line numbers |
| :set nonumber :set nonu | Do not display line numbers |

| | |
|---------------------------------|---|
| <code>:set numberwidth=5</code> | By default the line number width is set to 4 characters. You can change this to 5 character using numberwidth |
|---------------------------------|---|

You can jump to a specific line number as explained below.

| Navigation Key | Description |
|-------------------|--|
| <code>:50</code> | Go to the 50 th line |
| <code>50gg</code> | Another way to jump to 50 th line |
| <code>50G</code> | Another way to jump to 50 th line |

Hack 10. Source Code Navigation

These keys are very helpful for programmers who are coding using Vim (or) for Sysadmins who write shell scripts.

For regular Unix users, these can come in handy while browsing any source code.

| Navigation Key | Description |
|----------------|---|
| <code>%</code> | Go to the matching character of the pair. Jump to the matching parenthesis <code>()</code> , or curly braces <code>{}</code> or square bracket <code>[]</code> . |

When you are debugging a code and missing a matching parenthesis, use the following shortcuts for rescue.

| Navigation Key | Description |
|----------------|--------------------------------|
| [(| Go to the previous unmatched (|
|) | Go to the previous unmatched) |
| [{ | Go to the previous unmatched { |
| } | Go to the previous unmatched } |

Hack 11. Navigate From Insert Mode

You will use either `w` or `W` for word navigation in a normal mode. However sometime you may want to navigate from an insert mode. To do this press Shift and Right arrow.

If you are insert mode and realized that you have to navigate to next word and type new text, you don't need to press `<ESC>` `w` to go to next word and press `i` to come to insert mode again.

Instead, use these navigation keys from the INSERT mode to navigate words.

| Navigation Key | Description |
|---------------------|---|
| SHIFT-<Right Arrow> | Go to right word-by-word in insert mode |
| SHIFT-<Left Arrow> | Go to left word-by-word in insert mode |

Chapter 4: Expert Navigation

Hack 12. Jump Using CTRL-O and CTRL-I

Vim keeps track of your navigation using a jump list. You can go backward and forward through that list.

The jump list keeps tracks of all the places you've been to by tracking file name, line number and column number.

To view the jump list:

```
:jumps
```

| Jump Navigation | Description |
|-----------------|--|
| CTRL-O | Jump back to previous spot |
| CTRL-I | Jump forward to next spot |
| 5CTRL-O | Jump to location#5 shown above location#0 |
| 5CTRL-I | Jump to location#5 shown below location#0 |

Let us assume that currently you are editing names.txt as shown below.

```
$ vim names.txt
```

```
:jumps
```



```
jump line col file/text
  3   484   19 /home/ramesh/scsi-list.txt
  2     5    0 /etc/passwd
  1     6   19 /etc.yp.conf
>  0    16   51 John Smith
  1    10    7 /etc/sudoers
  2     4    3 /etc/group
  3   204    3 /home/ramesh/my-projects.txt
```

- In this example `:jump` was executed when user was editing `names.txt` file
- The current location will be marked with `location#0` and `>` in front of it. There will be numbers above and below `location#0`.
- In this example, Current `location#0` is `"> 0 16 51 John Smith"` in the current file `names.txt`
- To jump to `/etc/password`, which is `location#2` above current location, press `2CTRL-O`
- To jump to `/etc/group`, which is `location#2` below current location, press `2CTRL-I`

Hack 13. Navigate Within a Very Long Line

When you have a very long line without any newline, Vim treats it as single line. So, when you execute `j` on that line, it will jump to next line. However you'll get a feeling that it has skipped lot of lines. But in reality it is just only long line.

Visual Line: Let us assume that a very long line is wrapped down into 5 visual lines. Let us call each and every individual line as visual lines for the discussion purpose.

The following shortcuts can help in navigating a very long line effectively.

| Navigation Key | Description |
|----------------|---|
| gj | Scroll down a visual line |
| gk | Scroll up a visual line |
| g^ | Go to the starting of current visual line |
| g\$ | Go to the end of current visual line |
| gm | Go to the middle of current visual line |

Hack 14. Vim Command Line Navigation

When opening a file from the command line, you can navigate to a particular position by specifying command line arguments as shown below.

| Command Line | Description |
|------------------------------------|--|
| \$ vim +143 <filename> | Go to the 143rd line of file |
| \$ vim +/search-term <filename> | Go to the first match of the specified search term from top |
| \$ vim +?search-term <filename> | Go to the first match of the specified search term from bottom |
| \$ vim -t TAG | Go to the specific tag |

For example, if you are opening /etc/passwd file to edit user jsmith, you can do the following. This will open the file /etc/passwd and jump directly to jsmith record.

```
$ vim +/^jsmith /etc/passwd
```

Hack 15. Create Local Bookmarks inside file using Marks

There are two types of bookmarks -- Local bookmarks and Global bookmarks. Let us review local bookmarks in this hack.

| Bookmark Command | Description |
|---------------------|---|
| ma | Bookmark the current location with name 'a' |
| `a (backtick a) | Jump to the exact location of bookmark 'a' |
| 'a (single quote a) | Jump to the beginning of the line containing the bookmark 'a' |

Within a single file when you want to go to a particular position or line, you can use local marking. If the bookmark name is a lower case letter, then that is local mark.

Type m{mark-name}. mark-name is a single alphabet character that is the name of the bookmark.

```
m{mark-name}
```

How to create a bookmark inside the Vim editor?

If you type "ma", it will create a bookmark on the current line at the current location with the name "a". In the following example, typing ma has created a bookmark at the exact location where the cursor is highlighted.



ma

```
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
ErrorLog "logs/error_log"
CustomLog "logs/access_log" combined
```

Fig: Bookmark current location inside Vim using ma

Note that Vim differs from Vi in that marks persist after exiting the editor. This is a powerful feature that's a surprise to many UNIX users.

Method 1 to access the bookmark : `{mark-name}

backtick followed by the mark name will move to the exact bookmark location. This will jump to the exact character location within the line from where it was bookmarked earlier.

For example, if you type ``a` , it will take you to the bookmark with name “a” .
i.e It will take you to the place where the cursor is high-lighted in the above Fig.

```
`a
```

Method 2 to access the bookmark : `'{mark-name}`

single-quote followed by the mark name. Move to the beginning of the bookmarked line.

For example, if you type `'a` , it will take you to beginning of the line that has the bookmark with name “a” . It will take you to the beginning of the “CustomLog logs/access_log combined” line in the above Fig.

```
'a
```

Hack 16. Create Global Bookmarks inside Vim File

When you have multiple files open, if you want to go to a particular position in any one of the open files, then you can use Global mark feature of Vim. If the bookmark name is an upper case letter, then that is a Global Bookmark.

The following sequence of steps will explain how to use a global bookmark while editing multiple files.

1. Open multiple files: `# vim /etc/passwd /etc/group`
2. While editing `/etc/passwd` file go to a specific line and type `mP` to create a global bookmark called `P`

3. Type `:n` to jump from the `/etc/passwd` file to `/etc/group` file
4. While editing `/etc/group` file go to a specific line and type `mG` to create a global bookmark called `G`
5. Type ``P` (back-tick followed by upper-case `P`), which will take you to the bookmark in `/etc/passwd`
6. From `/etc/passwd`, type ``G` (back-tick followed by upper-case `G`), which will take you to the bookmark in `/etc/group`.

An exercise for the reader: Using two less important files than these, make global marks in two files, then modify the current file and jump to the other one without saving first. What does Vim do? How do things change if you `":set autowrite"`?

Hack 17. How to Display all Bookmarks

If you've created several bookmarks and don't remember them, you can easily get a list of all the bookmarks by typing `:marks` as shown below.

```
:marks

mark line   col file/text
a      15      9 ypserver 192.168.1.119
b      11     18 domain THEGEEKSTUFF
G      56      0 group
P      45      0 passwd
```

This indicates that the following bookmarks were created:

- a - local bookmark with name "a" at line 15 and col 9. This also displays the text of line#15 . This is from the current open file, which is yp.conf
- b - local bookmark with name "b" at line 11 and col 18. This also gives the text of line#18. This is from the current open file, which is yp.conf
- G - global bookmark with name "G" at line 56 and col 0 of "group" file
- P - global bookmark with name "P" at line 45 and col 0 of "passwd" file.

Apart from the above bookmarks, anytime you type :marks inside Vim, you may get the following lines. These marks ' (single-quote), " (double quote), [,], ^ and . (period) are created and managed by Vim and you don't have direct control over them.

```
:marks

mark line  col file/text
'      8    12 #^IUse  broadcast  on  the local net
"      1     0 # /etc/yp.conf - ypbind configuration
[     11     0 domain THEGEEKSTUFF
]     11    19 domain THEGEEKSTUFF
^     11    19 domain THEGEEKSTUFF
.     11    18 domain THEGEEKSTUFF
```

You can use the above displayed default marks as shown below.

| Default Marks | Description |
|---------------|---|
| `" | To the position where you did last edit before exit |
| `[| To the first character of previously changed or yanked text |
| `] | To the last character of previously changed or yanked text |
| '< | To the first line of previously selected visual area |
| '> | To the last line of previously selected visual area |
| '. | To the position of where the last change was made |
| '^ | To the position where the cursor was the last time when Insert mode was stopped |

Quick Summary of Vim Bookmark Commands

- **ma** - Creates a bookmark called a
- **`a** - Jump to the exact location (line and column) of the bookmark a
- **'a** - Jump to the beginning of the line of the bookmark a
- **:marks** - Display all the bookmarks
- **:marks a** - Display the details of the bookmark with name a
- **`.** - Jump to the exact location (line and column) where the last change was performed
- **'.** - Jump to the beginning of the line where the last change was performed

Hack 18. Navigate any Source Code effectively using Ctags

Install ctags package

```
# apt-get install exuberant-ctags

(or)

# rpm -ivh ctags-5.5.4-1.i386.rpm

warning: ctags-5.5.4-1.i386.rpm: V3 DSA signature:
NOKEY, key ID db42a60e
Preparing...
##### [100%]
  1:ctags
##### [100%]
```

Generating ctags on your source code

Go to the directory where your source code is located. In the example below, I have stored all my C programming source code under ~/src directory.

```
# cd ~/src

# ctags *.c
```

The ctags command will create a file named tags that will contain information (tags) about the *.c program files. The following is partial content from a ctags file.

```
# cat tags

AddAcl  dumputils.c      /^AddAcl(PQExpBuffer aclbuf,
```

```
const char *keyword)$/;"      f      file:
ArchiveEntry      pg_backup_archiver.c
/^ArchiveEntry(Archive *AHX,$$/;"      f
AssignDumpId      common.c
/^AssignDumpId(DumpableObject *dobj)$/;"      f
```

Usage 1: Navigate to particular function definition by specifying the function name using :ta

In the example below, :ta main will take you to the main function definition inside the mycprogram.c

```
# vim mycprogram.c

:ta main
```

By using this facility you can navigate to any function definition by specifying the function name.

Usage 2. Navigating to the function definition from 'function call' using CTRL +]

When the cursor is over the function call, then press CTRL +] to go to the function definition.

In the following example, when the cursor is anywhere within the word ssh_xcalloc, pressing CTRL +] will take you to the ssh_xcalloc function definition.

```
# vim mycprogram.c
      av = ssh_xcalloc(argc, sizeof(char *));
```

Note: If the ctags couldn't find that function, you'll get the following message in the Vim status bar at the bottom: E426 tag not found ssh_xcalloc

Usage 3. Returning back to the caller from the definition using CTRL-T

After using CTRL-] to jump to a function definition, you can press CTRL-T which will take you back to the function call again.

Usage 4. Navigating through a list of functions which have similar names

In this example, :ta will go to first function definition whose name starts with get. Vim also builds a list of all the functions whose names start with get which we can navigate.

```
# vim mycprogram.c

:ta /^get
```

The following Vim commands can be used to navigate through the matched tag list.

| Vim command | Description |
|-------------|--------------------------------------|
| :ts | Display the tag list |
| :tn | Go to the next tag in the list |
| :tp | Go to the previous tag in the list |
| :tf | Go to the first function in the list |
| :tl | Go to the last function in the list |

Hack 19. Convert Vim Editor to Beautiful Source Code Browser for Any Programming Language

Navigating source code using tags is fast and functional but not very visually attractive. If you want to navigate source code in a manner similar to navigating in a file browser, you can use the Vim taglist plugin to make Vim into a source code browser.

Vim taglist plugin author Yegappan Lakshmanan has this to say about the plugin:

The "Tag List" plugin is a source code browser plugin for Vim and provides an overview of the structure of source code files and allows you to efficiently browse through source code files for different programming languages.

Install and Configure Vim Taglist plugin

Download Vim Taglist plugin from vim.org website as shown below.

```
$ cd ~  
  
$ wget -O taglist.zip  
http://www.vim.org/scripts/download_script.php?src_id=7  
701
```

Install Taglist Vim plugin as shown below.

```
$ mkdir ~/.vim  
  
$ cd ~/.vim  
  
$ unzip ~/taglist.zip
```

```
Archive:  ~/taglist.zip
inflating:  plugin/taglist.vim
inflating:  doc/taglist.txt
```

Enable the plugin by adding following line to the ~/.vimrc

```
$ vim ~/.vimrc

filetype plugin on
```

Prerequisite: ctags should be installed to use taglist plugin. But it is not a must to generate the tag list manually by ctags command for using taglist plugin.

Usage 1: Open the Tag List Window in Vim using :TlistOpen

```
# vim mycprogram.c

:TlistOpen
```

From the Vim editor, execute :TlistOpen as shown above, which opens the tag list window with the tags of the current file as shown in the figure below.

```

" Press <F1> to display help
mod_dbd.c (/usr/src/httpd-2.4.18)
- macro
|   APR_WANT_MEMFUNC
|   APR_WANT_STRFUNC
|   NMIN_SET
|   NKEEP_SET
|   NMAX_SET
|   EXPTIME_SET
|   DEFAULT_NMIN
|   DEFAULT_NKEEP
|   DEFAULT_NMAX
|   DEFAULT_EXPTIME
|   ISINT
- struct
|   dbd_group_t
- typedef
|   dbd_cfg_t
|   dbd_group_t
|   svr_cfg
|   cmd_parts
|   dbd_query_t
- Tag List 1,5 Top mod_dbd.c [R0]

{
    svr_cfg *svr = apr_
    dbd_cfg_t *cfg = sv
    sizeof(dbd_cfg_t));

    cfg->server = s;
    cfg->name = no_dbdr
    ngful error messages */
    cfg->params = ""; /
    isconfiguration */
    cfg->persist = -1;
    #if APR_HAS_THREADS
    cfg->nmin = DEFAULT
    cfg->nkeep = DEFAUL
    cfg->nmax = DEFAULT
    cfg->exptime = DEFA
    #endif
    cfg->queries = apr_

    return svr;
}

static void *merge_dbd
oid *basev, void *addv)

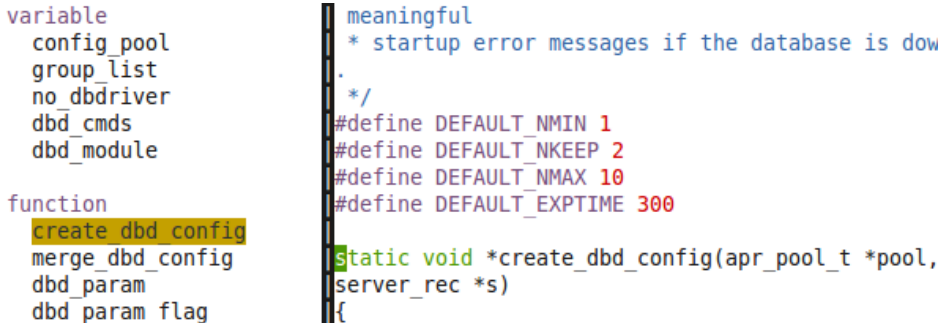
```

Fig: List of Functions and Variables in a Program

Usage 2: Jump to a Function Definition within a source code

By clicking on the function name in the left side panel, you jump to the definition of the function as shown in the Figure below.

Apart from jumping to functions quickly, you can jump to classes, structures, variables, etc., by clicking on the corresponding values from the tag-browser in the left side panel.

A screenshot of the Vim editor interface. On the left, a list of symbols is shown, categorized into 'variable' and 'function'. Under 'variable', there are 'config_pool', 'group_list', 'no_dbdriver', 'dbd_cmds', and 'dbd_module'. Under 'function', there is 'create_dbd_config' (highlighted in yellow), 'merge_dbd_config', 'dbd_param', and 'dbd_param flag'. On the right, the corresponding C code is displayed. It starts with a comment '/* startup error messages if the database is down */', followed by several #define macros for DEFAULT_NMIN, DEFAULT_NKEEP, DEFAULT_NMAX, and DEFAULT_EXPTIME. Then, the definition of 'static void *create_dbd_config' is shown, taking 'apr_pool_t *pool' and 'server_rec *s' as arguments. The cursor is positioned at the start of the function definition.

```
variable
config_pool
group_list
no_dbdriver
dbd_cmds
dbd_module

function
create_dbd_config
merge_dbd_config
dbd_param
dbd_param flag

/* startup error messages if the database is down */
#define DEFAULT_NMIN 1
#define DEFAULT_NKEEP 2
#define DEFAULT_NMAX 10
#define DEFAULT_EXPTIME 300

static void *create_dbd_config(apr_pool_t *pool,
server_rec *s)
{
```

Fig: Jump to a specific function quickly

Usage 3: Jump to a function defined in another source file

When you encounter a function in a source file that is defined elsewhere, and you want to go to the function definition, you can do this via two different methods.

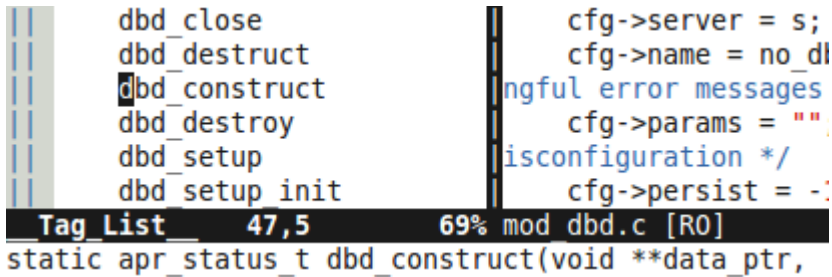
Method 1: If you had the ctags generated for that file, when the cursor is in the function call pressing CTRL +] will take you to the function definition. The tag list window will show the tags for that newly opened file.

Method 2: Open another file also in the same Vim session. Vim will update the tag list window with the information about that file. Search for that function name in the tag list window, and press <CR> on that function name, and Vim will take you to the function definition.

Usage 4: Viewing the prototype/signature of functions or variables.

Press 'space' when the cursor is in the function name or in the variable name in the tag list window to show the prototype (function signature) of it in the Vim status bar as shown below.

In the example below, click on dbd_construct within the tag window and press space to display the function signature in the bottom Vim Status bar.



The screenshot shows a Vim window with a tag list on the left and a function signature in the status bar. The tag list contains the following entries:

```

dbd_close
dbd_destruct
dbd_construct
dbd_destroy
dbd_setup
dbd_setup_init

```

The status bar displays the following information:

```

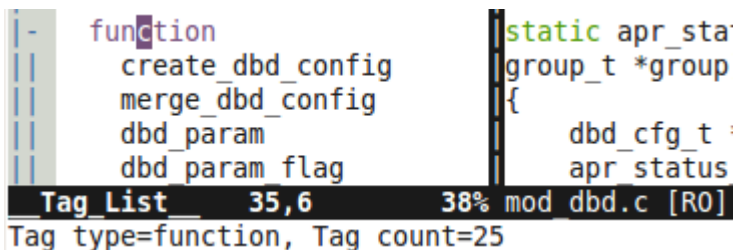
Tag List 47,5 69% mod dbd.c [R0]
static apr_status_t dbd_construct(void **data_ptr,

```

Fig: Display Function signature

Usage 5: Viewing the total number of functions or variables in a source code file

Pressing 'space' with the cursor over a tag type (like function or variable) shows the count of tags of that type. In the example below, when the cursor is over 'function' press space, which will display the total number of functions in the current source code.



The screenshot shows a Vim window with a tag list on the left and a function count in the status bar. The tag list contains the following entries:

```

function
create_dbd_config
merge_dbd_config
dbd_param
dbd_param flag

```

The status bar displays the following information:

```

Tag List 35,6 38% mod dbd.c [R0]
Tag type=function, Tag count=25

```

Fig: Display total number of functions

Chapter 5: Basic Text Manipulation

Hack 20. Insert or Append Text

Insert Text

The following explains various methods to insert text into a file.

| Key | Description |
|-------------|---|
| i | Insert text at the current position |
| I | Insert text at the beginning of the line Key: Uppercase I as in India |
| o | Insert a new line after the current line and insert text Key: Lowercase o as in orange. |
| O | Insert a new line before the current line and insert text Key: Uppercase O as in Orange. |
| :r FILENAME | Insert another file content into current file after the current line |
| :r! COMMAND | Insert output of a command into current file after the current line |

For example, you can insert the current date and time into the file you are editing by executing the following.

```
:r! date
```

Append Text

The following explains various methods to append text.

| Key | Description |
|-----|---|
| a | Append text after the current cursor position |
| A | Append text to the end of the line |

Hack 21. Replace Text

The following explains various methods to replace text in a file.

| Key | Description |
|------|--|
| r{c} | To replace a single character with the single character {c} |
| R | To replace characters until you press <ESC> (note: this acts like A after reaching the end of a line, rather than wrapping and replacing characters on the next line) |

Hack 22. Substitute Text

The following explains various methods to substitute text in a file.

| Key | Description |
|-----|---|
| s | Substitute the current character with new character |
| S | Substitute the current line with new text |
| 4s | Substitute 4 characters (from current position) with new text |
| 4S | Substitute 4 lines (from current line) with new text |

Let us assume that we are editing the following file

```
$ vim employee.txt
```

```
100      John Doe      DBA
200      John Smith    Sysadmin
300      Raj Patel     Developer
```

- If your cursor is over D on “John Doe” and if you type 2s, you’ll be replacing ‘Do’ with any new text that you type.
- If your cursor is anywhere on line 1 and if you type 2S, you’ll be replacing both the 1st and the 2nd line with any new text that you type.

Hack 23. Change Text

The following explains various methods to change text into a file.

| Key | Description |
|-----|---|
| cc | <p>Change the whole current line; synonymous with S.</p> <p>This will delete the full current line and put you in INSERT mode for new text.</p> |
| C | <p>Change the current line from the current cursor position.</p> <p>This will delete text in the current line from the current cursor position and put you in INSERT mode for new text.</p> |

Hack 24. Join Lines Using nojoinspaces Option

To join (combine) two lines do the following.

```
J
```

If there is no special character at the end of the line the cursor is on, the J command will add only one space when joining the two lines.

Vim 101 Hacks

www.thegeekstuff.com

If there is a special character (for example punctuation) at the end of the line the cursor is on, the J command will add two spaces when joining the two lines.

To avoid this and always use one space when joining the two lines, set the following option.

```
:set nojoinspaces
```

Chapter 6: Advanced Text Manipulation

Hack 25. Copy One Character, or Word, or Line, or up to a Position

| Key | Description |
|-------------------------|-------------------------------------|
| y<char navigation keys> | To copy a single character |
| y<word navigation keys> | To copy a single word |
| y<line navigation keys> | To copy a single line |
| y<mark name> | To copy up to a bookmarked line |
| y` <mark name> | To copy up to a bookmarked position |

The following are a few points to remember:

- The mnemonic for y is "yank"
- You can combine any operations along with the navigation keys to execute that operation until that point. For example, to copy a word, press yw.
- You can expand the above table to any other operations. For example, to copy up to the line at the middle of the screen use yM
- This concept can also be used for other operations. i.e It's not just for copy/paste.

Hack 26. Paste before or after the copied Line / Word / Others

| Key | Description |
|------------------|--|
| p (lower-case P) | Paste immediately after the current cursor location |
| P (upper-case P) | Paste immediately before the current cursor location |

If you've performed a few delete operations and if you would like to paste one of those deleted words, use the following method.

First, view the register with the following command.

```
:reg
```

Recent deleted content will appear in 0 - 9 register. Make a note of the register number of the deleted word that you would like to paste.

If you would like to paste the word from the register number 3, execute the following.

```
"3p
```

For example, here you can see all the registers "0 - "9 containing the text that was deleted earlier.

```
:reg  
  
--- Registers ---  
" " -----^J  
"0 " -----^J  
"1 eth0^J  
"2 this a testing text^J  
"3 100 John Smith^J  
"4 DBA^J  
"5 Section Data^J" Overview^J  
"6 command not found^J  
"7 /dev/sdc0 none swap sw 0 0  
"8 ^J  
"9 ^J
```

If you want to paste the content from register 7 highlighted above to your current document execute the following:

```
"7p
```

Hack 27. Delete Single Char, or Word, or Line

Delete is similar to copy. However you have to use d instead of y.

| Key | Description |
|-----|---------------------------------------|
| x | Delete the current character |
| dw | Delete the current word |
| dj | Delete the current line and next line |

Hack 28. Inserting Content from Clipboard Buffer

If you've copied text from a web browser, or any other application, you can paste it directly to the current open file in the Vim editor as shown below.

| Copy from clipboard | Description |
|---------------------|--|
| SHIFT-INSERT | Paste clipboard content to editor (ensure you are in insert mode first!) |
| "*p | Paste clipboard content to editor in normal mode |

Hack 29. Insert Content to Clipboard from File

You may want to put text from the current file into the clipboard. Once the text is transferred to the clipboard, you can paste it into any other application.

| Copy to clipboard | Description |
|-------------------|--|
| :%y+ | Copy the whole file to the clipboard |
| :y+ | Copy the current line from the file to the clipboard |
| :N,My+ | Copy the specific range from file to the clipboard |

To copy the visual selected line to the clipboard, first visually select the lines, and `:y+` which will appear as `:'<,'>y+`

After copying, you can paste this content to any other application using the traditional `<CTRL>+V` operation.

Hack 30. Write Part of File to another File

To write part of a file to a new file you can use any of the following methods.

Method 1: Select the particular lines in the visual mode. Go to visual mode (using either `v` or `V`) and navigate to the desired line, then do the following.

```
:w newfilename
```

Method 2: To write a part of file into another file, you can specify the range as shown below. This will write the lines from 5th to 10th of current file to the new file.

```
:5,10w newfilename
```

Hack 31. Swap Adjacent Characters

If you make a simple typing mistake with misplaced adjacent characters, you can use `xp`. For example, if you've typed 'teh' instead of 'the', navigate to e and press `xp`, which will fix the typo automatically.

xP

In reality, xp is really not fixing the typo.

- x - deletes the current character (e), which also moves the cursor to next character (h)
- p - Pastes the previously deleted character (e) after the current character (h).
- The mnemonic for xp is "transpose".

Hack 32. The power of the . (dot) Command

The . (dot) command is simple and yet powerful. The . (dot) command repeats the last file-content-affecting command. The following example demonstrates the use of the . command.

1. Search for a string in a file using: `/john<enter>`
2. Replace john with jason using: `cwjason<ESC>`
3. Search for the next occurrence of john: `n`
4. Replace john with jason using: `.(dot)`

In the above example, at step#4, you don't need to type cwjason again. Instead, simply type . (dot), which will execute the last change command, which is cwjason.

Hack 33. Visual Mode Commands

The following are different types of visual modes:

| Visual Mode Types | Description |
|-------------------|---|
| v (lower-case) | Start the normal visual mode. Use arrow keys to navigate and select text in visual mode. |
| V (upper-case) | Start the line visual mode. |
| CTRL-V | Start the visual block mode. |

The following screenshots show the difference between these three visual modes.

Normal Visual Mode

In this example, the whole 1st line and part of the 2nd line are selected. This can be done by pressing v (lower case V) and using arrow keys to navigate to a specific character in a line.

```
100 Jason Smith Developer
200 John Doe Sysadmin
300 Sanjay Gupta QA
400 Ashok Sharma DBA
~
~
```

Fig: Normal Visual Mode

Line Visual Mode

In this example, the whole 1st line and 2nd line is selected. This can be done by pressing V (upper case V) and using arrow key . In this mode, when you press down arrow (or j key), it will select the whole line.

```
100 Jason Smith Developer
200 John Doe Sysadmin
300 Sanjay Gupta QA
400 Ashok Sharma DBA
~
~
```

Fig: Line Visual Mode

Block Visual Mode

In this example, only the 2nd column (employee names) is selected. This can be done by pressing CTRL-V and using arrow key to select the columns.

```
100 Jason Smith Developer
200 John Doe Sysadmin
300 Sanjay Gupta QA
400 Ashok Sharma DBA
~
~
```

Fig: Block Visual Mode

| Visual Mode Commands | Description |
|----------------------|---|
| <ESC> | Exit visual mode |
| d | Delete only the highlighted text. For example, if only part of the line is selected, it deletes only the selected text from that line. |
| D | Delete rows under highlighted text. For example, if only part of the line is selected, it deletes the entire line. |
| y | Copy (yank) only the highlighted text |
| Y | Copy (yank) rows under highlighted text |
| c | Delete highlighted text and go to insert mode |
| C | Delete rows under highlighted text and go to insert mode |

Hack 34. Editing with :g

The following are some awesome examples to show the power of :g

| Example | Description |
|----------|------------------------------------|
| :g/^\$/d | Delete all empty lines in the file |

| | |
|---|--|
| <code>:g/^\s*\$/d</code> | Delete all empty and blank lines in the file |
| <code>:g/^\\$/./-j</code> | Reduce multiple blank lines into a single blank lines |
| <code>:g/pattern/d</code> | Delete the line which has a specific pattern |
| <code>:g/pattern/ . w>>filename</code> | Extract lines with specific pattern and write it into another file |
| <code>:g/^/m0</code> | Reverse a file |
| <code>:g/^\s*PATTERN /exe "norm! I/* \<ESC>A */\<ESC>"</code> | Add a C Style comment (/* text */) to all lines matching the pattern |

Negate Operation Using :g! or :v

Negate operation will match everything except the pattern as explained below.

Create the following employees.txt file

```
$ vim employees.txt

Emma Thomas:100:Marketing
Alex Jason:200:Sales
Madison Randy:300:Product Development
Sanjay Gupta:400:Support
Nisha Singh:500:Sales
```

Delete all lines containing Sales:

```
:g/Sales/d
```

Delete all lines except Sales:

```
:g!/Sales/d
```

(or)

```
:v/Sales/d
```


Chapter 7: Expert Text Manipulation

Hack 35. Copy Lines to Named Buffer for Later Use

You can copy (yank) lines to a named buffer, which you can use later as shown below.

Valid named buffer: a to z (26 total valid named buffers)

| Command | Description |
|---------|--|
| "ayy | Copy current line to buffer "a" |
| "a5yy | Copy 5 lines to buffer "a" |
| "ap | Paste copied lines from buffer "a" after the cursor |
| "aP | Paste copied lines from buffer "a" before the cursor |

Hack 36. Convert Inserted Text to Normal Mode Commands

Have you ever typed a normal mode command when you are in insert mode by mistake? This hack is very helpful under those situations as shown below.

- Assume that you have the following text inside Vim editor - john
- You wanted to change john to Jason.
- You forgot that you are insert mode and typed the following -
cwjasonjohn

- Now you can simply press the <F2> function key, which will undo your previous insert and use it as normal mode commands which in this case change the word john to Jason.

To achieve this, you should add the following line to your .vimrc

```
$ cat ~/.vimrc  
  
inoremap <F2> <ESC>u@.
```

Note: There is a . (dot) next to '@' at the end of the above line

Hack 37. Abbreviation and Unabbreviation

In the following example, whenever you type US it will expand to "United States" when you've defined the following abbreviation.

To abbreviate a word temporarily, execute the abbr command in command mode as shown below.

```
:abbr US United States
```

To abbreviate a word permanently, you can place it in the .vimrc as shown above.

To remove an abbreviation definition temporarily, execute the noabbr command in command mode.

```
:noabbr US
```

To remove an abbreviation definition permanently, remove it from .vimrc

If you are typing your website URL or email address frequently, you can create an abbreviation as shown below.

```
:iabbrev tgs http://www.thegeekstuff.com  
:iabbrev myemail ramesh.thegeekstuff@gmail.com
```

After the above, whenever you type myemail, it will expand your email address automatically.

You can also insert special keys in the iabbrev value. For example, you can add carriage return key <CR> as shown below.

```
:iabbrev TRR Thanks,<CR>Regards,<CR>Ramesh Natarajan
```

In this example, whenever I type TRR, it will expand to the following.

```
Thanks,  
Regards,  
Ramesh Natarajan
```

Hack 38. Automatic Spelling Correction

The autocorrect.vim plugin has a collection of all typical spelling mistakes and their correct spelling.

Author of the Plugin, Anthony Panozzo describes the plugin as:

“Correct common typos and misspellings as you type”

The following are few examples from the autocorrect.vim plugin.

```
ia Britian Britain
ia Brittish British
.
.
ia Acceptible Acceptable
ia accesories accessories
```

Install and Configure autocorrect.vim plugin

Download the plugin from vim.org website.

```
$ cd ~

$ wget -O autocorrect.tar
http://www.vim.org/scripts/download_script.php?src_id=1
0423

$ tar xvf autocorrect.tar
```

Install autocorrect.vim plugin. From Vim, you can execute “:source /path/to/the/autocorrect.vim” whenever needed (or) For permanent usage, add the following line to ~/.vimrc

```
$ vi ~/.vimrc

:source ~/autocorrect.vim
```

After doing this, when you make a typo of a word that is in the autocorrect.vim list, it will be corrected automatically.

```
$ vi test-typo.txt
```

```
thsi is acceptable
```

[**Note:** Above line will automatically change to "this is acceptable"]

When you want the expansion or spelling mistake correction to be stopped for a particular word, you can do the following inside vim. This is a temporary unabbreviation. If you want this to be permanent, remove the word either from ~/.vimrc or ~/.autocorrect.vim.

```
$ vim test-typo.txt
```

```
:una US
```

Hack 39. Record and Play Using Macros

This hack explains how to perform record and play back macros inside Vim using an example.

High level steps to record and play back a macro inside Vim.

- **Step 1:** Start recording by pressing q, followed by a lower case character to name the macro

- **Step 2:** Perform any typical editing actions inside Vim editor, which will be recorded
- **Step 3:** Stop recording by pressing q
- **Step 4:** Play the recorded macro by pressing @ followed by the macro name
- **Step 5:** To repeat macros multiple times, press : NN @ macro name. NN is a number

This example explains how you can execute the same command, with different inputs. i.e Framing the same command, with different arguments.

1. Open the change-password.sql that has only the names.

```
$ vim change-password.sql
Annette
Warren
Anthony
Preston
Kelly
Taylor
Stiller
Dennis
Schwartz
```

2. Start the Recording and store it in register a

```
q a
```

- q indicates to start the recording
- a indicates to store the recordings in register a

- When you do q a, it will display the message “recording” at the bottom of the screen.

3. Go to Insert Mode and Type ALTER USER

```
I "ALTER USER "
```

Place the cursor anywhere in the first line, and then press I (upper case i), which will take you to the first character of the line. Type ALTER <space> USER <space>

4. Copy the Next Word (i.e the name)

```
<ESC> w yw
```

- Press ESC, and then press w to go to the next word (name).
- yw, copies the current word (name).

5.Go to the end and type IDENTIFIED BY '

```
<ESC> A " IDENTIFIED BY `"
```

- Press ESC, and A to move the cursor to the end of the line, and then type space.
- Type IDENTIFIED BY '

6. Paste the copied Name

```
<ESC> p
```

Press ESC, and then type p to paste the name that was copied in the step #4.

7. Complete the quote at the end.

```
<ESC> A ` ;
```

Press ESC, and A to go to the end of the line, and ` ;

8. Jump to the next line and stop the macro recording.

```
<ESC> j q
```

- j to move to the next line.
- q to stop the recording

Note: The recording message shown at the bottom of the screen will now disappear. At this stage, the file change-password.sql will look like the following.


```
ALTER USER Annette IDENTIFIED BY 'Annette';  
Warren  
Anthony  
Preston  
Kelly  
Taylor  
Stiller  
Dennis  
Schwartz  
~  
~
```

Fig: Vim Macro completed the recording

9. Repeat the Macro with the arguments in the corresponding line

```
8 @ a
```

- Now repeat this job 8 times by typing 8@a
- @a repeats the macro "a" one time.
- 8@a repeats the macros "a" 8 times completing the rest of the line automatically as shown below

```
ALTER USER Annette IDENTIFIED BY 'Annette';  
ALTER USER Warren IDENTIFIED BY 'Warren';  
ALTER USER Anthony IDENTIFIED BY 'Anthony';  
ALTER USER Preston IDENTIFIED BY 'Preston';  
ALTER USER Kelly IDENTIFIED BY 'Kelly';  
ALTER USER Taylor IDENTIFIED BY 'Taylor';  
ALTER USER Stiller IDENTIFIED BY 'Stiller';  
ALTER USER Dennis IDENTIFIED BY 'Dennis';  
ALTER USER Schwartz IDENTIFIED BY 'Schwartz';  
~  
~
```

Fig: Vim Macro Play completed

Hack 40. Sort File Content

Starting from Vim version 7, a built in Vim sort command is available.

Sort file content from Vim as shown below.

```
:sort
```

Sort part of the file content as shown below.

- Go to visual mode by pressing v
- Use arrow keys and select the lines that need to be sorted.
- Press :, which will show : '<,>' at the bottom of the Vim.
- Add !sort at the end to sort the selection

```
: '<,>'!sort
```

The following options are available for the :sort Vim command

| :sort option | Description |
|--------------|---|
| :sort | Sort in ascending order |
| :sort! | Sort in descending order |
| :sort i | Ignore case while sorting |
| :sort u | Remove duplicate lines. U stands for unique. |

| | |
|-----------|--|
| :sort! ui | You can also combine all the sort command options. |
|-----------|--|

Hack 41. Recover Deleted Text

If you've deleted text by mistake you can recover it. You can recover up to 9 deleted text fragments.

| Recover Delete | Description |
|----------------|-----------------------------------|
| "1p | Recover the last delete |
| "2p | Recover the second to last delete |
| "3p | Recover the third to last delete |

If you don't know exactly when you've deleted something, you can browse through all the 9 delete buffers as shown below. When you see the text that you wish to recover, just stop at that stage.

Browse all delete buffers, until you find correct one.

```
"1pu.u.u.u.u.
```

You can also see what's in registers 0 through 9 (the delete registers) using :reg - this tells you exactly what is in each register.

Hack 42. Add Automatic Headers to Files Using

Let us review how to use the powerful autocmd feature of Vim to create a header section inside a file (for example, header in a C programming code) with file name, creation date, last modified date/time automatically populated when you open a file in vi.

Vim autocmd syntax:

```
autocmd {event} {pattern} {cmd}
```

Events: There are more than 40 autocmd events. The following are few sample autocmd events.

| Event | Description |
|--------------|---|
| BufNewFile | Starting to edit a file that doesn't exist |
| FileReadPre | Before reading a file with a ":read" command |
| BufWritePre | Starting to write the whole buffer to a file |
| FileWritePre | Starting to write part of a buffer to a file |
| BufDelete | Before deleting a buffer from the buffer list |
| BufWipeout | Before completely deleting a buffer |
| BufNew | Just after creating a new buffer |
| BufEnter | After entering a buffer |
| BufLeave | Before leaving to another buffer |

| | |
|------------|--------------------------------|
| SwapExists | Detected an existing swap file |
|------------|--------------------------------|

Many developers want some default header for their programs. For example, when opening a ".c" file, you typically need a file header with a number of items. The following template is loaded automatically whenever I open a new ".c" file. You can achieve this in three steps as mentioned below.

```
/* .....  
  
* File Name : l.c  
  
* Purpose :  
  
* Creation Date : 22-12-2008  
  
* Last Modified : Mon 22 Dec 2008 10:36:49 PM PST  
  
* Created By :  
  
.....*/
```

Step 1: Create a template file

Save the above template in a text file with ":insert" in the first line, followed by the template and a "."(dot) in the last line as shown below.

```
$ cat c_header.txt  
  
:insert  
/* .....  
  
* File Name :  
  
* Purpose :  
  
* Creation Date :
```

```
* Last Modified :  
* Created By :  
.....*/  
.
```

Step 2: Add autocmd commands to ~/.vimrc

Add the following lines in the `~/.vimrc` file.

```
$ cat ~/.vimrc

autocmd bufnewfile *.c so /home/jsmith/c_header.txt
autocmd bufnewfile *.c exe "1," . 10 . "g/File Name
:./s//File Name : " .expand("%")
autocmd bufnewfile *.c exe "1," . 10 . "g/Creation Date
:./s//Creation Date : " .strftime("%d-%m-%Y")
autocmd Bufwritepre,filewritepre *.c execute "normal
ma"
autocmd Bufwritepre,filewritepre *.c exe "1," . 10 .
"g/Last Modified :./s//Last Modified :./Last Modified
: " .strftime("%c")
autocmd bufwritepost,filewritepost *.c execute "normal
`a"
```

Step 3: Create a new *.c file with automatic header

Now, when you create a new *.c file using vim, this will automatically add the header defined in the Step1 and populate the File Name and Creation Date automatically as shown below.

```
$ vi myfile.c
/* .....
* File Name : myfile.c
```

```
* Purpose :
* Creation Date : 20-12-2008
* Last Modified :
* Created By :
.....*/
```

When you save the myfile.c file, it will automatically update the Last Modified field accordingly as shown below.

```
$ vi myfile.c

/* .....

* File Name : myfile.c

* Purpose :

* Creation Date : 20-12-2008

* Last Modified : Sat 20 Dec 2008 09:37:30 AM PST

* Created By :

.....*/
```

Explanation of the autocmd commands inside ~/.vimrc

```
$ cat -n ~/.vimrc

      1  autocmd bufnewfile *.c so
/home/jsmith/c_header.txt
      2  autocmd bufnewfile *.c exe "l," . 10 . "g/File
Name : */s//File Name : " .expand("%")
      3  autocmd bufnewfile *.c exe "l," . 10 .
```

```
"g/Creation Date :.*s//Creation Date : "  
.strftime("%d-%m-%Y")  
    4 autocmd Bufwritepre,filewritepre *.c execute  
"normal ma"  
  
    5 autocmd Bufwritepre,filewritepre *.c exe "l," .  
10 . "g/Last Modified :.*s//Last Modified :.*s//Last  
Modified : " .strftime("%c")  
    6 autocmd bufwritepost,filewritepost *.c execute  
"normal `a"
```

- **Line 1** defines the template file. This indicates that for *.c file, /home/jsmith/c_header.txt template file should be used.
- **Line 2** will search for the pattern "File Name :" from the 1st line to 10th line. If found, it will write the current filename in that line.
- **Line 3** will update the Creation Date field.
- **Line 5** will update the Last Modified field with the current date and time when you save the file.
- **Line 4 & 6:** While saving the file, the cursor will move to the "Last modified :" (because of last write operation). If you want the cursor back to the previous position then, you need to add Line 4 and 6 to the .vimrc file.
- **Line 4** will mark the current cursor position before updating.
- **Line 6** will restore the cursor position back to its previous position.

Final Note:

- Verify whether autocmd is enabled in Vim - Execute :version from Vim. If autocommand feature is enabled, it will display +autocmd.
- Autocommand help - Execute :help au from Vim, to get quick help on Vim autocmd features.

Chapter 8: Vim as a Programmers Editor

The following hacks will be of use to you for all programming languages and shell scripts.

Hack 43. Make Vim Highlight Your Code Smartly

| Command | Description |
|----------|----------------------------------|
| :syn on | Turn on the syntax highlighting |
| :syn off | Turn off the syntax highlighting |

The following screenshot shows the difference between on and off state.

| Syntax Highlighting On | Syntax Highlighting Off |
|--|--|
| <pre>// my first program in C++ #include <iostream> using namespace std; int main () { cout << "Hello World!"; return 0; } ~ ~</pre> | <pre>// my first program in C++ #include <iostream> using namespace std; int main () { cout << "Hello World!"; return 0; } ~ ~</pre> |

Hack 44. Smart Indentation

To indent a block in visual mode, do the following:

- Select the block using CTRL-V mode.
- Press > to move the block right, and < for left.

Sometime you may want to perform multiple indentations after you've selected the block. When you execute > or < , it will indent one time and the visual selection will be lost. You have to do the visual selection again to indent.

To avoid this, set the following

```
:vnoremap < <gv  
:vnoremap > >gv
```

Once you've remapped the < and > as shown above, after you indent, the block will be still selected. This way, you can keep indenting the block as many time you would like and finally press <ESC> to get out of visual mode.

Vim will do the following types of indentations depending on the settings:

- Autoindent: When you create a new line, it copies indentation from the current line.
- Smartindent: Similar to autoindent, but increases / reduces indentation when you have { / }.
- Cindent: Enables automatic C program indenting.

Hack 45. Access Unix Man page for Functions from Vim

From the Vim editor, press K on the word for which you want to read the man page.

```
K
```

To access another section man page give {n}K. For example, to access the 2nd section of the man page, do the following.

```
2K
```

Example: Sleep is a command, as well as library routine. So from a C program if you attempt to view this man page, then you need to use 3K

Customize Man page Lookup (For example, Change to Perldoc)

To look up things other than Unix man pages, specify a different keywordprg.

For example, if you are a Perl programmer, you will be using the perldoc command often for reading information about a function. So, set the following in Vim editor.

```
:set keywordprg=perldoc\ -f
```

After the above setting, when you press K on a function name it will open the perldoc instead of the Unix man page.

Hack 46. Jump to Variable Declaration

Go to the local declaration of a variable.

```
gd
```

Go to the global declaration of a variable.

```
gD
```

Hack 47. Align the Variable Assignment

The following is an example code, where variables are not aligned properly.

```
$a = 1;  
$a_very_long_variable_name_value = 1;  
$b = 1;  
$my_short_variable_value = 1;
```

Install the [Align.vim](#) plugin.

```
$ vim Align.vba.gz  
:so %  
:q
```

Visually select the text need to be aligned, and do the following.

```
: '<,'>Align =
```

After aligning, the variables will look properly aligned as shown below.

```
$a                                = 1;  
$a_very_long_variable_name_value = 1;  
$b                                = 1;  
$my_short_variable_value         = 1;
```

You can also use align as shown below:

```
:<range>Align Separator1 Separator2
```

Range can be visually selected in visual mode, or specified as line numbers (for example: 5,10)

Hack 48. Increment and Decrement Number Using CTRL Keys

| CTRL Key | Description |
|----------|--|
| CTRL-A | Increment Number. Place the cursor over a number in Vim editor and press CTRL-A, which will increase the number by 1. |
| CTRL-X | Decrement Number. Place the cursor over a number in Vim editor and press CTRL-X, which will decrease the number by 1. |

Hack 49. Execute One Vim Command in Insert Mode

When you are in insert mode, if you want to execute a single Vim command, you don't need to press <ESC> to switch to command mode.

To execute single Vim command in insert mode.

```
CTRL-O
```

The following is the sequence:

- You are in insert mode typing characters.
- Press CTRL-O, which will temporarily take you to command mode.
- Press any Vim command (for example, 5j to jump 5 lines)
- You are automatically back in insert mode after the single Vim command is executed.

Hack 50. View Current File Details

When you are editing a file, press CTRL-G or gCTRL-G to view the file details as shown below.

View basic file details.

```
CTRL-G
```

```
"test.txt" [Modified] line 3 of 6 --50%-- col 1
```

View advanced file details.

```
g CTRL-G
```

```
Col 1 of 5; Line 3 of 6; Word 3 of 6; Byte 10 of 29
```

Hack 51. Take Control of the Vim Status Bar

You can enable status bar in Vim editor to display useful information about the current file.

By default the status bar is disabled in the Vim editor. Enable the status bar as shown below.

```
:set laststatus=2
```

The following is a simple status line example:

```
:set statusline=Filename:%t\ Line:\ %l\ Col:\ %c
```

Additional status line examples from :help statusline

- :set statusline=%<%f\ %h%m%r%=%-14.(%l,%c%V%)\ %P
- :set statusline=%<%f%h%m%r%=%b\ 0x%B\ \ %l,%c%V\ %P
- :set statusline=%<%f%=\ [%1*M%*%nR%H]\ %-19(%3l,%02c%03V%)%O'%02b'
- :set statusline=...%r%{VarExists('b:gzflag','\ [GZ]')}%h...

The following are few key variables that can be used in the status line. For a complete list refer to :help statusline

- F - Full path to the file in the buffer.
- M - Modified flag, text is "+", "-" or ",".
- R - Readonly flag, text is "RO".
- H - Help buffer flag, text is "HLP".
- Y - Type of file in the buffer, e.g., "VIM". See 'filetype'.
- N - Printer page number. (Only works in the 'printhead' option.)
- L - Number of lines in buffer.
- c - Column number.
- P - Percentage through file of displayed window. This is like the percentage described for 'ruler'. Always 3 in length.

Hack 52. Change Case

The following are methods to change case of a text in Vim.

| CTRL Key | Description |
|----------|---|
| ~ | <p>Normal Mode:</p> <p>Change the case of the character under the cursor and moves the cursor to next character.</p> <p>If you keep pressing ~ , you'll keep changing the characters one-by-one until you reach the end of the line.</p> |

| | |
|----------------|---|
| | Visual Mode: This will the change case of all the highlighted text. |
| 5~ | Change the case of the next 5 characters |
| g~{motion-key} | Change the case of characters from under the cursor through the entire specified motion. For example: g~\$ changes the case from the cursor through the end of the line. |
| g~~ | Change the case of the entire current line |
| gUU | Change the entire current line to upper case |
| guu | Change the entire current line to lower case |
| gUaw | Change current word to upper case |
| guaw | Change current word to lower case |
| U | Visual Mode: Change current highlighted text to upper case |
| u | Visual Mode: Change current highlighted text to lower case |
| guG | Change text from current position to end of file to lower case |
| gUG | Change text from current position to end of file to upper case |

Hack 53. Spell Check

The following are spelling check commands:

| Spell check Command | Description |
|---------------------|---|
| :set spell | Start the spell check process. This will highlight all spelling mistakes in the current document. |
|]s | Jump to the next spelling mistake |
| [s | Jump to the previous spelling mistake |
| z= | Suggestions for the misspelled word. From the list, type the number to select a specific suggestion. |
| zg | Add the highlighted wrong words as a valid word. |
| :echo &spelllang | Displays the language code that is used for spell check. For example, this displays en for English |
| :set spelllang=code | If the default spellang is not set properly, use this method to set it to your language. |

Hack 54. Setup Quit Confirmation

If you forgot to save changes and execute :q you'll get the following message. In this case, you have to save the file and then do the :q again.

```
:q  
  
E37: No write since last change (add ! to override)
```

To get a save confirmation dialog when exiting Vim, do the following.

```
:confirm q  
  
Save changes to "test.txt"?  
[Y]es, (N)o, (C)ancel:
```

Note: You can map the :q command to the ":confirm q" command using the Vim map feature explained in Hack 80.

Hack 55. Use :up and Avoid :w

When you execute :w , it will save the file. The main problem with :w is that when you type :w, it will update the file timestamp even if the file has not been changed.

Luckily, :up will save the file and update the timestamp only when changes have been made to the file.

```
:up
```

Note: You can map the `:w` command to the `:up` command using the Vim map feature explained in Hack 80.

Hack 56. Edit Current Buffer Content

You can change the content of the opened buffers using `bufdo` as shown below.

Syntax:

```
:bufdo! LINERANGE ! CMD
```

For example, you can replace a pattern of text in the buffer as shown below.

```
:bufdo! 1,$ ! sed "s/pattern/replace/"
```

(or)

```
:bufdo! 1,$s/pattern/replace/
```

Hack 57. Tabs and Spaces

The following are important tabs and space related commands.

| Command | Description |
|-----------------------------|---|
| <code>:set expandtab</code> | Convert tabs to spaces automatically. For example, When you type Tab, it will be converted it to 8 spaces. |

| | |
|--------------------------------|--|
| <code>:set tabstop=4</code> | If you want Tab to be converted to 4 spaces, specify it with tabstop. |
| <code>:retab</code> | Convert all the tabs in a files to spaces based on expandtab and tabstop option. |
| <code>:set shiftwidth=4</code> | Specifies the number of spaces that should be used when you indent a line. |
| <code>:set ai</code> | <p>In insert mode, Vim will automatically indent new lines (when you hit return) to the same indent level of the current line.</p> <p>Use ^D at the beginning of the new line to decrease the indent by <shiftwidth> characters.</p> |

Chapter 9: Vim Command Line Hacks

You can view all of the available command line options for Vim as shown below.

```
$ vim -h
```

Hack 58. Open File in Read Only Mode

Open files in read only mode as shown below using the -R option.

```
# vim -R filename.txt  
  
(or)  
  
# view filename.txt
```

Get into the habit of using one of the above methods when you don't intent to edit a file. This will help you to avoid making any unintentional modifications to the file.

Hack 59. Recover Swap File Explicitly

Use option -r as shown below to list swap files from current directory, ~/tmp, /var/tmp, /tmp.

The following command shows all the swap files.

In this example, there are three swap files associated with the previous edits on file1.c, file2.txt and change-password.sql in the current directory.

```
$ vim -r

Swap files found:
  In current directory:
1. .file1.c.swp
   owned by: ramesh    dated: Sat Apr 25 06:58:49 2009
   file name: ~ramesh/file1.c
   modified: YES
   user name: ramesh   host name: ramesh-laptop
   process ID: 14374
2. .file2.txt.swp
   owned by: ramesh    dated: Sat Apr 25 07:28:49 2009
   file name: ~ramesh/file2.txt
   modified: YES
   user name: ramesh   host name: ramesh-laptop
   process ID: 14145
3. .change-password.sql.swp
   owned by: ramesh    dated: Sun Jan 11 13:11:51 2009
   file name: ~ramesh/change-password.sql
   modified: YES
   user name: ramesh   host name: ramesh-laptop
   process ID: 24686

In directory ~/tmp:
  -- none --
In directory /var/tmp:
  -- none --
In directory /tmp:
  -- none --
```

When a swap file exists and if you try to open the original file, you'll get the following message.

```
# vim file1.c
```

E325: ATTENTION

Found a swap file by the name ".file1.c.swp"

(1) Another program may be editing the same file.

If this is the case, be careful not to end up with two different instances of the same file when making changes.

Quit, or continue with caution.

(2) An edit session for this file crashed.

If this is the case, use ":recover" or "vim -r file1.c" to recover the changes (see ":help recovery").

If you did this already, delete the swap file ".file1.c.swp" to avoid this message.

Swap file ".file1.c.swp" already exists!

[O]pen Read-Only, (E)dit anyway, (R)ecover, (Q)uit,
(A)bort:

You will normally only see the above message for one of the following reasons:

- Someone else is editing the file currently
- A previous edit session crashed.

Based on why this happened do one of the following as prompted by Vim:

- Open Read Only (viewing the content of the file)
- Edit anyway (editing the content of the file)
- Recover (replacing the content of the file with the content of the swap file)
- Quit or Abort

Hack 60. Execute any Vim Command when opening a file

By using the option `-c` you can execute any Vim command when opening a file.

In the following example, after Vim opens the file, it will jump to line 50.

```
$ vim -c ':50' filename.txt
```

You can also execute multiple Vim commands from the command line:

```
$ vim -c '<command 1>' -c '<command 2>' <filename>
```

Hack 61. Execute Commands Stored in a File

When you find yourself executing the same sequence of Vim commands frequently, you can store them in a file and execute them as shown below.

```
$ vim -w repetitive_task.txt file_to_edit.txt
```

Hack 62. Skip Loading Plugins Temporarily

While opening a file, you can temporarily stop loading all plugins only during that particular file editing as shown below.

```
$ vim --noplugin filename.txt
```

Hack 63. Enter Restricted Mode in Vim

You can enter restricted mode in Vim using one of the following methods.

```
$ vim -Z filename  
  
(or)  
  
$ rvim filename
```

From :help -Z

Restricted mode (-Z): All commands that make use of an external shell are disabled. This includes suspending with CTRL-Z, ":sh", filtering, the system() function, backtick expansion, etc.

Chapter 10: gVim Hacks

gVim is an X-window based interface to the Vim editor.

Hack 64. Display and Hide gVim Menu and Toolbar

Sometimes to get more screen real estate you may want to disable gVims menu bar, tool bar, scroll bar or other visual components.

This can be achieved using `:set guioptions`.

For example, to disable the toolbar do the following. Please note that there is a - symbol before the =

```
:set guioptions-=T
```

To enable the toolbar do the following. Please note that there is a + symbol before the =

```
:set guioptions+=T
```

You can manipulate the following gVim GUI elements.

| UI Element Code | Description |
|---------------------------------------|-------------------------------|
| <code>:set guioptions+=TmrIRL</code> | Display all gVim GUI elements |
| <code>: set guioptions-=TmrIRL</code> | Hide all gVim GUI elements |

| | |
|--------------------|--|
| :set guioptions-=T | Hide gVim Toolbar |
| :set guioptions-=m | Hide gVim Menu Bar |
| :set guioptions-=r | Hide gVim Right side scroll bar |
| :set guioptions-=l | Hide gVim Left side scroll bar |
| :set guioptions-=R | Hide gVim Right side scroll bar that appears when window is split vertically |
| :set guioptions-=L | Hide gVim Left side scroll bar that appears when window is split vertically |

Hack 65. Adding a Custom Menu or Menu Items to gVim

You can add your own menu and menu item to gVim for custom operation.

New Menu Item under an Existing Menu

For example, you can add the following two menu items under the Tools menu

- Hide Tool Bar - This will hide the tool bar
- View Tool Bar - Once you've hidden the tool bar, use this to get it back.

To add the Tools -> Hide Tool Bar menu item, do the following:

```
:amenu Tools.&Hide-Tool-Bar :set guioptions-=T<cr>
```

To add Tools -> View Tool Bar menu item, do the following:

```
:amenu Tools.&View-Tool-Bar :set guioptions+=T<cr>
```

If you Press Alt+T to drop down the tools menu, you can see H and V highlighted. This is because we placed an & in front of H and V when we defined the menu items.

So, you can use the following shortcuts to invoke the custom menu items you've defined.

- ALT+T H to Hide Tool Bar
- ALT+T V to View Tool Bar

New Top Level Menu Bar

The following example will add a new menu bar called Bookmark (with Alt+K key) and a menu item called "Windows Explorer" (with Alt+K E) that will launch Windows Explorer.

```
:amenu <silent>Bookmar&k.Windows\ &Explorer  
:!explorer<cr>
```

Note: The above command is one command that should be typed in a single line. There should be a space after &Explorer.

Hack 66. Change Font in gVim

You might not like the default font of the gVim. You can change it using one the following two methods:

Method 1:

This example sets the font type to **Courier New** and size to **10**. For this method, you should already know the name of the font.

```
:set guifont=Courier\ New:h10
```

Method 2:

The following will launch a font chooser UI, where you can select the font type and size.

```
:set guifont=*
```

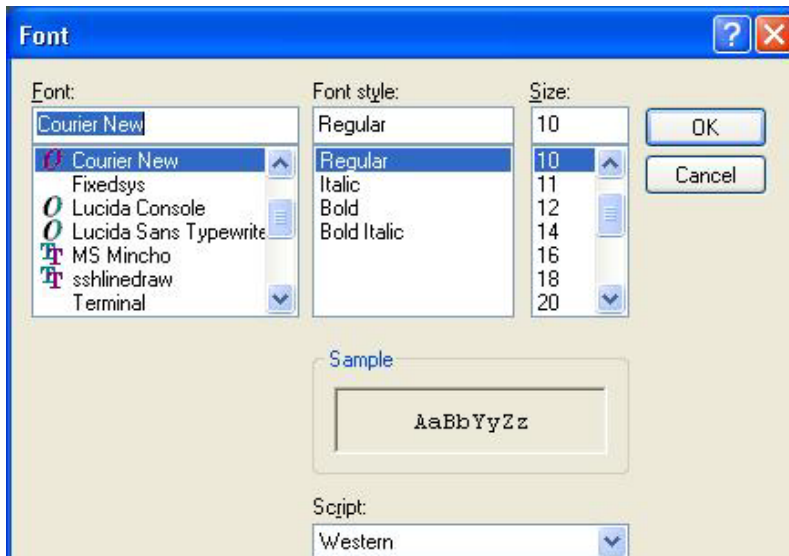


Fig: Font chooser on Windows gVim

Chapter 11: Vim Look and Feel, Tabs, and Windows

Hack 67. Split Windows Horizontally and Vertically

Split the current Window

To split the current window into two windows horizontally, do the following.

```
:split
```

To split the current window into two windows vertically, do the following.

```
:vsplit
```

To close a window that was created by split, type :q to close that window.

```
:q
```

Open a different file in another window

If you are viewing /etc/passwd and you would like to open /etc/group using a split horizontal view, do the following.

```
:split /etc/group
```

If you are viewing `/etc/passwd` and you would like to open `/etc/group` using a split vertical view, do the following.

```
:vsplit /etc/group
```

Navigate Between Windows

To navigate between windows do the following:

```
CTRL-W {Navigation key - j, k, h, l}
```

For example, to jump to the window above, do the following.

```
CTRL-W k
```

To jump to the window below, do the following.

```
CTRL-W j
```

Resize Split Windows

When you have split windows inside the Vim editor, you can enlarge or reduce the size of the current window as shown below.

| CTRL Key | Description |
|----------|---|
| CTRL-W + | Increase the size of current window in split mode |
| CTRL-W - | Reduce the size of current window in split mode |

Set the Size of a Window

To open a vertical split with 25 columns do the following

```
:25 vsplit filename.txt
```

To open a horizontal split with 3 lines do the following

```
:3 split filename.txt
```

Hack 68. Change Window Title

To change the text displayed in the title, do the following.

```
:set title titlestring=My\ Favorite\ File
```

The above will change the text in the Vim window title bar to "My Favorite File".

Note: The following example is shown in the Vim documentation, when you do :help titlestring

```
:set title titlestring=%<%F%=%l/%L-%P titlelen=70
```

- %F - Name of the file in the current window

- %l - Line number of the line where the cursor is located
- %L - Total number of lines in the file
- %p - percentage of the file. For example, if the cursor is at the middle of the file, this will show 50%

Hack 69. Change Vim Colors

First view all available color schemes in your Vim editor

```
:!ls $VIMRUNTIME/colors  
  
blue.vim      delek.vim    evening.vim  
murphy.vim    README.txt  slate.vim  
darkblue.vim  desert.vim  koehler.vim  
pablo.vim     ron.vim     torte.vim  
default.vim   elflord.vim morning.vim  
peachpuff.vim shine.vim    zellner.vim
```

For example, if you see blue.vim or evening.vim listed, you can change to those color schemes as shown below.

```
:colorscheme evening  
  
(or)  
  
:colorscheme blue
```

You can also download additional color scheme and put it under \$VIMRUNTIME/colors. Get those from http://www.vim.org/scripts/script_search_results.php?keywords=&script_type=color+scheme&order_by=rating

You can check <http://code.google.com/p/vimcolorschemetest/> for screenshots of available Vim color schemes before downloading them.

Hack 70. Edit Multiple Files in Tabs

One efficient way of editing multiple files in a single Vim session is by using tabs.

Open multiple files from the command line.

```
$ vim -p file1 file2 file3
```

The following screen shot shows three files opened in tabs

```
$ vim -p helloworld.cc employee.txt /etc/passwd
```



Fig: Three files opened in tabs

Once the files are opened in tabs, you can use any one of the following tab commands.

| Tab Command | Description |
|-------------------------------------|---|
| :tabedit FILENAME :tabe FILENAME | Open another file in a new tab under current Vim session. |
| :tabs | List all open tabs |
| :tabn N | Go to Nth tab |
| :tabclose :tabc | Close the current tab |
| :tabdo CMD | Execute a command in all tabs |
| :tabn | Go to the next tab |
| :tabp | Go to the previous tab |

Chapter 12: Additional Features in Vim Editor

Hack 71. Repeat an Operation N number of times

The "repeat operation" capability works with virtually all Vim operations. For example, to move down 10 lines at a time, you can type 10j as shown below.

```
10j
```

The following are few repeat commands.

| Repeat Command | Description |
|----------------|---|
| @@ | Repeat previously executed macro |
| n | Repeat the search in same direction |
| N | Repeat the search in opposite direction |
| . | Repeat the last edit command |
| @: | Repeat the last command line |

Hack 72. Undo and Redo Action

Single Undo:

To undo one change, do the following:

```
u
```

Multiple Undo:

To undo multiple changes, do the following. The example below will undo 5 changes.

```
5u
```

Undo All:

To undo all latest changes (in the current line), do the following.

```
U
```

Redo Action

If you've performed an undo by mistake, you can redo the change as shown below.

```
:red
```

```
(or)
```

```
CTRL-R
```

Hack 73. Open the File whose Name is under the Cursor

This hack is helpful in the following situations:

- To verify the filenames given inside configuration files are valid.
- While editing a text document, if you want to go to another file whose name is specified in the text document.
- While editing source code, to visit a local file which is included or imported by filename reference.

Open a file (in the same window) whose name is currently under the cursor.

```
gf
```

Open a file (in a new window) whose name is currently under the cursor.

```
CTRL-W f
```

Open a file (in a new tab) whose name is currently under the cursor.

```
CTRL-W gf
```

If the filename under the cursor doesn't have a full or relative path specification included, Vim will search for the file under the current directory.

For certain files, Vim will open the file even without the full path as Vim knows where to locate those files. For example,

- Header file included in a C program
- Perl module included in a Perl program

Hack 74. Edit Multiple Files Using the Traditional Method

Using this hack you can edit multiple files in a single Vim session.

Open multiple files from command line.

```
$ vim file1 file2 file2
```

Open another file when you are already in a Vim session.

```
:e another_file
```

List all open files in the current Vim session.

```
:ls
 1 %a      "helloworld.cc"                line 1
 2         "employee.txt"                 line 0
 3         "/etc/passwd"                   line 0
```

Go to the Nth file from the above :ls output.

```
:e #N
```

Toggle between two files.

```
CTRL-^
```


Moving Between Files While Editing Multiple Files

Go to next file when multiple files are opened.

```
:next
```

Go to previous file when multiple files are opened.

```
:previous
```

Hack 75. Saving Files Automatically

Vim normally will give an error message if you have unsaved changes when you try to switch buffers or files.

To enable the automatic writing of files when switching buffers/files, do the following.

```
:set autowrite
```

Write all files using a single command. (this can be VERY useful in macros)

```
:wall
```

Hack 76. Encrypt File in Vim

Save and encrypt the current file:

```
:X  
  
Enter encryption key: *****  
Enter same key again: *****
```

Once you've encrypted a file using :X, the next time you open that file, Vim will prompt for the encryption key.

Hack 77. Save and Resume Vim Sessions

When you are editing files in a Vim session, if you have to perform some other task you may have to close all files and Vim sessions. However after a while you may want to come back and continue the Vim sessions exactly where you left off earlier.

In that case, you should save the Vim sessions, which will save your current settings such as:

- Buffers
- Window size
- Custom Options
- Folds
- Current directory

Vim 101 Hacks

www.thegeekstuff.com

All of the above information will be stored when you save your session. You can also customize and decide which options you prefer to have stored by the save session command.

To save the current session, do the following.

```
:mksession
```

When you have N files opened along with folds, multiple options set, different directory, window size customization, mksession will save all those things.

To open a saved session, do the following.

```
$ vim -S Session.vim
```

| Session Command | Description |
|---------------------|---|
| :mksession | Creates a new session with the default file name Session.vim in the current working directory. |
| :mksession filename | Saves the session in the specified filename in the current working directory. |
| \$ vim -S | Opens the default saved session. i.e., Opens Session.vim in the current directory. |
| \$ vim -S filename | Opens a session using the session filename in the current directory. |
| :source Session.vim | applies all session settings from a particular session file after you are already in the Vim editor |

Hack 78. Execute Unix Shell Command Inside Vim

To execute a Unix shell command from inside the Vim editor, do the following.

```
:!unix-command  
  
:!ls  
  
:!date
```

You can also pass the current file name as a parameter to the Unix command using the following methods.

Let us assume that you are running all the following commands when you have /etc/sysctl.conf file open in the Vim editor:

| Command | Description |
|------------------------------------|---|
| :!echo % sysctl.conf | % will pass the current file name to the Unix command. For example, :!ls -l % -- This will execute ls -l on the current file in Vim editor |
| :!echo %:p /etc/sysctl.conf | %:p will pass the full path name of the file to the Unix command |
| :!echo %:e conf | %:e will pass the extension of the file to the Unix command |

Hack 79. Review the Differences between Files using Vimdiff

Similar to the Unix diff command, vimdiff is used to show the difference between files. Unlike the Unix diff command, vimdiff is more colorful and user-friendly.

Unix diff command text output:

```
$ diff employee.txt new-employee.txt

1c1
< 100 Jason Smith   Developer
---
> 100 Jason Smith   Senior Developer
4a5,7
> 500 King James    Manager
> 600 Raj Patel     Team Lead
> 700 Emily Jacob   HR
```

Vimdiff visual output:

In the following example, it is very easy to visually see what was changed and added between the two files.

```
$ vimdiff employee.txt new-employee.txt
```

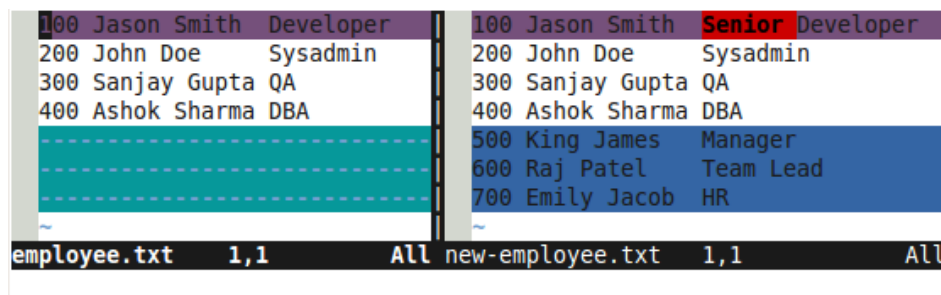


Fig: vimdiff showing visual diff

| Vimdiff Command | Description |
|---|---|
| \$ vimdiff file1 file2 \$ vim -d file1 file2 | Show the diff using a vertical window split |
| \$ vimdiff -o file1 file2 \$ vim -d -o file1 file2 | Show the diff using a horizontal window split |
| \$ vim file1 :diffsplit file2 | If you are already inside a file, use :diffsplit to launch horizontal diff |
| \$ vim file1 :vert diffsplit file2 | If you are already inside a file, use :vert diffsplit to launch vertical diff |
| [c | Go to the next change inside vimdiff |
|]c | Go to the previous change inside vimdiff |

Hack 80. Vim Map Command

Using the Vim Map feature, you can map a key to a particular job that you perform repeatedly.

Create a Map in Vim

In the following example, anytime you type `:write` , it will compile the current open *.c program file and execute the `./a.out`, if the compilation is successful.

```
:map :write :!cc % && ./a.out
```

```
/* Hello World C Program */  
  
#include<stdio.h>  
  
main()  
{  
    printf("Hello World!");  
}  
~  
~  
:map :write :!cc % & ./a.out
```

Fig: Create a map

- `:map` - Vim command to create the map
- `:write` - Name of the map (map-name)
- `:!cc % & ./a.out` - The command that should be executed when the map-name is called.

Execute the map

To execute the map, call the name of the map. In the example shown above, `:write` is the name of the map.

When you type `:write`, this will be replaced with `:!cc % & ./a.out` and then Vim will use `$SHELL` to compile the C program and execute the `a.out`.

Additional Map Examples

You can map the `:w` command to the `:up` command as shown below.

```
map :w :up
```

You can map the `:q` command to `:confirm q` as shown below.

```
:map :q :confirm q
```

Display Defined maps

Type `:map` to display all the defined maps as shown below.

```
:map

:write          :!cc % & ./a.out
<xHome>         <Home>
<xEnd>          <End>
<S-xF4>         <S-F4>
<S-xF3>         <S-F3>
<S-xF2>         <S-F2>
<S-xF1>         <S-F1>
<xF4>           <F4>
<xF3>           <F3>
<xF2>           <F2>
<xF1>           <F1>
```


Hack 81. Make Bash Shell work like Vim Editor

Once you've become familiar with Vim shortcuts, you may want to use the same from your Unix command prompt.

By default the bash command line editing is done with emacs keystrokes.

To use Vim keystrokes for command line editing, set the following.

```
$ set -o vi
```

After you perform `set -o vi`, bash will act virtually like it is in insert mode. Press ESC to go to command mode. From here you can execute most Vi commands to perform command line editing.

Make this change permanent by setting this option in your `.bashrc`.

```
$ cat ~/.bashrc  
set -o vi
```

Execute the following to disable Vim mode and go back to emacs mode.

```
$ set -o emacs
```

Hack 82. Set Vim Options

There are lots of set options available in Vim editor. The following are some of the key options.

| Option | Description |
|----------------|--|
| :set nu | Show line numbers |
| :set ic | Ignore case when searching |
| :set ro | Disable writing of files by Vim |
| :set wm=n | Set right-most margin column, after which Vim will wrap words as best it can |
| :set ai | Turn on auto indentation |
| :set all | Display all settings of your Vim session |
| :set list | Displays invisible characters. For example, ^I for tab, \$ for end of line. |
| :set hlsearch | Highlight matched patterns |
| :set incsearch | Activates Incremental search mode |

Hack 83. Unset Vim Options

Add the prefix "no" before an option to unset it. Do the following to turn off a particular option.

```
:set no<OPTION>
```

To display line numbers,

```
:set nu
```

To unset the display line number option.

```
:set nonu
```

Hack 84. Default registers and their uses

Anytime the text is deleted, copied, or substituted, it will be available in a register which you can access.

There are default registers that store information as explained below.

| Register Name | Description |
|---------------|-----------------------------------|
| % | Name of the current file |
| # | Name of the alternate file |
| : | Most recent executed command line |
| / | Last search pattern |
| " | Last used registers |

To paste the content from the register, execute the following.

```
"<Register Name>p
```

For example, to paste the name of the file as text inside the file, do the following in normal mode.

```
"%p
```

Hack 85. Numeric Registers and Recovering Deletes

The following are a few important points about registers.

- There are 10 registers available that are numbered from 0 through 9
- Most recent yank (copy) is stored in register 0
- Most recent deletion is stored in register 1

With each deletion, Vim shifts the previous content from 1 to 2, 2 to 3 and so on.

Hack 86. Vim Directory Operation

You can use Vim editor as a file manager to navigate the filesystem and perform various operations as explained in this hack.

```
# vim DIRNAME
```

The following command will show a list of all directories and files in the /etc/ directory inside vim.

```
# vim /etc/

" =====
" Netrw Directory Listing                      (netrw v125)
"   /etc
"   Sorted by      name
"   Sort sequence:
"   Quick Help:
" =====
../
NetworkManager/
X11/
acpi/
alchemist/
alsa/
alternatives/
audisp/
audit/
...
```

You can perform the following actions within the Vim file explorer.

| Key | Description |
|---------|--|
| <ENTER> | Open the file under the cursor Go to the directory under the cursor |
| D | Delete the file under the cursor |
| R | Rename the file under the cursor |
| X | Execute the file under the cursor |

| | |
|---|--------------------------------|
| o | Open a horizontal split window |
|---|--------------------------------|

You can also launch the Vim file explorer from within Vim using any one of the following methods.

| Key | Description |
|-------------------------|---|
| :Ex (mnemonic: Explore) | Open current directory in Vim file explorer |
| :Ex ~/etc/ | Open a specific directory in Vim file explorer |
| :Sex | Open current directory in Vim file explorer in horizontal split windows |
| :Vex | Open current directory in Vim file explorer in Vertical split windows |
| :Tex | Open current directory in Vim explorer in a new Tab |

Chapter 13: Power of Search

Hack 87. Navigation by Search

Execute */search-term* to search for the first occurrence of keyword from the current position.

| Navigation Key | Description |
|----------------|--|
| / | Search forward (also used as Find next) |
| ? | Search backward (also used as Find previous) |

When you do */search-term* , it will search for the first occurrence of search-term from the current position. After that, you only need to type *n* or *N* to search for the same search-term again. These are similar to 'Find Next' and 'Find Previous' in many applications.

| Navigation Key | Description |
|----------------|---|
| n | Go to the next occurrence (Find Next) |
| N | Go to the previous occurrence (Find Previous) |
| // (or) ?? | Repeat previous forward or reverse search |

Hack 88. Go to Next / Previous Occurrence of the Current Word

Sometimes you may want to search for the next occurrences of the keyword where your cursor is located.

| Navigation Key | Description |
|----------------|---|
| * | Go to the next occurrence of current word under the cursor. |
| # | Go to the previous occurrence of current word under the cursor. |

Tip: This is a little different from the previous hack. You don't need to first search for the word and then do *n* or *N*. Instead you can type * or # without first searching for the word where the cursor is located.

You can also do a partial search for the word that is currently under the cursor.

| Navigation Key | Description |
|----------------|--|
| g* | Go to the partial match of next occurrence of current word under the cursor. For example, if current word is 'top', g* will also match next 'stop', 'laptop'. |
| g# | Go to the partial match of previous occurrence of current word under the cursor. |

To list all the occurrences of the word under the cursor, use [I (upper-case i)

```
[I
```

Hack 89. Search for a Character within a Line

Use the following keys to navigate within a line.

| Navigation Key | Description |
|----------------|---|
| fX | Go to character X within a line in forward direction |
| FX | Go to character X within a line in reverse direction |
| tX | Go to one character before character X within a line in forward direction |
| TX | Go to one character before character X within a line in forward direction |
| ; | Repeat latest f, F, t or T in forward direction |
| , | Repeat latest f, F, t or T in backward direction |

Hack 90. 12 Powerful Find and Replace Examples

In this hack, let us review how to perform both basic and advanced text and pattern substitution features in the Vim Editor. These features are explained using 12 very practical and powerful text substitution examples.

Syntax of the text substitution command inside the Vim editor:

```
: [range] s [ubstitute] / {pattern} / {string} / [flags] [count]
```

The following are three possible substitution flags.

- [c] Confirm each substitution.
- [g] Replace all occurrences in the line.
- [i] Ignore case for the pattern.

Example 1: Substitute all occurrences of a text with another text in the whole file

This is a very common usage of the text substitution feature inside Vim. When you want a specific text to be replaced with another text everywhere in the entire file then you can use the following sequence.

```
:%s/old-text/new-text/g
```

- %s - specifies all lines. Specifying the range as '%' means do substitution in the entire file.
- g - specifies all occurrences in the line. If this 'g' flag is not used then only first occurrence in the line only will be substituted.

Example 2: Substitution of a text with another text within a single line

Lack of range specification means do substitution on the current line only. With the 'i' flag, you specify case insensitive searching.

```
:s/helo/Hello/gi
```

Example 3. Substitution of a text with another text within a range of lines

You can specify only a range of lines to be affected in the substitution. Specifying 1, 10 as the range limits substitution to only lines 1 - 10.

```
:1,10s/I/We/g
```

Example 4. Substitution of a text with another text by visual selection of lines

You can control the substitution range by visually selecting specific lines. Press CTRL + V in command mode, use navigation keys to select the part of the file you want to be used as range. Press ':' which will automatically change to :<,>

Once it automatically changes to :<,> , you can start typing the rest of the command as shown below.

```
: '<,'>s/helo/hello/g
```

Example 5. Substitution of a text with another text in only the next N number of lines

Specifying a count argument at the end of a substitution specifies that the substitution should be applied over the next {count} lines. For example, to do a substitution over the next 4 lines from the current line, you would enter:

```
:s/helo/hello/g 4
```

Example 6. Substitute only whole words and not partial matches

Let us assume that you want to change only the whole word 'his' to 'her' in the original text mentioned below. If you do the standard substitution, apart from changing his to her, it will also change This to Ther as shown below.

Standard Substitution:

```
Original Text: This is his idea  
:s/his/her/g  
Translated Text: Ther is her idea
```

Whole Word Substitution:

```
Original Text: This is his idea  
:s/\<his\>/her/  
Translated Text: This is her idea
```

Note: You should enclose the word with < and > , which will force the substitution to search only for the full word and not any partial match.

Note 2: novices sometimes use spaces instead of < and > , not realizing this won't match words at the beginnings and ends of lines or words with proximate punctuation.

Example 7. Substitute either word1 or word2 with a new word using a regular expression

In the following example, Vim will replace any occurrences of either good or nice with awesome.

```
Original Text: Linux is good. Life is nice.
```

```
:%s/\(good\|nice\) /awesome/g
```

```
Translated Text: Linux is awesome. Life is awesome.
```

You can also do full-word substitution by specifying regular expression. The following example does the substitution of hey or hi with hai. Please note that this does not do any substitution within larger words such as 'they', 'this'.

```
:%s/\<\(hey\|hi\) \>/hai/g
```

- \< - word boundary.
- \| - "logical or" (in this case hey or hi)

Example 8. Interactive Find and Replace in Vim Editor

You can perform an interactive search and replace using the 'c' flag of the substitute command, which specifies to ask for confirmation to do substitution or not, as explained below. In this example, Vim editor will do a global search for the word 'awesome' and replace it with 'wonderful'. However, it will do the replacement only based on your input as explained below.

```
:%s/awesome/wonderful/gc
```

```
replace with wonderful (y/n/a/q/l/^E/^Y)?
```

- y - Will replace the current highlighted word. After replacing it will highlight the next word that matched the search pattern

- **n** - Will not replace the current highlighted word. But it will automatically highlight the next word that matched the search pattern
- **a** - Will substitute all of the remaining matches without further prompting.
- **I** - This will replace only the current highlighted word and terminate the find and replace effort.

Example 9. Prepending every line with its line number

When the replacement string starts with '**\=**', it should be evaluated as an expression. Using the '**line**' function we can get the current line number.

```
:%s/^/\=line(".") . ". "/g
```

Note: This is different from "**:set number**" which does not write the line numbers into the file. When you use this substitution you are making these line numbers part of the actual content of the file.

Example 10. Substituting a special character with an equivalent value.

Substituting the **~** character with the value of the **\$HOME** variable.

```
Original Text: Current file path is ~/test/
```

```
:%s!\~!\= expand($HOME)!g
```

```
Translated Text: Current file path is  
/home/ramesh/test/
```

Notes:

- You can use the **expand** function to expand any available predefined or user defined variables.

- We use ! instead of / above because the value of \$HOME will contain at least one / which will confuse the substitution. There are many characters that can be used instead of /

Example 11. Alter the sequence number in a numbered list when inserting a new item

Assume that you have a numbered list like the following inside a text file. In this example, let us assume that you want to add a new line after Article 2. For this, you should change the number of all other articles accordingly.

```
vi / vim tips & tricks series
Article 1: Vi and Vim Editor: 3 Steps To Enable Thesaurus Option
Article 2: Vim Autocommand: 3 Steps to Add Custom Header To Your File
Article 3: 5 Awesome Examples For Automatic Word Completion Using CTRL-X
Article 4: Vi and Vim Macro Tutorial: How To Record and Play
Article 5: Tutorial: Make Vim as Your C/C++ IDE Using c.vim Plugin
Article 6: How To Add Bookmarks Inside Vim Editor
Article 7: Make Vim as Your Bash-IDE Using bash-support Plugin
Article 8: 3 Powerful Musketeers Of Vim Editor ? Macro, Mark and Map
Article 9: 8 Essential Vim Editor Navigation Fundamentals
Article 10: Vim Editor: How to Correct Spelling Mistakes Automatically
Article 11: Transfer the Power of Vim Editor to Thunderbird for Email
Article 12: Convert Vim Editor to Beautiful Source Code Browser
```

3rd Article “Make Vim as Your Perl IDE Using perl-support.vim Plugin” got missed. So when you want to add it, then you want to change “Article 3” to “Article 4”, “Article 4” to “Article 5”, up to “Article 12” to “Article 13”.

This can be achieved by the following Vim substitution command.

```
:4,$s/\d\+/\=submatch(0) + 1/
```

- Range: 4,\$ - From the 4th line through the last line. (the 4 is manually determined)
- Pattern to Search - \d\+ - A string of digits

- Pattern to Replace - `\=submatch(0) + 1` - gets the matched pattern and adds 1 to it.
- Flag - as there is no flag, by default it substitutes only the first occurrence in each line.

After executing the substitute statement the file will look like this; then you can add the 3rd Article.

vi / vim tips & tricks series

Article 1: Vi and Vim Editor: 3 Steps To Enable Thesaurus Option

Article 2: Vim Autocommand: 3 Steps to Add Custom Header To Your File

Article 4: 5 Awesome Examples For Automatic Word Completion Using CTRL-X

Article 5: Vi and Vim Macro Tutorial: How To Record and Play

Article 6: Tutorial: Make Vim as Your C/C++ IDE Using c.vim Plugin

Article 7: How To Add Bookmarks Inside Vim Editor

Article 8: Make Vim as Your Bash-IDE Using bash-support Plugin

Article 9: 3 Powerful Musketeers Of Vim Editor ? Macro, Mark and Map

Article 10: 8 Essential Vim Editor Navigation Fundamentals

Article 11: Vim Editor: How to Correct Spelling Mistakes Automatically

Article 12: Transfer the Power of Vim Editor to Thunderbird for Email

Article 13: Convert Vim Editor to Beautiful Source Code Browser

Example 12. Capitalizing the first character of every sentence.

When formatting a document, properly capitalizing sentences is quite useful. This can be done easily with the following substitution.

```
:%s/\.\s*\w/\=toupper(submatch(0))/g
```

- `\.\s*\w` - Search Pattern - a literal . (period) followed by zero or more spaces, then a word character.
- `toupper` - converts the given text to upper case.
- `submatch(0)` - returns the matched pattern.

Text before substitution:

Lot of vi/vim tips and tricks are available at thegeekstuff.com. reading

these articles will make you very productive. following activities can be done very easily using vim editor.

- a. source code walk through,
- b. record and play command executions,
- c. making the vim editor as ide for several languages,
- d. and several other @ vi/vim tips & tricks.

Text after substitution: (changes in bold)

Lot of vi/vim tips and tricks are available at thegeekstuff.com. **Reading** these articles will make you very productive. **Following** activities can be done very easily using vim editor.

- a. **Source** code walk through,
- b. **Record** and play command executions,
- c. **Making** the vim editor as ide for several languages,
- d. **And** several other @ vi/vim tips & tricks.

Hack 91. Search across Multiple Files using vimgrep

You can search for a search term across multiple files using vimgrep command. You can execute vimgrep from inside the Vim editor as shown below.

The following example will search for the search term Jason inside all files ending in .txt in the current directory.

```
:vimgrep jason *.txt
```

By default vimgrep will jump to the first file that contains a match. Use :cn to jump to the next match.

| Key | Description |
|------------------------|---|
| :vimgrep search-term * | Search for search-term in multiple files |
| :cn | Jump to the next match of the vimgrep search |
| :cN | Jump to the previous match of the vimgrep search |
| :clist | View all the files that matched the vimgrep search keyword without jumping to the individual files. |
| :cc number | Jump to the specific search number. Pick the search number based on the :clist output |

You can also do recursive searching using vimgrep. Use ** for recursive searching as shown below.

For example, the following will search for the search term table in all *.html files under the current directory and all subdirectories.

```
:vimgrep table **/*.html
```

Hack 92. Highlight Search Results with Color

When you search for a keyword, you may want to automatically highlight all the matches. Use :hlsearch option for this.

To enable search result highlight

```
:set hlsearch
```

After this when you search for a keyword using /keyword, all the matches of the keyword will automatically be highlighted in the current file.

| Key | Description |
|-----------------|--|
| :set hlsearch | Enable search result highlighting Note: Add this to ~/.vimrc to make it permanent |
| :set nohlsearch | Disable search result highlighting |
| :nohlsearch | Clear the active search highlights |

Hack 93. Vim Incremental Search

Once you get used to incremental search, you cannot live without it in Vim editor.

To enable Incremental search,

```
:set incsearch
```

Incremental search will start searching for the keyword as soon as you start typing.

To disable incremental search,

```
:set noincsearch
```

Hack 94. The Power of :match

Use :match to display all instances of a particular keyword in a certain color scheme. For example, if you want to highlight the keyword Error in red, use the following

```
:match ErrorMsg /Error/
```

In the above example:

- :match - the match command
- ErrorMsg - Predefined color scheme (red) available in the Vim.
- /Error/ - search pattern defined by user

The following are few predefined color schemes available in Vim:

- ErrorMsg
- WarningMsg
- ModeMsg
- MoreMsg

You can create your own Color scheme as shown below.

```
:highlight custom-color-scheme ctermbg=COLOR  
ctermfg=COLOR
```

Once the custom color scheme is created, you can use it as shown below.

```
:match custom-color-scheme /\cPATTERN/
```

Chapter 14: Automatic Completion

Hack 95. Automatic Word Completion

You can perform automatic word completion in Vim using CTRL-X in insert or append mode. By typing the first few characters of a word you can get the whole word either from a dictionary, or a thesaurus, or even from the words that are already present within the file that you are editing.

You can use the following Vim shortcut keys to choose between the existing expansions for a given word.

| Key | Description |
|---------------|----------------------------|
| CTRL-X CTRL-N | Word completion - forward |
| CTRL-X CTRL-P | Word completion - backward |

1. Word / pattern completion
2. Line completion
3. Dictionary word completion
4. File name completion

Using Vim editor is lot of Follow

~
~
~
~
~
~
~

Follow

Fun
Frighten
Fry
Friend
File

Fig: Keyword completion in Vi / Vim using CTRL-X CTRL-N

Hack 96. Automatic Line Completion

If you want to insert a copy of an existing line, type the first few words/characters of the line, and then press the Vim shortcut keys "CTRL-X CTRL-L", which will display all the lines matching that pattern.

CTRL-X CTRL-L

```
merge_dbd_config,  
dbd_cmds,  
dbd_hooks  
};  
  
dbd gr  
dbd_group_t *next;  
dbd_group_t *group = params;  
dbd_group_t *group = data;  
dbd_group_t *group;  
dbd_group_t *group = svr->group;
```

-- Whole line completion (^L^N^P) match 1 of 23

Fig: Vim Whole line completion using CTRL-X CTRL-L

Hack 97. Automatic Filename Completion

Insert the name of any file that the current users can see anywhere on the Linux system using the short cut key "CTRL-X CTRL-F".

CTRL-X CTRL-F


```
# Backup destination location
DESTINATION="/home/backup"

# Backup source location
SOURCE="/etc/DIR_COLORS"
~ /etc/DIR_COLORS
~ /etc/DIR_COLORS.xterm
~ /etc/Muttrc
~ /etc/Muttrc.local
~ /etc/NetworkManager/
~ /etc/X11/
~ /etc/a2ps-site.cfg
~ /etc/a2ps.cfg
~ /etc/acpi/
~ /etc/adjtime
~ /etc/alchemist/
~ /etc/aliases
-- File name completion (^F^N^P) match 1 of 222
```

Fig: Vim File name completion using CTRL-X CTRL-F

Hack 98. Dictionary Completion

Enable the dictionary in Vim by adding the following line to ~/.vimrc.

```
$ cat ~/.vimrc
set dictionary+=/usr/share/dict/words
```

This is a great feature when you stumble for the correct spelling for a word that you are typing. After typing the first few characters, Press the Vim short cut key CTRL-X CTRL-K to display the matching dictionary words.

CTRL-X CTRL-K

```
start()  
{  
# Create keys if necessar  
~      necessar /usr/share/dict/words  
~      necessarian /usr/share/dict/words  
~      necessarianism /usr/share/dict/words  
~      necessities /usr/share/dict/words  
~      necessarily /usr/share/dict/words  
~      necessariness /usr/share/dict/words  
~      necessarium /usr/share/dict/words  
~      necessarius /usr/share/dict/words  
~      necessary /usr/share/dict/words  
-- Dictionary completion (^K^N^P) match 1 of 9
```

Fig: Vim Dictionary word completion using CTRL-X CTRL-K

Hack 99. Thesaurus Word Completion

This hack explains how to use Vim effectively by enabling a thesaurus in three steps.

Step 1: Define a Thesaurus file

Any number of synonyms can be listed together on a single line using either spaces to separate the words or comma delimiting if some of the synonyms are multi-word phrases. For example, you can create your own thesaurus file as shown below for the word "important".

```
$ vim /home/jsmith/mythesaurus.txt
important, valuable, substantial, significant
```

Step 2: Specify Thesaurus File Location in ~/.vimrc

Add the following line to .vimrc specifying the location of thesaurus file.

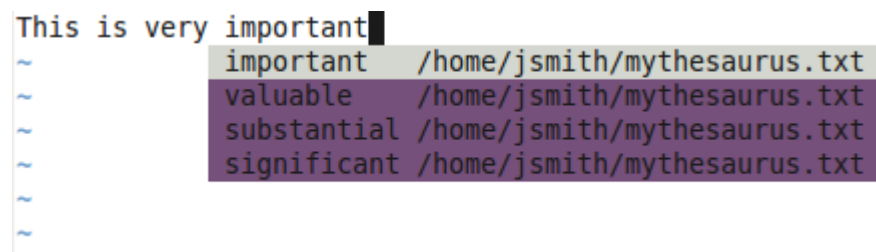
```
$ vim ~/.vimrc
set thesaurus+=/home/jsmith/mythesaurus.txt
```

Step 3: Use Thesaurus While Editing Document Using CTRL-X CTRL-T

From vim, if you want to use an alternative word, press CTRL x + CTRL t in the insert mode.

For example, when you've typed the word "important", press CTRL-X and CTRL-T, which will show a popup with the alternate words "valuable", "substantial" and "significant" as shown below from your /home/jsmith/mythesaurus.txt file.

CTRL-X CTRL-T



The screenshot shows a Vim editor window with the text "This is very important" on the first line. A popup menu is displayed over the word "important", listing four alternative words from the thesaurus: "important", "valuable", "substantial", and "significant". Each word is followed by the file path "/home/jsmith/mythesaurus.txt". The popup menu has a light green header bar and a dark purple body. The word "important" is highlighted in the list. The editor's status bar at the bottom shows "~".

Fig: Launch Thesaurus from Vim using CTRL-X and CTRL-T

Download and Use a Pre-defined Thesaurus

Instead of defining your own custom thesaurus, download and use the pre-defined famous moby thesaurus as shown below.

```
$ wget
http://www.gutenberg.org/dirs/etext02/mthes10.zip

$ unzip mthes10.zip
Archive:  mthes10.zip
  inflating: aaREADME.txt
  inflating: roget13a.txt
  inflating: mthesaur.txt
```

Use mthesaur.txt as the thesaurus file. It is quite large and you would get more than 50 related words for each word.

Add the following line to .vimrc to specify the location of the mthesaur.txt thesaurus file.

```
$ vim ~/.vimrc
set thesaurus+=/home/jsmith/mthesaur.txt
```

How can a programmer use thesaurus feature in vim?

This can be very helpful for programmers. For example, a PHP programmer can create a php-functions file with the following lines and specify this as thesaurus file inside the ~/.vimrc.

```
$ vim /home/jsmith/php-functions.txt
```

```
math abs acos acosh asin asinh atan atan2 atanh  
base_convert bindec ceil cos  
errors debug_backtrace debug_print_backtrace  
error_get_last error_log error_reporting  
restore_error_handler
```

Add the php-functions.txt to .vimrc specifying the location of thesaurus file.

```
$ vim ~/.vimrc  
set thesaurus+=/home/jsmith/mythesaurus.txt  
set thesaurus+=/home/jsmith/mthesaur.txt.txt  
set thesaurus+=/home/jsmith/php-functions.txt
```

Now, when you type “math” in your PHP file and press CTRL x and CTRL t, all the PHP math functions will be displayed. Also, please note that you define multiple thesaurus files as shown above.

Hack 100. Automatically open a Pop-up menu for Completion

Install and Configure the Auto Completion Popup Plugin

Download the autocmplpop.vim plugin from vim.org as shown below.

```
$ mkdir -p ~/.vim/plugin  
  
$ cd ~/.vim/plugin  
  
$ wget -O autocmplpop.zip  
http://www.vim.org/scripts/download_script.php?src_id=1  
1538
```

Enable the plugin by adding the following line to the ~/.vimrc

```
$ vim ~/.vimrc  
filetype plugin on
```

Example1: Pop-up with available word choices

After this plugin is installed, you don't need to type a command sequence to activate it - it will activate automatically. Whenever you type two characters, it will display the available word choices which start with those two characters. In this example, when sp is typed, it displays both spider and spout in the pop-up.

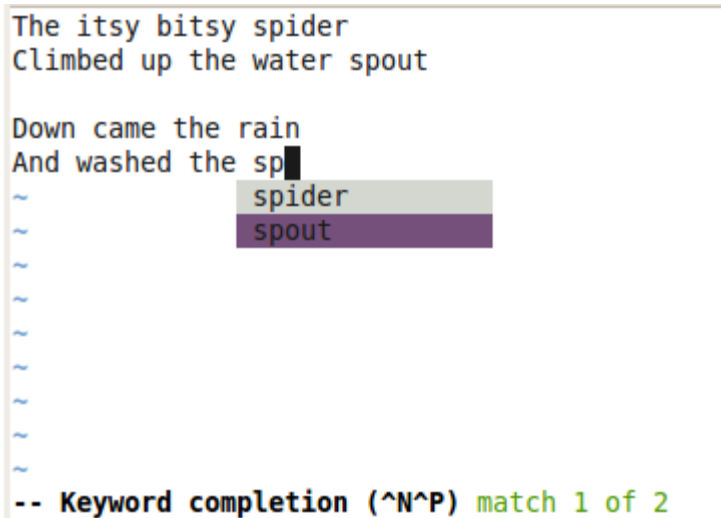


Fig: Auto completion pop-up menu with matching words

Example2: File name completion pop-up

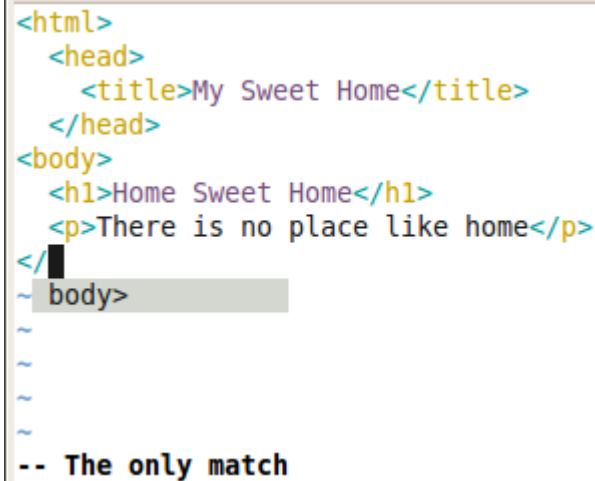
When you are typing file names inside a program, it will automatically display the pop-up with appropriate choices as shown below.

```
#!/bin/bash
#
# Shell script to backup system files
#
# Backup destination location
DESTINATION="/home/backup"
#
# Source Directory
SOURCE="/etc/"
~ /etc/ConsoleKit/
~ /etc/NetworkManager/
~ /etc/ODBCDataSources/
~ /etc/PolicyKit/
~ /etc/X11/
~ /etc/acpi/
~ /etc/adduser.conf
~ /etc/adjtime
-- match 1 of 223
```

Fig: Auto completion pop-up menu with matching file names

Example3: Omni completion for tags

This plugin does the omni completion for HTML, XHTML, CSS, Ruby, Python. In the example below, when you type `</`, it lists the available tags automatically.



The screenshot shows a Vim editor window with an HTML file. The code is as follows:

```
<html>
  <head>
    <title>My Sweet Home</title>
  </head>
  <body>
    <h1>Home Sweet Home</h1>
    <p>There is no place like home</p>
  </
~ body>
~
~
~
~
-- The only match
```

The pop-up menu is triggered by typing the closing tag for the body element, and it shows the suggestion "body>". The text "-- The only match" is displayed at the bottom of the menu.

Fig: Omni completion pop-up menu for HTML files

Hack 101. Automatically offers Word Completion as you type

When you type something in Vim editor, `word_complete.vim` plugin will display only one relevant word, and if you like what is displays, you can press TAB key to accept it's suggestion.

This is a completely non-intrusive and very effective plugin.

Install and Configure Vim Word Complete plugin

Download `word_complete.vim` plugin from vim.org as shown below.

```
# if the directory does not exist already
$ mkdir -p ~/.vim/plugin
```



```
$ cd ~/.vim/plugin  
  
$ wget  
http://www.vim.org/scripts/download_script.php?src_id=6504
```

Enable the plugin by adding the following line to the ~/.vimrc

```
$ vim ~/.vimrc  
filetype plugin on
```

Two methods to Enable Auto Complete Plugin

Method 1: Enable on an on-demand basis.

This is probably the better method, as you can enable the auto completion mode only when you need it. Once you've opened a file, execute : call DoWordComplete() - to enable this plugin temporarily.

```
$ vim mystory.txt  
  
: call DoWordComplete()
```

Method 2: Enable by default

```
$ cat ~/.vimrc  
  
:autocmd BufEnter * call DoWordComplete()
```

Once you have this plugin enabled by default, if you would like to disable it temporarily you can do the following.

```
$ vim mystory.txt

:call EndWordComplete()
```

Usage Examples of Auto Word Complete.

In the example shown below, when I typed spi, it filled in the word spider automatically.

- If you would like to accept the suggested word, press TAB.
- If you don't want to accept the suggested word, just keep typing.

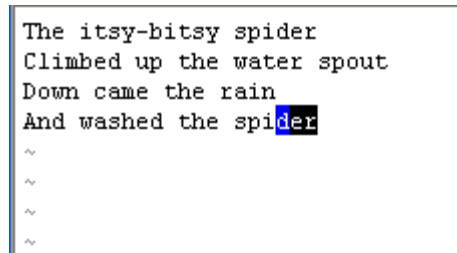
A screenshot of a Vim editor window. The text inside the editor is:
The itsy-bitsy spider
Climbed up the water spout
Down came the rain
And washed the spider
Below the text, there are four tilde (~) characters, indicating the start of a new line. The word 'spider' is highlighted in blue, and the cursor is positioned at the end of the word.
The editor has a dark background and a light blue border.

Fig: Auto completion plugin suggesting word spider.

Chapter 15: Bonus Hacks

Bonus Hack 1. Add Bullet Point Style to List of Items

Let us assume that you would like to convert the following items to bulleted list in Vim editor.

```
The Geek Stuff article categories:
```

```
Vi / Vim Tips and Tricks  
Linux Tutorials  
SSH Tips and Tricks  
Productivity Tips  
HowTo & FAQ  
Hardware Articles  
Nagios Tutorials  
MySQL and PostgreSQL Tips
```

Follow the steps below to convert the above to bulleted list.

Step 1: Go to the first character of the first item that needs to be converted to bullet list. From here go to visual mode by pressing CTRL-V

```
CTRL-V
```

Step 2: Select the first character of all the bullet points in the visual mode as shown below.

```
The Geek Stuff article categories:
Vi / Vim Tips and Tricks
Linux Tutorials
SSH Tips and Tricks
Productivity Tips
HowTo & FAQ
Hardware Articles
Nagios Tutorials
MySQL and PostgreSQL Tips
~
~
-- VISUAL BLOCK --
```

Fig: Visual Block Mode of 1st Character

Step 3: Press I (upper case i)

Step 4: Press TAB

Step 5: Insert * and press space. This will add bullet point only to the 1st item at this stage.

Step 6: Repeat it for all items by pressing ESC key twice.

```
ESC ESC
```

After above 6 steps, you'll see the bullet points as shown below.

```
The Geek stuff article categories:
```

```
* Vi / Vim Tips and Tricks
* Linux Tutorials
* SSH Tips and Tricks
* Productivity Tips
* HowTo & FAQ
* Hardware Articles
* Nagios Tutorials
* MySQL and PostgreSQL Tips
```

Bonus Hack 2. Set Vim as Universal Default Editor using update-alternatives

If you are a system administrator, you may want to set Vim as default editor for the whole system. Use update-alternatives as shown below. This works on all Debian based system.

```
# update-alternatives --set editor <PATH OF VIM>
```

Bonus Hack 3. Make Vim as Default Editor

You can set Vim as default editor as shown below.

```
$ export EDITOR=vi
```

Most of the Unix applications (For example, crontab) refer this EDITOR variable to check which editor can be used. So it is better to set this in your bashrc

Bonus Hack 4. Format a Paragraph

To format a paragraph, do the following.

```
gqap
```

Bonus Hack 5. Edit Macros for Reuse

After you've recorded a macro, if you found that there is a mistake in the macro, you have following two options:

- Record the macro again.
- Edit the macro and correct only the mistake.

The following three steps explain how to edit the macro and correct the mistake (Instead of recording the macro again)

Step 1: Paste the macro from the register where the macro is recorded.

```
"ap  
ALTER USER □wywA IDENTIFIED FOR '□p□a';□:w
```

Step 2: Edit the macro. In this example, the FOR should be BY as shown below.

```
ALTER USER □wywA IDENTIFIED BY '□p□a';□:w
```

Step 3: Copy that macro in to the register.

```
"ayy
```

Bonus Hack 6. Indent Code Block

| Before Indent | After Indent |
|--|--|
| <pre>int main() { printf("Hello World!\n"); printf("Done."); }</pre> | <pre>int main() { printf("Hello World!\n"); printf("Done."); }</pre> |

There are two methods to indent the code as show above.

Method 1:

- Move the cursor to either the { or }
- Press >i{ to indent the code located in between { and }

Method 2:

- Move the cursor to the 1st line after { . i.e printf
- Enable visual mode by pressing v key
- Use arrow key to select the lines in between { and }

- Press > to indent the code located in between { and }

Bonus Hack 7. Power of Combination

You can combine navigation commands with editing commands to achieve powerful results.

For example, dj will delete line by line, where as d`a will delete up to the mark 'a' position.

| Keys | Description |
|-------------------|--|
| d<Navigation Key> | Delete until specified by the navigation key |
| dw | Delete the word |
| d\$ | Delete until end of the line |
| d0 | Delete until start of the line |
| dG | Delete up to end of the file |
| dgg | Delete up to start of the file |
| dk | Delete current line and previous line |
| dj | Delete current line and next line |
| dM | Delete until middle of the screen |
| dH | Delete until middle of the screen |

| | |
|-------------------|--|
| dL | Delete until bottom of the screen |
| y<Navigation Key> | Copy until specified by the navigation key |
| c<Navigation Key> | Change until specified by the navigation key |

Bonus Hack 8. Identify the changes done to a file

You can identify all the changes done to a file after opening it using `:changes` as shown below.

```
:changes
```

Bonus Hack 9. Refresh the Screen

When your screen is visually distorted for some reason, you can redraw it using `CTRL-L`

```
CTRL-L
```

Bonus Hack 10. Insert Non Keyboard Characters

You can insert non keyboard characters into a file using `:digraphs`

```
:digraphs
```

For example, to insert copyright symbol, do the following in insert mode.

```
CTRL-K Co
```

From :help digraphs

Digraphs are used to enter characters that normally cannot be entered by an ordinary keyboard. These are mostly accented characters which have the eighth bit set. The digraphs are easier to remember than the decimal number that can be entered with CTRL-V (see |i_CTRL-V|).

Bonus Hack 11. Vim ex Mode

From normal mode, Press Q to go to the Vim ex mode.

You can go to the ex mode when you want to execute commands in : mode continuously.

Once you've pressed Q you'll stay in ex mode (: mode) until you decide to come out of it.

Enter Ex mode

```
Q
```

Exit Ex Mode

```
:vi
```

Bonus Hack 12. Place the cursor at the end of the match

When you search inside Vim using `/pattern`, by default the cursor will be placed at the beginning of the match.

But when you want the cursor to be placed at the end of the match, you can use `/PATTERN\zs`

Cursor at the beginning of the pattern

```
/pattern
```

Cursor at the end of the pattern

```
/pattern\zs
```

Bonus Hack 13. View ASCII value of a character

For some reason, if you want to know a decimal, Hex and Octal value of a character, move your cursor to the character on press `ga`, which will display the ASCII value as shown below.

ASCII value of character n.

```
ga
```

```
<n> 110, Hex 6e, Octal 156
```

Bonus Hack 14. Edit Binary files in Vim Editor

To edit binary files in Vim editor, use the option -b to Vim command line as shown below.

```
$ vim -b binaryfile
```

Bonus Hack 15. Folding - View Only Required Part of Code

Automatic Folding

To enable folding based on indentation, set the following

```
:set foldmethod=indent
```

After this setting, your code will be folded based on the indentation as shown below.

```
switch ((long) cmd->info) {
case cmd_name:
+-- 22 lines: cfg->name = val;---
case cmd_params:
+-- 2 lines: cfg->params = val;-
#if APR_HAS_THREADS
case cmd_min:
+-- 4 lines: ISINT(val);-----
case cmd_keep:
+-- 4 lines: ISINT(val);-----
case cmd_max:
+-- 4 lines: ISINT(val);-----
case cmd_exp:
+-- 4 lines: ISINT(val);-----
#endif
}
```

Fig: Folded C code.

You can manipulate folds using following keys:

| Fold Keys | Description |
|-----------|----------------------------------|
| za | Toggle the fold under the cursor |
| zR | Unfold all folds |
| zM | Fold everything back again |

Manual Folding

Enable the manual fold as shown below:

```
:set foldmethod=manual
```

You can manipulate manual folds using following keys:

| Fold Keys | Description |
|--------------------|--|
| zf<Navigation-Key> | To fold lines selected by the navigation key |
| zf/pattern | To fold lines selected by the search pattern |
| :range fold | To fold lines specified by a range |

You can also save all your folds as shown below:

```
:mkview
```

Open the view to see all saved custom folds:

```
:loadview
```

Your Feedback and Support

I hope you found the **Vim 101 Hacks** eBook helpful. I sincerely appreciate all the support given by the regular readers of my www.thegeekstuff.com blog, who have encouraged me in more ways than they know.

Please use this contact form <http://www.thegeekstuff.com/contact/> to send me your feedback, question, or clarification on any of the 101 hacks mentioned in this book.