Learning to play Vim

Master the best text editor, from Beginner to Composer

All rights reserved.

While every precaution has been taken in the preparation of this book, the author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Contents

Welcome, Vim Enthusiasts	11
What this book is not?	11
Who Should Read This Book?	11
What Do You Need to Follow Along?	
How To Read This Book?	12
Structure of the book	12
Notation Conventions	
Vim's Help	
How to Get The Most Out of This Book?	
Tips to Learn Efficiently	13
Writing Your Own Cheatsheet	13
Tweaking Endlessly Vim's Configuration	
Playing Vim?	_
What to play: Vim or Neovim?	14
Beyond the Book	14
Going Further	15
The Beginning of the Journey	16
Vim Is An Instrument	16
The Power Is In Your Fingers	
Efficient Typing: The Two Rules	
The First Week	
The Second Week	
Speed and Accuracy	
Keyboard Layout	19
In A Nutshell	
Going Deeper	19
Develor Wine for Develor	21
Rank I - Vim for Rookies	21
The Modes of Vim	
The NORMAL Mode	
The INSERT Mode	
The COMMAND-LINE Mode	
Moving Around with Motions (NORMAL mode)	
Ditching the Arrow Keys	
Horizontal Motions	25
Beginning and End of Line	26
Vertical Motions	
The Language of Vim (NORMAL Mode)	28
The Operators	28
The Text-Objects	29
Undo and Redo (NORMAL mode)	30
Bending Vim to Your Will (Customization)	31
Creating Your Configuration	31
The Configuration Addiction	33
Exercises	34
1/00/10/10/10/10	24

Basics										34
Beyond the Basics										35
Beyond the Basics	• •	•	•	•	• •	•	• •	•	•	36
Playtime Solutions	• •	• •	•	• •	• •	•		• •	•	36
Exercises Solutions		• •	•	• •		•			•	37
Exercises solutions			•	• •		•			•)
Rank II - Vim for Beginners										40
The Modes of Vim, the Return										40
Counting Motions or Text-Objects in NORMAL mode			•	• •		•			•	40
More Veretralice to Cruitely to INCERT Mede			•			•			•	
More Keystrokes to Switch to INSERT Mode			•		• •	•			•	41
The VISUAL Mode			•			•			•	42
The REPLACE Mode			•			•			•	42
Deleting In Vim (NORMAL mode)										44
Cross What You Don't Want										44
Deleted Content To										44
Searching (COMMAND-LINE Mode)										45
Navigating Vim's Help (COMMAND-LINE Mode)										47
Basics						-				47
Follow The Definition	• •	• •	•	• •	• •	•	• •	• •	•	48
Finding What Your Heart Desire		• •	•	• •		•			•	49
Vimagint or Luce (Customization)			•	• •		•			•	50
Vimscript or Lua? (Customization)			•	• •		•	• •		•	
Vim's Options (Customization)			•	• •		•			•	50
Defining Options			•			•			•	50
Setting Options The Interactive Way										52
Useful Options										52
Exercises										53
Basics										53
Beyond the Basics										54
Playtime Solutions										55
Exercises Solutions										56
Beyond The Basics Solutions										56
Deyona The Basics Solations			•	• •		•		• •	•	70
Rank III - Vim for Aspirant										58
Introduction										58
Vim Spatial Organization										58
										58
The Buffers										62
The Windows										
The Tab Pages										
Creating and Deleting Tabs									•	65
Moving From Tab to Tab										65
Moving Tabs										65
Searching (NORMAL mode)										66
Ranges (COMMAND-LINE Mode)										67
Line Specifiers										67
Selection Range for VISUAL Mode										68
Creating Your Own Mappings (Customization)	• •	• •	•	• •	• •	•		• •	•	69
Basics										69
										70
Nested and Recursive Mapping										
Deleting a Mapping										70
Custom Characters										71
The Leader Key										72
Some First Mapping										73
Finding Customized Mapping										73
Exercises										74
Basics										74
Beyond the Basics										75
	•	•		•		-	•	•	•	
Rank IV - Vim for Intermediates										77
Introduction						_				77
	•	- •	-	•	•	•	•	٠.	•	

The Types of Register Reading From the Registers in NORMAL and INSERT modes 86 Using The Registers The Expression Register The Expression Register 87 The Lymping Around (NORMAL mode) 88 Jumping Around (NORMAL mode) 88 The Jumping to Match Elements 88 Jumping to Match Elements 88 Paragraph Jumping 89 Method Jumping 89 The Command Line History (COMMAND-LINE mode) 89 Extending Vim with Plugins (Customization) 80 Exercises 81 Basics 81 Basics 83 Beyond the Basics 84 Beyond the Basics 85 Beyond the Basics 80 Boutions 10 Rank VI - Vim for Proficients Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Changing the Working Directory 10 Seopes of the Working Directory 10 Searching a File 10 Formatting Text (NORMAL mode) 10 Formatting Text (NORMAL mode) 11 Visual Mode and Visual Mode Linewise 11 Useful Keystrokes (COMMAND-LINE mode) 11 Useful Keystrokes (Command History 12 Copying From the Buffers to the COMMAND-LINE 8 Basics 8 Basics 11 Basics 12 Basics 13 Basics 14 Basics 15 Basics 16 Basics 17 Basics 18 Basics 19 Using an External Program 11 Indenting (NORMAL mode) 12 Using an External Program 11 Indenting Keystrokes 12 Controlling the Indentations 12 Displaying Tabs, Spaces, and Others 12 The Equal Operator 13 Advanced Search in Buffer (COMMAND-LINE mode) 12 Advanced Search in Buffer (COMMAND-LINE mode) 12 Advanced Search in Buffer (COMMAND-LINE mode)	Vim's Registers			
Reading From the Registers in NORMAL and INSERT modes Using The Registers The Expression Register 38 Jumping Around (NORMAL mode) 38 The Jump List 38 The Change List Jumping to Match Elements 38 Paragraph Jumping 39 Method Jumping 49 Method Jumping 40 The Command Line History (COMMAND-LINE mode) 40 Extending Vim with Plugins (Customization) 40 Extending Vim with Plugins (Customization) 40 Exercises 40 Exercises 40 Basics 40 Basics 40 Basics 40 Beyond the Basics 40 Beyond the Basics 40 Beyond the Basics 41 Boylutions 40 Boylution	Basics		 	. 77
Reading From the Registers in NORMAL and INSERT modes Using The Registers The Expression Register 38 Jumping Around (NORMAL mode) 38 The Jump List 38 The Change List Jumping to Match Elements 38 Paragraph Jumping 39 Method Jumping 49 Method Jumping 40 The Command Line History (COMMAND-LINE mode) 40 Extending Vim with Plugins (Customization) 40 Extending Vim with Plugins (Customization) 40 Exercises 40 Exercises 40 Basics 40 Basics 40 Basics 40 Beyond the Basics 40 Beyond the Basics 40 Beyond the Basics 41 Boylutions 40 Boylution	The Types of Register		 	. 78
Using The Registers The Expression Register 8 Jumping Around (NORMAL mode) 8 The Jump List 8 The Jump List 8 The Change List 9 Jumping to Match Elements 9 Paragraph Jumping 8 Method Jumping 8 M	Reading From the Registers in NORMAL and INSERT modes		 	. 80
The Expression Register	Using The Registers		 	. 80
Jumping Around (NORMAL mode)	The Expression Register		 	. 81
The Jump List The Change List Sile The Change List Sile Jumping to Match Elements Paragraph Jumping Method Method Method Method Jumping Method Me	Iumping Around (NORMAL mode)		 	. 82
The Change List Jumping to Match Elements Signargarph Jumping Method Jumping Meth	The Jump List			. 82
Jumping to Match Elements S: Paragraph Jumping 34 Method Jumping 38 Method Jumping 38 Extending Vim with Plugins (Customization) 86 A Plugin to Manage Plugin 86 Exercises 88 Basics 88 Beyond the Basics 88 Beyond the Basics 10 Solutions 10 The Current Working Directory 10 Changing the Working Directory 10 Changing the Working Directory 10 Soopes of the Working Directory 10 Soopes of the Working Directory 10 Sormatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting Text (NORMAL mode) 10 Vim's Formatting with External Programs 10 Line or Blockwise' (VISUAL Mode) 11 Visual Mode and Visual Mode Linewise 11 Visual Mode and Visual Mode Linewise 11 Useful Keystrokes (COMMAND-LINE mode) 11 Through The Ex command History 11 Through The Ex command History 11 Multiple Commands on One Line 11 Mapping Special Arguments (Customization) 11 Exercises 11 Mapping Special Arguments (Customization) 11 Exercises 11 Mapping Special Arguments (Customization) 12 Exercises 11 Indenting Special Arguments (Customization) 12 Exercises 11 Indenting (NORMAL mode) 12 Indenting (NORMAL mode) 12 Indenting (NORMAL mode) 12 Indenting With External Program 12 Indenting With External Program 12 Indenting With External Program 12 Advanced Search in Buffer (COMMAND-LINE mode) 12 Indenting With External Program 12 Advanced Search in Buffer (COMMAND-LINE mode) 12 Indenting With External Program 12 Advanced Search in Buffer (COMMAND-LINE mode) 12 Case Sensitivity 12 Search Flags 12	The Change List		 	83
Paragraph Jumping	Jumping to Match Flements	• • •	 	
Method Jumping 88 The Command Line History (COMMAND-LINE mode) 85 Extending Vim with Plugins (Customization) 86 A Plugin to Manage Plugin 88 Exercises 81 Basics 88 Beyond the Basics 10 Solutions 10 Rank VI - Vim for Proficients 10 Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Scopes of the Working Directory 10 Visual Mode and Visual Mode 10 Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 11 Visual Mode and Visual mode Linewise 11 Visual Mode and Visual mode Linewise 11 Useful Keystrokes (COMMAND-LINE mode) 11 Through T	Daragraph Jumping		 	. 03
The Command Line History (COMMAND-LINE mode) 88 Extending Vim with Plugins (Customization) 86 A Plugin to Manage Plugin 87 Exercises 88 Basics 88 Beyond the Basics 88 Beyond the Basics 10 Solutions 10 Rank VI - Vim for Proficients 10 Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Changing the Working Directory 10 Scopes of the Working Directory 10 Searching a File 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Vim's Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 11 Visual Mode and Visual Mode Linewise 11 Visual Mode and Visual mode Linewise 11 Visual Keystrokes (COMMAND-LINE mode) 11 Through The Ex command History<	Mothod lumping		 	. 04
Extending Vim with Plugins (Customization) A Plugin to Manage Plugin Exercises Basics Basics Beyond the Basics Beyond the Basics Beyond the Basics Solutions Rank VI - Vim for Proficients Navigating A Project The Current Working Directory Changing the Working Directory Changing the Working Directory Scopes of the Working Directory Scopes of the Working Directory Searching a File Formatting Text (NORMAL mode) Vim's Formatting Formatting with External Programs Line or Blockwise? (VISUAL Mode) Visual Mode and Visual Mode Linewise Visual Mode and Visual Mode Linewise Usseful Keystrokes (COMMAND-LINE mode) Through The Ex command History Copying From the Buffers to the COMMAND-LINE Malping Special Arguments (Customization) Exercises Basics Controlling the Basics Indenting (NORMAL mode) Indenting Keystrokes Controlling the Indentations Displaying Tabs, Spaces, and Others The Equal Operator Indenting With External Program Indenting With External Program Indenting With External Program Indenting With External Program Indenting Mith External Program Indenting With External Program Indenting With External Program Indenting Mith External Program Indenting With External Program Indenting More Search in Buffer (COMMAND-LINE mode) Case Sensitivity Case Sensitivity Case Sensitivity Lavarces Exercises Basics Basics	The Command Line History (COMMAND, LINE mode)		 	. 04
A Plugin to Manage Plugin Exercises	The Command Line History (Continant Line Hidde)		 	. 00
Exercises 88 Basics 88 Beyond the Basics 88 Beyond the Basics 10 Solutions 10 Rank VI - Vim for Proficients 10 Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Scopes of the Working Directory 10 Scarching a File 10 Formatting work 10 Vimal Mode and Visual Mode 11 Used of Visual Mode and Visual Mode 11 Usual Mode and Visual Mode Linewise 11 Usual Mode and Visual Mode Linewise 11 Usual Mode and Visual Mode Linewise	Extending vim with Plugins (Customization)		 	. 80
Basics 88 Beyond the Basics 81 Beyond the Basics 10 Solutions 10 Rank VI - Vim for Proficients 10 Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Scopes of the Working Directory 10 Searching a File 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 116 Visual Mode and Visual Mode Linewise 11 Visual Mode and Visual mode Linewise 11 Useful Keystrokes (COMMAND-LINE mode) 11 Through The Ex command History 11 Copying From the Buffers to the COMMAND-LINE 11 Multiple Commands on One Line 11 Mapping Special Arguments (Customization) 11 Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 12 <	A Plugin to Manage Plugin		 	. 86
Beyond the Basics 88 Beyond the Basics 10 Solutions 10 Rank VI - Vim for Proficients 10 Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Scopes of the Working Directory 10 Scarching a File 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 11 Visual Mode and Visual Mode Linewise 11 Useful Keystrokes (COMMAND-LINE mode) 11 Through The Ex command History 11 Copying From the Buffers to the COMMAND-LINE 11 Multiple Commands on One Line 11 Mapping Special Arguments (Customization) 11 Exercises 16 Besics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 18 Searching in Multiple Files 11 Vim Internal Search: Vimgrep 118				
Beyond the Basics 100 Solutions 101 Solutions 101 Solutions 101 Solutions 101 Solutions 101 Solutions 102 Solutions 102 Solutions 103 Solutions 103 Solutions 104 Solutions 105 Solutions 106 Solutions 107 Solutions 106 Solutions 107 So				
Solutions 100 Rank VI - Vim for Proficients 100 Navigating A Project 100 The Current Working Directory 100 Changing the Working Directory 100 Scopes of the Working Directory 100 Scopes of the Working Directory 100 Searching a File 100 Formatting a File 100 Formatting Text (NORMAL mode) 100 Vim's Formatting 100 Formatting with External Programs 100 Formatting with External Programs 100 Line or Blockwise? (VISUAL Mode) 110 Uisual Mode and Visual Mode Linewise 111 Visual Mode and Visual Mode Linewise 112 Useful Keystrokes (COMMAND-LINE mode) 112 Through The Ex command History 113 Copying From the Buffers to the COMMAND-LINE 113 Malpping Special Arguments (Customization) 114 Exercises 116 Basics 116 Basics 116 Basics 116 Beyond the Basics 116 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 121 Displaying Tabs, Spaces, and Others 122 Indenting With External Program 122 Advanced Search In Buffer (COMMAND-LINE mode) 122 Case Sensitivity 122 Case Sensitivity 122 Case Sensitivity 124 Search Flags 126				
Rank VI - Vim for Proficients 10 Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Scopes of the Working Directory 10 Searching a File 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 11 Visual Mode and Visual Mode Linewise 11 Visual Mode and Visual mode Linewise 11 Useful Keystrokes (COMMAND-LINE mode) 12 Through The Ex command History 11 Copying From the Buffers to the COMMAND-LINE 11 Multiple Commands on One Line 11 Mapping Special Arguments (Customization) 14 Exercises 16 Basics 16 Basics 11 Basics 11 Basics 12 Using an External Program 11 Using an External Program 12 Using an External Program 12 Undenting Keystrokes 12				
Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Scopes of the Working Directory 10 Searching a File 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 116 Visual Mode and Visual Mode Linewise 117 Usual Mode and Visual mode Linewise 111 Useful Keystrokes (COMMAND-LINE mode) 112 Through The Ex command History 112 Copying From the Buffers to the COMMAND-LINE 113 Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Wing Internal Search: Vimgrep 118 Using an	Solutions		 	. 100
Navigating A Project 10 The Current Working Directory 10 Changing the Working Directory 10 Scopes of the Working Directory 10 Searching a File 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 116 Visual Mode and Visual Mode Linewise 117 Usual Mode and Visual mode Linewise 111 Useful Keystrokes (COMMAND-LINE mode) 112 Through The Ex command History 112 Copying From the Buffers to the COMMAND-LINE 113 Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Wing Internal Search: Vimgrep 118 Using an				
The Current Working Directory Changing the Working Directory Scopes of the Working Directory Scopes of the Working Directory Searching a File 100 Formatting Text (NORMAL mode) Vim's Formatting Formatting with External Programs 100 Formatting with External Programs 100 Line or Blockwise? (VISUAL Mode) Visual Mode and Visual Mode Linewise Visual Mode and Visual mode Linewise Visual Mode and Visual mode Linewise 111 Useful Keystrokes (COMMAND-LINE mode) 112 Through The Ex command History 113 Copying From the Buffers to the COMMAND-LINE Multiple Commands on One Line 114 Mapping Special Arguments (Customization) 115 Exercises 116 Basics 116 Beyond the Basics 116 Beyond the Basics 117 Rank VII - Vim for Seasoned Searching in Multiple Files Vim Internal Search: Vimgrep Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 121 Controlling the Indentations 122 The Equal Operator 123 Indenting With External Program 124 Advanced Search in Buffer (COMMAND-LINE mode) 125 Case Sensitivity 126 Case Sensitivity 126 Search Flags 126 Case Sensitivity 126 Search Flags				
Changing the Working Directory 10. Scopes of the Working Directory 10. Searching a File 10. Formatting Text (NORMAL mode) 10. Vim's Formatting 10. Formatting with External Programs 10. Line or Blockwise? (VISUAL Mode) 110. Visual Mode and Visual Mode Linewise 111. Visual Mode and Visual mode Linewise 112. Useful Keystrokes (COMMAND-LINE mode) 112. Through The Ex command History 113. Copying From the Buffers to the COMMAND-LINE 115. Multiple Commands on One Line 117. Mapping Special Arguments (Customization) 114. Exercises 116. Basics 116. Beyond the Basics 116. Solutions 117. Rank VII - Vim for Seasoned 118. Searching in Multiple Files 118. Vim Internal Search: Vimgrep 118. Using an External Program 119. Indenting (NORMAL mode) 120. Indenting Keystrokes 120. Controlling the Indentations 122. <t< td=""><td>Navigating A Project</td><td></td><td> </td><td>. 101</td></t<>	Navigating A Project		 	. 101
Changing the Working Directory 10. Scopes of the Working Directory 10. Searching a File 10. Formatting Text (NORMAL mode) 10. Vim's Formatting 10. Formatting with External Programs 10. Line or Blockwise? (VISUAL Mode) 110. Visual Mode and Visual Mode Linewise 111. Visual Mode and Visual mode Linewise 112. Useful Keystrokes (COMMAND-LINE mode) 112. Through The Ex command History 113. Copying From the Buffers to the COMMAND-LINE 115. Multiple Commands on One Line 117. Mapping Special Arguments (Customization) 114. Exercises 116. Basics 116. Beyond the Basics 116. Solutions 117. Rank VII - Vim for Seasoned 118. Searching in Multiple Files 118. Vim Internal Search: Vimgrep 118. Using an External Program 119. Indenting (NORMAL mode) 120. Indenting Keystrokes 120. Controlling the Indentations 122. <t< td=""><td>The Current Working Directory</td><td></td><td> </td><td>. 101</td></t<>	The Current Working Directory		 	. 101
Scopes of the Working Directory 10 Searching a File 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 11 Visual Mode and Visual Mode Linewise 116 Visual Mode and Visual mode Linewise 117 Useful Keystrokes (COMMAND-LINE mode) 112 Through The Ex command History 112 Copying From the Buffers to the COMMAND-LINE 113 Multiple Commands on One Line 11 Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting Keystrokes 120 Controlling the Indentations 122 Displaying Tabs, Spaces, and Others 122 Indenting With External Program 12 I	Changing the Working Directory		 	. 102
Searching a File 10 Formatting Text (NORMAL mode) 10 Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 110 Visual Mode and Visual Mode Linewise 111 Visual Mode and Visual mode Linewise 111 Useful Keystrokes (COMMAND-LINE mode) 112 Through The Ex command History 112 Copying From the Buffers to the COMMAND-LINE 11 Multiple Commands on One Line 11 Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Basics 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 122 Indenting With External Program 122 <	Scopes of the Working Directory		 	. 102
Formatting Text (NORMAL mode)	Searching a File		 	. 103
Vim's Formatting 10 Formatting with External Programs 10 Line or Blockwise? (VISUAL Mode) 116 Visual Mode and Visual Mode Linewise 116 Visual Mode and Visual mode Linewise 112 Useful Keystrokes (COMMAND-LINE mode) 112 Through The Ex command History 112 Copying From the Buffers to the COMMAND-LINE 115 Multiple Commands on One Line 115 Mapping Special Arguments (Customization) 116 Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 122 Indenting Keystrokes 120 Controlling the Indentations 122 Displaying Tabs, Spaces, and Others 122 Indenting With External Program 12 Advanced Search In Buffer (COMMAND-LINE mode) 122 Case Sensitivity 122 <tr< td=""><td>Formatting Text (NORMAL mode)</td><td></td><td> </td><td>. 106</td></tr<>	Formatting Text (NORMAL mode)		 	. 106
Formatting with External Programs 10' Line or Blockwise? (VISUAL Mode) 116 Visual Mode and Visual Mode Linewise 117 Useful Keystrokes (COMMAND-LINE mode) 117 Through The Ex command History 117 Copying From the Buffers to the COMMAND-LINE 117 Multiple Commands on One Line 117 Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Beyond the Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 122 Displaying Tabs, Spaces, and Others 122 Indenting With External Program 122 Advanced Search In Buffer (COMMAND-LINE mode) 122 Case Sensitivity 122 Search Flags 122	Vim's Formatting		 	. 106
Line or Blockwise? (VISUAL Mode) Visual Mode and Visual Mode Linewise Visual Mode and Visual mode Linewise Useful Keystrokes (COMMAND-LINE mode) Through The Ex command History Copying From the Buffers to the COMMAND-LINE Multiple Commands on One Line Mapping Special Arguments (Customization) Exercises Basics Beyond the Basics Solutions Solutions Vim Internal Search: Vimgrep Using an External Program Indenting (NORMAL mode) Indenting Keystrokes Controlling the Indentations Displaying Tabs, Spaces, and Others The Equal Operator Indenting With External Program Advanced Search In Buffer (COMMAND-LINE mode) Case Sensitivity Case Sensitivity Case Sensitivity Search Flags 12 Controllage Indentations 12 Case Sensitivity 12 Case Sensitivity 12 Search Flags	Formatting with External Programs		 	. 109
Visual Mode and Visual mode Linewise 11 Visual Mode and Visual mode Linewise 11 Useful Keystrokes (COMMAND-LINE mode) 11 Through The Ex command History 11 Copying From the Buffers to the COMMAND-LINE 11 Multiple Commands on One Line 11 Mapping Special Arguments (Customization) 11 Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 12 Displaying Tabs, Spaces, and Others 12 The Equal Operator 12 Indenting With External Program 12 Advanced Search In Buffer (COMMAND-LINE mode) 12 Case Sensitivity 12 Search Flags 12	Line or Blockwise? (VISUAL Mode)			. 110
Visual Mode and Visual mode Linewise 11 Useful Keystrokes (COMMAND-LINE mode) 112 Through The Ex command History 112 Copying From the Buffers to the COMMAND-LINE 112 Multiple Commands on One Line 113 Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 121 Displaying Tabs, Spaces, and Others 122 The Equal Operator 122 Indenting With External Program 121 Advanced Search In Buffer (COMMAND-LINE mode) 122 Case Sensitivity 126 Search Flags 124	Visual Mode and Visual Mode Linewise		 	110
Useful Keystrokes (COMMAND-LINE mode) Through The Ex command History Copying From the Buffers to the COMMAND-LINE Multiple Commands on One Line Mapping Special Arguments (Customization) Exercises Basics Basics Beyond the Basics Solutions 117 Rank VII - Vim for Seasoned Searching in Multiple Files Using an External Program Indenting (NORMAL mode) Indenting (NORMAL mode) Indenting Keystrokes Controlling the Indentations Displaying Tabs, Spaces, and Others The Equal Operator Indenting With External Program 127 Indenting With External Program 128 Advanced Search In Buffer (COMMAND-LINE mode) Case Sensitivity Search Flags	Visual Mode and Visual mode Linewise	• • •	 	111
Through The Ex command History Copying From the Buffers to the COMMAND-LINE Multiple Commands on One Line Mapping Special Arguments (Customization) Exercises Basics Beyond the Basics Beyond the Basics Solutions Rank VII - Vim for Seasoned Searching in Multiple Files Vim Internal Search: Vimgrep Using an External Program Indenting (NORMAL mode) Indenting (NORMAL mode) Indenting Keystrokes Controlling the Indentations Displaying Tabs, Spaces, and Others The Equal Operator Indenting With External Program 12 Advanced Search In Buffer (COMMAND-LINE mode) Case Sensitivity Search Flags	Useful Keystrokes (COMMAND-LINE mode)		 	112
Multiple Commands on One Line 113 Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 121 Displaying Tabs, Spaces, and Others 122 Indenting With External Program 122 Advanced Search In Buffer (COMMAND-LINE mode) 123 Case Sensitivity 126 Search Flags 124	Through The Ex command Distory		 	112
Multiple Commands on One Line 113 Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 121 Displaying Tabs, Spaces, and Others 122 Indenting With External Program 122 Advanced Search In Buffer (COMMAND-LINE mode) 123 Case Sensitivity 126 Search Flags 124	Conving From the Duffers to the CONTAGAND LINE		 	112
Mapping Special Arguments (Customization) 114 Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 122 Displaying Tabs, Spaces, and Others 122 The Equal Operator 122 Indenting With External Program 122 Advanced Search In Buffer (COMMAND-LINE mode) 123 Case Sensitivity 124 Search Flags 124	Copying From the Butters to the COMMAND-LINE		 	. 113
Exercises 116 Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 121 Displaying Tabs, Spaces, and Others 122 The Equal Operator 122 Indenting With External Program 122 Advanced Search In Buffer (COMMAND-LINE mode) 123 Case Sensitivity 124 Search Flags 124	Multiple Commands on One Line		 	. 113
Basics 116 Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 121 Displaying Tabs, Spaces, and Others 122 The Equal Operator 122 Indenting With External Program 122 Advanced Search In Buffer (COMMAND-LINE mode) 122 Case Sensitivity 122 Search Flags 124				
Beyond the Basics 116 Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 122 Displaying Tabs, Spaces, and Others 122 The Equal Operator 122 Indenting With External Program 122 Advanced Search In Buffer (COMMAND-LINE mode) 122 Case Sensitivity 122 Search Flags 124				
Solutions 117 Rank VII - Vim for Seasoned 118 Searching in Multiple Files 118 Vim Internal Search: Vimgrep 118 Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 120 Displaying Tabs, Spaces, and Others 120 The Equal Operator 120 Indenting With External Program 120 Advanced Search In Buffer (COMMAND-LINE mode) 120 Case Sensitivity 120 Search Flags 120				
Rank VII - Vim for Seasoned Searching in Multiple Files				
Searching in Multiple Files118Vim Internal Search: Vimgrep118Using an External Program119Indenting (NORMAL mode)120Indenting Keystrokes120Controlling the Indentations121Displaying Tabs, Spaces, and Others122The Equal Operator123Indenting With External Program123Advanced Search In Buffer (COMMAND-LINE mode)123Case Sensitivity124Search Flags124	Solutions		 	. 117
Searching in Multiple Files118Vim Internal Search: Vimgrep118Using an External Program119Indenting (NORMAL mode)120Indenting Keystrokes120Controlling the Indentations121Displaying Tabs, Spaces, and Others122The Equal Operator123Indenting With External Program123Advanced Search In Buffer (COMMAND-LINE mode)123Case Sensitivity124Search Flags124	n 1 mm m ()			440
Vim Internal Search: Vimgrep118Using an External Program119Indenting (NORMAL mode)120Indenting Keystrokes120Controlling the Indentations121Displaying Tabs, Spaces, and Others122The Equal Operator123Indenting With External Program123Advanced Search In Buffer (COMMAND-LINE mode)124Case Sensitivity124Search Flags124				
Using an External Program 119 Indenting (NORMAL mode) 120 Indenting Keystrokes 120 Controlling the Indentations 121 Displaying Tabs, Spaces, and Others 122 The Equal Operator 122 Indenting With External Program 123 Advanced Search In Buffer (COMMAND-LINE mode) 123 Case Sensitivity 124 Search Flags 126	Searching in Multiple Files		 	. 118
Indenting (NORMAL mode)120Indenting Keystrokes120Controlling the Indentations121Displaying Tabs, Spaces, and Others122The Equal Operator122Indenting With External Program123Advanced Search In Buffer (COMMAND-LINE mode)124Case Sensitivity124Search Flags124	Vim Internal Search: Vimgrep		 	. 118
Indenting (NORMAL mode)120Indenting Keystrokes120Controlling the Indentations121Displaying Tabs, Spaces, and Others122The Equal Operator122Indenting With External Program123Advanced Search In Buffer (COMMAND-LINE mode)124Case Sensitivity124Search Flags124	Using an External Program		 	. 119
Indenting Keystrokes120Controlling the Indentations121Displaying Tabs, Spaces, and Others122The Equal Operator122Indenting With External Program123Advanced Search In Buffer (COMMAND-LINE mode)123Case Sensitivity124Search Flags124	Indenting (NORMAL mode)		 	. 120
Controlling the Indentations	Indenting Keystrokes		 	. 120
Displaying Tabs, Spaces, and Others	Controlling the Indentations		 	. 121
The Equal Operator				
Indenting With External Program				
Advanced Search In Buffer (COMMAND-LINE mode)	Indenting With External Program		 	. 123
Case Sensitivity	Advanced Search In Buffer (COMMAND-LINE mode)		 	. 123
Search Flags				
1			 	125

Find and Replace One Occurrence at a Time								. 12	25
Search Highlight								. 12	25
Searching Without Moving the Cursor								. 12	26
Using Search as Range								. 12	26
Writing Custom Functions (Customization)								. 13	28
Checking Existing Functions	•	•		•	•	•	•	11	28
Creating Or Copying Functions	•	•	• •	•	•	•	•	12	28 28
Evargicae	•	•		•	•	•	•	11	20 29
Exercises									29 29
Basics									
Beyond the Basics									29
Solutions	•	•		•	•	•	•	. 13	30
Rank VIII - Vim for Adepts								11	31
Introduction	•	•		•	•	•	•	. 1.	31
The Quickfix Lists and Location Lists									31
Quickfix Lists									31
Location Lists									34
Filtering the Quickfix List									35
Acceding the Quickfix Lists								. 13	35
Changing Case (NORMAL mode)								. 13	36
Using Your Shell Commands in Vim (COMMAND-LINE mode)								. 13	38
Executing External Command								. 13	38
Inserting the Output of a Shell Command in the Current Buffer.								. 13	39
Using Part of the Buffer as Input								. 14	40
Replacing a Range of Lines with a Shell Command Output	•	•	• •	•	•	•	•	1/	40
Replacing Lines with a Shell Command Output in Normal Mode									41
The Command Execute (Customization)	•	•		•	•	•	•	1,	41 42
									42 44
Exercises									
Basics									44
Beyond the Basics									44
Solutions	٠	•		•	•	•	•	. 14	44
Interlude II Heaful Vim Dluging the Deturn								1/	4 5
Interlude II - Useful Vim Plugins, the Return Bringing Git in Vim									+5 45
									_
Linters for Vim	•	•		•	•	•	•	. 14	45
Manipulating Buffers, Windows, and Tabs	•	•		•	•	•	•	. 14	46
Closing Buffers Without Closing Windows	•	•		•	•	•	•	. 14	46
Managing Windows Easily	•	•		•	•	•	•	. 14	46
Different Navigation		•		•			•	. 14	46
Outliners									46
Debugging Any Language								. 14	47
Rank IX - Vim for Believers									48
Introduction	•	•		•				. 14	48
Vim Regular Expressions								. 14	48
A Brief Return to Basics								. 14	49
The Concept of Atom								. 1	50
Vim's "Magical" Patterns									51
Word Boundaries									52
Character classes and Vim Options									52
Greedy and Non-Greedy Identifiers									52 52
Vim Lookaround Assertions									54
Regexes Matching On Multiple Lines	•	•		•	•	•	•	11	55
Only Matching the Viewal Calcution	•	•		•	•	•	•	. 13	
Only Matching the Visual Selection									56 57
Manipulating Numbers (NORMAL mode)									57
The Global Command (COMMAND-LINE mode)									59
Basics									59
Useful Examples									60
Global Command and Substitution							_	. 16	52

Autocommands (Customization)	
Basics	. 163
Multiple Events and Patterns	. 164
Autocommand Groups	
Ignoring Events	. 167 . 168
Exercises	
Basics	
Beyond the Basics	
Solutions	. 109
Rank X - Vim for Veterans	170
Marks	. 170
Basics	. 170
Read Only Marks	
Special Marks	
Regexes and Marks	
Absolute and Relative Line Numbers (NORMAL mode)	
Switching Between Absolute and Relative Line Number	
From INSERT to NORMAL Mode for One Command (INSERT mode)	
The Normal Ex Command (COMMAND-LINE mode)	. 176
The Normal and Global Ex Command	
The Normal Ex Command and Special Keys	
TO SORT	. 178
Basics	
Attributes for User Commands	. 180
Exercises	. 183
Basics	
Beyond the Basics	
Solutions	
Interlude III - Vim Runtime	184
Vim's Startup	
Startup's Order	
Profiling Vim's Startup	. 185
Special Environment Variables	
The Runtime Path	
Subdirectories of the Runtime Paths	
Autoloading Functions	
The Directory after	
The Runtime Command	
Disabling Runtime Files	. 190
The Startup Has Been Revealed	. 190
•	
Rank XI - Vim for Experts	191
Undo In Depth	
Persisting Undo	
The Undo Tree	
Creating Undo Nodes	. 194
Abbreviations (INSERT mode)	
Basics Ex Commands	. 196 . 197
Replacing and Moving the Cursor	
Abbreviations and Mapping	
The Message History (COMMAND-LINE mode)	
The Operator Pending Map (Customization)	
Exercises	
Rasirs	201

Beyond the Basics	
Rank XII - Vim for Champions	202
Folding	202
Fold Options	202
Choosing Your Fold Method	202
NORMAL Mode Keystrokes	203
Opening and Closing Folds with Ex Commands	204
Folding Tips	
Digraphs (INSERT mode)	205
Special Strings for Vim Commands (COMMAND-LINE mode)	206
The viminfo and shada Files (Customization)	208
Exercises	211
Basics	
Beyond the Basics	
Solutions	
Interlude IV - Vimscript and Lua	212
When Using Lua?	
Lua in Vimscript Files	
Lua scripts	
Lua API	
Interoperability: Vimscript From Lua	
Testing and debugging	
Global Functions	
Vim Loop	
Often Used Lua Functions	215
Rank XIII - Vim for Masters	216
Syntax Highlighting	
Enabling and Disabling Syntax Highlighting	216
Color Schemes	
Highlight Groups	
Linking Highlight Groups	
Creating Syntax Groups	
The Verbose Command	221
Troubleshooting for Syntax Highlighting and Big Files	222
Displaying Useful Information About File and Cursor Position (NORMAL mode)	223
Diff Demystified (NORMAL mode)	
Beginning a Diff	
Configuring Diffs with An Option	
Debugging Vimscript & Lua (Customization)	227
Verbosity	
The Debug Mode	
Exercises	
Basics	
Beyond the Basics	
Solutions	231
Rank XIV - Vim for Grand Masters	232
Compiling and Linting	232
Running a Binary Against Your Code	232
Running the Executable	
Parsing Error Messages	
Creating a Compiler for Lua	
Completion and Completion Submode (INSERT mode)	237
The Completion Submode	
Scrolling in INSERT MODE	

The Complete Option		 	 	238
The Omni-completion		 	 	239
The Arglist (COMMAND-LINE mode)		 	 	240
Practical Use: Find and Replace in Multi	iple Files	 	 	242
Managing Plugins in Vanilla Vim (Customiza	ation)	 	 	243
The Vim Native Package Manager		 	 	243
Installing New Plugins		 	 	244
Loading Plugins on Demand		 	 	245
Exercises		 	 	245
Basics				
Beyond the Basics				
Solutions		 	 	247
The File Manager netrw				248
Opening netru				
Opening netrw				
Browsing				
Basics				
Display				
Filtering Display				
Listing the Browsing History		 	 	252
Marking Files And Directories		 	 	253
Managing Files and Directories		 	 	254
Creating and Deleting		 	 	254
Renaming Files or Directory		 	 	254
Copying Files			 	255
Moving Files				
File Permissions				
Opening Files with External Applications .		 	 	
Bookmarking		 	 	256
Remote Operations and Protocols		 	 	256
Heing con via CCII		 	 	250 257
Using scp via SSH				
Using HTTP (read only)		 	 	258
Listing Directories		 	 	258
Obtaining a file				
FTP				
The NETRC file				
Customizing Mapping		 	 	259
Overwriting Variables and Functions				
Command Line Editing		 	 	
TODO		 	 	260
Rank XV - Vim for Champions				261
Vim's Spelling				
Basics				
Adding Words to Spell Files				
Fixing Spelling with Word Suggestions		 	 	264
Navigating Through Your Wrong Words	s	 	 	265
Advanced Macro (NORMAL & VISUAL mode))	 	 	265
Visual Mode Macro				
Creating Mapping from Macros				
Recursive Macro				
Redirections (COMMAND-LINE mode)				
Improving Vim Performances				
General Profiling		 	 	270 270
Startup Profiling				
Profiling Syntax Files				
Exercises		 • • • •	 	212 272

Beyond the Basics	
Exercice xxxx - Advanced Macro	272 272
Solutions	212
Rank XVI - Vim for Heroes	273
Jumping to Definition	273
Vim and ctags	273 275
Completion with tags	275
Saving Settings and Vim Sessions (COMMAND-LINE mode)	281
Saving Vim's Options and Mapping in a File	281
Creating and Loading a Session	281
Fine Tuning Vim's Sessions	282
Setting Your Status Line (Customization)	283
Status Line Options	283 285
Status Line Separator	286
Setting your Tab Line	286
Exercises	290
Basics	290
Beyond the Basics	290
Solutions	290
Rank XVII - Vim for Composers	291
Restoring the Cursor Position	291
Opening a Window Fullscreen	292
Output Redirection into a Scratch Buffer	295 300
Git Information From the Current Line	300
Creating a Popup	301
CONCLUSION	302
Annexes	303
	205
	305
Default word characters	306
Options	307
Text Objects	308
Mapping Special Keys	310
Operators	311
Ranges	312
Special Keycodes	313
Mapping Special Arguments	314
The "g" Commands	315
The "do" Ex commands	316
The "do" Ex commands Vimscript and Lua Functions Seen in the Book	316 317

Highlights Highlight Attributes	320
Autcommands	322
Useful Vimscript Functions	323
Internal word list for spellchecker	324
Commands Managing Windows	325
Statusline	326
External programs	327
Search Ex Commands on Lists (COMMAND-LINE mode	328 328

Rank I - Vim for Rookies

This is were our adventure begins: on the shore of the very basic functionalities of Vim. If you try to actively understand the concepts explained in this chapter, you'll be able to use Vim for your daily editing. It won't replace your IDE right away; but it's enough to edit some configuration or other plain text files.

More specifically, we'll see, in this chapter:

- The basic Vim modes. This is mainly what makes Vim so different from any other text editor
- How to move your cursor around, only using our precious keyboard.
- How to use Vim's "language".
- How to undo and redo your changes.
- How to configure Vim.

I recommend you to open the file <code>rank_01/functions.lua</code> from the book companion to try and experiment by yourself what's explained here. That's what I mean by "actively understand" this book: fire Vim (or Neovim), try the different commands, and experiment by yourself.

I see that you're bursting with anticipation: let's not wait any longer!

The Modes of Vim

Vim is a *modal editor*. It means that the keys of your keyboard will perform different actions depending on the mode you're in. This is the first big difference with the text editors you're likely used to, and arguably one of the biggest reason why Vim is so beloved and powerful.

The modes you'll use the most are the NORMAL mode and the INSERT mode. It's important to understand both of them to be able to use Vim efficiently.

The NORMAL Mode

When you open most text editors out there, it will allow you to directly type some text with your beloved keyboard. What you type will appear immediately on your screen. Obvious, isn't it? Well, it's exactly why many beginners are confused when firing Vim for the first time: it doesn't quite work like that.

To understand what I mean, try to open Vim. You'll see a welcome message. Now, if you try to type "x" in the editor, you might be surprised: no "x" appears on the damn screen!

It's because Vim always start by default in NORMAL mode. This mode is not meant to *insert* new text, but to *edit* already existing text.

In NORMAL mode, we can use many different commands to move the cursor and target the exact text we want to edit. In many text editors, you would need to use the mouse to select the portion of text you want to change, or to move your cursor to insert new text. Vim allows you to do that, but only using your keyboard. It's better for your hands, and it's also way more efficient.

You can think of NORMAL mode commands as keystrokes, or keyboard shortcuts. Usually, in other text editors, if you don't want to use your mouse all the time, you would need to hit a shortcut to perform an action, like selecting all your text with CTRL-a for example. In Vim, since you don't write anything directly when typing on your keyboard in NORMAL mode, you have way more keystrokes to edit your text. You have access to all the keys on your keyboard, in fact.

But if there are so many NORMAL mode commands, how is it possible to remember them all? It's where it gets very interesting. These commands make sense in Vim, most of the time, compared to the usual and meaningless shortcuts you'll find in other editors.

Vim's NORMAL mode commands use mnemonics for you to remember them easily. Even better: they are *composable*: you can combine some of them in a logical, easy-to-remember way.

Let's take for example the shortcut CTRL+shift+n from a random IDE. By only looking at it, you've no idea what it does. In contrast, you'll see soon that it's possible to guess NORMAL mode commands in Vim.

Vim's NORMAL mode is a keyboard-centered way to control your editor, *telling* Vim what you want to do, and it will obey your mighty will. Vim is your new slave.

To really grasp this concept, we need some text to edit, so let's insert some text first. To do so, let's switch to the second most important mode, the INSERT mode.

The INSERT Mode

Let's now execute our first NORMAL mode command, which will switch Vim from NORMAL mode to INSERT mode. Simply hit i on your keyboard.

Depending on the editor you use (Vim or Neovim), and how your terminal is configured, the shape of your cursor might change. More importantly, you'll see -- INSERT -- in the bottom left corner of Vim.

Welcome to INSERT mode!

You're now able to type the text you always wanted to bring to the world. Go ahead, don't be afraid: type anything you want, like you would do in any other editors. At the end, Vim is not that different.

Now, let's try to hit the ESC key. The indicator -- INSERT -- disappears.

Welcome back to NORMAL mode!

That's exactly what do a Vim user, most of the time: switching between NORMAL mode to edit existing text, and INSERT mode to insert directly new text.

The NORMAL mode command a can also let you switch to INSERT mode, but it will do it \underline{a} fter the character you're on. Try it to see the difference! Remember: to switch back to NORMAL mode, simply hit ESC.

That's what I'm talking about when I say that Vim uses mnemonic for the NORMAL mode commands: i for insert, a for insert after.

To summarize what we've just seen:

Keystroke	Description
i	Switch from NORMAL mode to INSERT mode (insert before current character).
а	Switch from NORMAL mode to INSERT mode (insert <u>a</u> fter current character).
<esc></esc>	Switch from INSERT mode to NORMAL mode.

There are more NORMAL mode commands allowing us to switch to INSERT mode. We'll see them in the next chapter; for now, the ones above should be enough for your basic editing needs.

The COMMAND-LINE Mode

The NORMAL and the INSERT modes are the ones you'll use most often. Following my subjective order of importance, we'll find a third mode, the COMMAND-LINE mode.

Now, you might have noticed that we've spoken about NORMAL mode *commands*. What's this COMMAND-LINE mode? A new way to enter commands? Well, kind of.

In NORMAL mode, you run NORMAL mode commands; in COMMAND-LINE mode, you run Ex commands. The word "command" is used slightly differently here, and it's quite confusing; that's why I'll speak about NORMAL mode keystrokes, instead of NORMAL mode commands. That said, be aware that many other resources about Vim (including Vim's help) often speak about NORMAL mode commands.

Let's go back to our new mode, the COMMAND-LINE mode. First, to switch to COMMAND-LINE mode, you need to use the NORMAL mode keystrokes : . To come back to normal mode, you need again to hit ESC .

When you switch to COMMAND-LINE mode, your cursor moves automatically at the very bottom of Vim. It will be after a colon:, indicating that you can write and run an Ex command.

You can think of Ex commands as the menu you would normally use in a text editor with a graphical user interface (GUI). In COMMAND-LINE mode, you can run Ex commands to save the file you're editing, or to search and replace some text for example. Again, contrary to other text editors out there, you can do all of that only using your keyboard. It's your instrument to play some Vim!

Here are some useful basic Ex commands:

Ex command	Short Name	Description
:write	: W	Write (save) the current file open.
:write!	:w!	Write (save) the current file open (even if it's read-only).
<pre>:edit {filepath}</pre>	<pre>:e {filepath}</pre>	Edit the file with filepath {filepath}.
:quit	: q	Quit the current window.
:quit!	:q!	Quit the current window without saving.
:wq	: wq	Write (save) the current file and quit Vim.

A last important Ex command, maybe the most important of all: $\verb|:help {subject}|$, to open the help about whatever $\verb||subject||$ you want. For example, if you want to know more about the NORMAL mode command $\verb||i|$, you can run the Ex command $\verb|:help|$ i.

Do you already have a best friend? Ditch her. Vim's help is your new best friend from now on. It's you're go to if you need any kind of information about anything Vim, really. If you don't remember how to quit Vim for example (it happens often), you can run the Ex command shelp quit.

I'll often directly reference Vim's help in this book, at the end of many sections. If you want to, it will allow you to dig deeper into the functionalities we'll cover.

For example:



Help Yourself

:help vim-modes
:help write-quit

Don't worry if you don't really understand what's written in these help files at first, or if there is too much information. It will make more sense when you'll get more comfortable with Vim.

A last important tip about the COMMAND-LINE mode: you can use the TAB key to complete Ex commands. It's useful when you don't remember exactly the commands, or to discover new ones! Also, Ex commands can take some arguments, and they can be completed too!

Similarly to a shell (like Bash or Zsh), you can also use the arrow keys <up> and <down> to go through your Ex command history.

Moving Around with Motions (NORMAL mode)

Let's now see how we can move our cursor horizontally or vertically, thanks to NORMAL mode keystrokes called *motions*.

Don't worry if you don't remember every single NORMAL mode keystroke we'll see here. You can always come back to this chapter and experiment with the one you forgot.

Ditching the Arrow Keys

We're now at the most difficult part in our journey to learn Vim. At least it was the most difficult part for me: ditching the arrow keys to move the cursor.

As I said in the previous chapter, our fingers should stay on the row keys of the keyboard. First, for our typing speed and accuracy to improve, and second because the Vim's keystrokes you can use in NORMAL mode are more easily accessible from the row keys. Your hands shouldn't move too much; only your fingers should.

If you look at your arrow keys, you'll see that they're far away from the row keys, forcing you to move your hands each time you want to use them. That's why, instead of using the arrow keys, many Vim users use the h, j, k and 1 keys instead, to move respectively left, down, up and right.

I would highly encourage you to use these keys, too. They'll improve your Vim experience significantly.

Why hjkl, and not some other keys close to the home row? For historic reasons. Vim is the ancestor of Vi, which was used on physical terminals. When you look at the keyboard of some of them, you'll see that the arrow keys are also the hjkl keys.

Like many other habits which seem ingrained in our brain, it will be difficult not to use the arrow keys at first. Your hand will come back to them over and over, even if you try not to. You need to accept this fact and be patient; you'll get there, and faster than you think.

For an easier transition, we can try to answer an important question: how to remember what $\, h \,$, $\, j \,$, $\, k \,$ and $\, 1 \,$ do in NORMAL mode? Here are some useful mnemonics:

- The h key is on the left of the sequence hjkl, and l is on the right. As a result, hitting h will move your cursor to the left, and l to the right.
- j moves your cursor down. Here are 3 mnemonics for this key:
 - With a bit of imagination, you can see j as an arrow going down.
 - The j has a little bump at the bottom of the key, meaning that it goes down.
 - Let's speak typography: j has a descender, meaning that part of the letter descends from its baseline. As a result, j goes down.
- k is the only letter left, so it has to go up. To come back to the secret art of typography, k is a letter with an ascender, meaning that part of the letter ascend from its baseline. As a result, k goes up.

Practice will get you there. I've got you covered for this one, with a revolutionary AAA game everybody will speak about in twenty years. To play it, you *must* use <code>hjkl</code> . If you prefer puzzle games, try this wonderful sokoban. You can use <code>hjkl</code> or the arrow keys this time, but try to only use <code>hjkl</code>.

Horizontal Motions

The keys h and 1 are not the only ones you can use to move horizontally, on the current line. Actually, long time Vim users rarely use them. Instead, we can use other motions in NORMAL mode to move faster, like these:

Keystroke	Description
W	Move forward to the beginning of the next word.
W	Move forward to the beginning of the next $\underline{\mathbf{W}}$ ORD.
е	Move forward to the <u>e</u> nd of the next word.
E	Move forward to the <u>e</u> nd of the next WORD.
b	Move <u>b</u> ackward to the beginning of the word.
В	Move <u>b</u> ackward to the beginning of the WORD.
ge	Move backwatd to the end of the previous word.

A question arise: what's the difference between a "word" and a "WORD"? They represent two different motions. A "WORD" follows the usual concept of a word; a string of characters delimited by spaces.

You can think of a "word" as a keyword, containing only a specific set of characters. Mainly, it doesn't include some special characters.

For example, if you have the file "rank_01/functions.lua" from the book companion open, you can enter

```
-- A word and a WORD: see the difference?
local function restorePosition() {
```

Now, try to use the motions we've seen above to see the difference. Using "WORD" motions will skip the parenthesis, but the "word" motion won't.

Here are some more horizontal motions I find particularly useful:

Keystroke	Description
f{character}	To <u>find a</u> {character} after your cursor.
F{character}	To <u>find</u> a {character} before your cursor.
t{character}	Move <u>till</u> a {character} after your cursor.
T{character}	Move <u>till</u> a {character} before your cursor

After using one of these four keystrokes above, you can continue to move from character to character with:

Keystroke	Description
;	Move forward
,	Move backward



Help Yourself

```
:help cursor-motions
:help left-right-motions
```

Beginning and End of Line

Moving word by word can be slow and boring if you want to go to the beginning or the end of the line. Here are some more NORMAL mode keystrokes:

Keystroke	Description
0	Move to the first character of the current line.
\$	Move to the last character of the current line.
^	Move to the first non-whitespace character on the current line.

A whitespace can be a space or a tab.



It's Playtime!

Exercise A - Horizontal Motions

Using the hjkl keys in NORMAL mode, move your cursor at the beginning of restorePosition:

How would you:

- 1. Move to the position first f of function?
- 2. Move back to the first r of restorePosition()?
- 3. Move to the end of the line?

Vertical Motions

Here are more NORMAL mode keystrokes, but to move your cursor vertically this time:

Motion	Description
{line_number}G	Move to the beginning of the line numbered {line_number}. For example, 10G will move the cursor on line 10.
G	Move to the last line of your file.
1G or gg	Move to the first line of your file.
CTRL-u	move <u>u</u> pward for half a screen.
CTRL-d	move <u>d</u> ownward for half a screen.
CTRL-b	move <u>b</u> ackward (upward) an entire screen.
CTRL-f	move <u>f</u> orward (downward) an entire screen.

You can also use COMMAND-LINE mode to move to a specific line number, with $:\{1ine-number\}$. For example, :10 will move your cursor to line 10.

Finally, here are three last keystrokes allowing us to move to the top, middle, or last line of the actual display:

Motion	Description
Н	Move to the top ($\underline{\mathbf{H}}$ ome, or $\underline{\mathbf{H}}$ ighest) of the window.
L	Move to the <u>l</u> ast line of the window.
М	Move to the <u>m</u> iddle line of the window.



It's Playtime!

Exercise B - Vertical Motions

In NORMAL mode, how would you:

- 1. Move to the 5th line of the file?
- 2. Move back to the very beginning of the file?
- 3. Move to the very end of the file?
- 4. Move multiple lines upward with one keystroke?



Help Yourself

:help up-down-motions

The Language of Vim (NORMAL Mode)

If you searched online some tutorials or other resources about Vim, I'm pretty sure you've seen this one: Vim has a language! You can speak with your editor!

In Vim, NORMAL mode keystrokes can be seen as "sentences", describing an action you want to perform. That's what I meant when I was saying that the keystrokes are *composable*. It's nothing less than brilliant.

These "sentences" are quite easy to understand. It allows us to link what we know already (the sentence) by what we need to learn (the NORMAL mode keystrokes).

Even better: knowing that Vim has a "keystroke language" will push you to combine them instinctively to do what you need to do, and, in many cases, it will work!

The Operators

We've learned how to walk in Vim with motions. It's time to perform some actions. To operate on our content.

The *operators* are the verbs of the Vim language. Here are three common operators:

Operator	Description
d	<u>d</u> elete
С	<u>c</u> hange
У	yank (copy)

These operators won't do anything if you hit them in NORMAL mode. You need to combine them with motions. For example:

Keystroke	Description
d\$	To $\underline{\mathbf{d}}$ elete from your cursor to the end of line. You can also use the alias D .
dgg	To $\underline{\mathbf{d}}$ elete everything from the cursor to the beginning of the file.
ggdG	Move your cursor to the beginning of the file, and $\underline{\mathbf{d}}$ elete everything till the end.

Let's explain a bit further what these three operator are doing:

- The delete operator is self-explanatory.
- The <u>c</u>hange operator will delete and immediately switch to INSERT mode, effectively allowing us to... change our text.
- The <u>y</u>ank operator allows us to copy some of our text. Then, using the keystroke p, it can be paste (the official term is put) somewhere else in your text.

By default, the keystroke $\,_P$ will paste the content after the character under the cursor. To put it before, use the uppercase variant of the keystroke $\,_P$. The logic stays the same when you \underline{y} ank whole lines and put them back.

I encourage you to try out all these operators. You can combine them with the motions we've seen in this chapter. Again, more practice you'll have, better you'll get! The exercises at the end of the chapter will help you to get there, too.



Help Yourself

:help operator

:help objet-motions

The Text-Objects

Instead of motions, we can also use another construct with our operators: the famous Vim text-objects. If the operators are the verbs of the Vim language, the text-objects are the nouns.

Simply put, a text-object is a set of character with a specific start and end. In Vim, "a word" is a text object, as well as "a sentence", or "a paragraph".

For example, you can use operators and text-objects in NORMAL mode as follows:

Keystroke	Description
diw	To <u>d</u> elete <u>i</u> nside the <u>w</u> ord.
	It deletes the current word under the cursor.
daw	To <u>d</u> elete <u>a</u> round the <u>w</u> ord.
	It deletes the current word under the cursor and its leading and trailing whitespaces.
ciw	To change inside the word.
	It deletes the current word under the cursor and switch to INSERT mode.
	In short, you change the word!
dip	To <u>d</u> elete <u>i</u> nside the <u>p</u> aragraph.

Note that each of these examples are composed of an operator and a text-object. For example, for the first example, d is the operator, iw is a text-object.

In Vim's help, daw is described as $\underline{\mathbf{d}}$ elete $\underline{\mathbf{a}}$ word. I find this "translation" quite confusing, that's why I use $\underline{\mathbf{a}}$ round instead of $\underline{\mathbf{a}}$. Indeed, text-objects beginning with a "a" (like $\underline{\mathbf{a}}$ w) often delete something more than the text-object beginning with "i" (like $\underline{\mathbf{i}}$ w).

There are more text-objects you can use for your editing needs. You can discover new ones in the bonus exercises at the end of this chapter, or by looking at Vim's help.



Help Yourself

:help text-objects

Undo and Redo (NORMAL mode)

I would be totally lost if I didn't have any way to undo or redo my work. Here's how to do so in NORMAL mode:

Keystroke	Description
u	To <u>u</u> ndo your last edit.
CTRL-r	To <u>r</u> edo.

You can think of CTRL-r as you being in control (CTRL) of your content.

You'll notice that whatever you're doing in INSERT mode is equivalent to one undo. We'll learn how to change this behavior later in the book.



It's Playtime!

Exercise C - Operators and Text Objects

Using the hjkl keys in NORMAL mode, move your cursor to the following position:

- 1. How can you delete inside a word, keeping the space(s) surrounding it?
- 2. How can you undo what you just did?
- 3. How can you delete the word local , this time with the space following it?
- 4. Undo your change. How would you delete the word <code>local</code> , and directly switch to <code>INSERT</code> mode, only using 3 keys?
- 5. Again, undo your change. Move to the a of the word local, and hit dw. Can you explain what happened? What's the difference between dw and diw?



Help Yourself

:help undo-redo

Bending Vim to Your Will (Customization)

In Vim, almost everything is configurable. It's insane, I tell you. You can shape your editor according to your megalomaniac desires. Let's see the very basics here.

Creating Your Configuration

Your main configuration file should be in the following path by default, depending on what you use:

Editor	File	
Vim	~/.vimrc.	
Neovim	<pre>\$XDG_CONFIG_HOME/nvim/init.vim</pre>	

This file is sourced when Vim starts. Except that, they have nothing special; you can create another file with some Vimscript inside (including Ex commands) and source it manually if you

want. We'll explore this functionality later in this book.

Whatever you're using, I'll call this configuration file *the vimrc* throughout the book. You can actually see what vimrc file is used if you run vim --version or nvim --version in your shell.

As you can see, the path of Neovim's vimrc depends on the environment variable TG_{DOME} . It's most likely Config. If you don't know what are the XDG user directories, here's a good resource to learn more about them.

Let's now write our first lines of configuration. I would encourage you to write them using Vim. What you've learned in this first chapter should be enough for you to put your toes into Vim's relaxing waters.

To open this file, you can run vim {path} in your terminal. For example: vim ~/.vimrc.

First, let's add the following to our config:

```
noremap <up> <nop>
noremap <down> <nop>
noremap <left> <nop>
noremap <right> <nop>
```

The Ex command noremap allows us to bind a key to something else. We'll see this command more in details in the third chapter of the book.

Here, we assign the arrow keys to... nothing. This will force you to use hjkl instead. It might be a bit painful at first, but, trust me, it's for your own good. You can also conclude that I'm just a little pig, and then decide to use the arrow keys all your life; I'll be sad, but it's your right.

Here's another line we can add to our vimrc:

```
set clipboard+=unnamedplus
```

It will make the copy-paste mechanism less confusing, till you learn more about it in Rank IV.

We now have Ex commands on every line of our file. That's right: you could also run them in COMMAND-LINE mode, but they would be reset the next time you're closing Vim. In fact, you can add any Ex command in your vimrc, and Vim will automatically execute them during its startup, in order.

We've already seen that these commands have a long and short form. You can use either of them in COMMAND-LINE mode; but, when you write these Ex commands in a vimrc, I would encourage you to use the long form for a better readability.

If you use Vim instead of Neovim, we need to add these lines too:

```
" No compatibility with Vi
set nocompatible

" Display line numbers
set number

" Enhanced completion in command-line mode
set wildmenu

" Syntax highlighting
syntax on

" Enable filetype, indentation, plugin
filetype plugin indent on
```

As you can see, every line beginning with a double quote " is a comment.

Some explanations:

- set nocompatible Vim looks less like Vi, its ancestor. That's great, because we don't want to use Vi, but Vim.
- set number Display the line numbers.
- syntax on Enable the syntax highlighting.
- filetype plugin indent on Load a bunch of files to set automatically the indentation, and other options depending on the type of files open in Vim.

To see the effect of your new configuration, you can relaunch Vim and try to use your arrow keys. They don't work anymore. Great! Out of some constraints can come great creativity.

With all this knowledge in mind, I advise you to add some comments for everything you add to your vimrc, at least at the beginning. For example:

```
" Easier copy-paste from other applications to Vim
set clipboard+=unnamedplus
```



Help Yourself

:help vimrc

The Configuration Addiction

At that point, I'd like to warn you: configuring Vim can become addictive. Not I-lost-my-house-and-my-partner-left-me kind of addictive, but you can easily spend (too) many hours trying to come up with the best configuration in the universe.

My advice: just try to add what's useful for you, step by step. Don't try to recreate all the functionalities you had in your text editor or, even worst, your IDE. You'll get eventually there when we'll speak about plugin a bit later in this book but, before that, you should consider trying to understand and use the functionalities directly available in Vim.

Exercises

Basics

To solve these exercises, open with Vim the file rank_01/functions.lua from the book companion. You can do so by going into the root of the repository and run vim rank_01/functions.lua.

Don't forget that you can quit Vim with the Ex-command :q. Add a *bang* to the command to quit without saving: :q!.

Each exercise has a series of question. You should solve them in order. The changes done for each question is the starting point for the next one.

Exercise 1 - Horizontal Motions, the Return

Using the hjkl keys in NORMAL mode, move your cursor at the beginning of the following line:

```
local function restorePosition()
   if vim.fn.line("'\"") > 1 and vim.fn.line("'\"") <= vim.fn.line("$") then
      vim.cmd([[normal! g`"]])
   end
end</pre>
```

Using only one key per question, how would you:

```
    Move to this position:
    if vim.fn.line("'\"") > 1?
    Move to this position:
    if vim.fn.line("'\"") > 1?
    Move to this position:
    if vim.fn.line("'\"") > 1?
    Move to this position:
```

Exercise 2 - Operators, Motions, and Text-Objects

Using the hjkl keys in NORMAL mode, move your cursor at the beginning of the word "vim" as follows:

```
local function restorePosition()
   if vim.fn.line("'\"") > 1 and vim.fn.line("'\"") <= vim.fn.line("$") then
      vim.cmd([[normal! g`"]])
   end
end</pre>
```

- 1. How would you move to the first parenthesis of the same line, only using two keys?
- 2. How would you delete everything inside these parentheses?
- 3. Undo your change.
- 4. How would you move your cursor at the "i" of "line", using only two keys (including the key i), as follows: if vim.fn.line("'\"") > 1?
- 5. How would you move to the "i" of "vim", using only one key, as follows: if vim.fn.line("'\"") > 1?

6. While staying on the same line, try to use $\, fn \,$, $\, Fn \,$, $\, tn \,$, $\, Tn \,$. Then, try to use $\, ; \,$ or $\, , \,$, to get used to these movements. Try to replace "n" with other characters present on the line, too.

Beyond the Basics

These exercises dive deeper in some concept seen in the chapter.

Exercise 3 - More Text Objects!

Using the hjkl keys in NORMAL mode, move your cursor at the beginning of the function "restorePosition" as follows:

```
local function restorePosition()
   if vim.fn.line("'\"") > 1 and vim.fn.line("'\"") <= vim.fn.line("$") then
      vim.cmd([[normal! g`"]])
   end
end</pre>
```

A paragraph is also a text object. It starts on an empty line, and end on the next empty line.

- 1. How would you delete the whole function restorePosition only using three keys?
- 2. Undo your change. How would you delete the next block of parenthesis, only using three letter keys?
- 3. Undo your change. How would you delete a sentence?
- 4. To find the start and end of the text-object "sentence", what Ex-command would you use?
- 5. What text-object could be useful to edit some HTML? Do you think this text-object exists in vanilla Vim?

Exercise 4 - More Motions!

Using the hjkl keys in NORMAL mode, move your cursor at the beginning of the function name "restorePosition" as follows:

```
local function restorePosition()
   if vim.fn.line("'\"") > 1 and vim.fn.line("'\"") <= vim.fn.line("$") then
      vim.cmd([[normal! g`"]])
   end
end</pre>
```

- 1. We've seen the text-object "paragraph" in the previous exercise. Using Vim's help, try to find what motion allowing you to move from one paragraph to another.
- 2. How would you delete the letter r in local function restorePosition() with two letter keys? One?
- 3. Undo your change. How would you delete the space before your cursor: local function restorePosition() with two letter keys?
- 4. The answers of the two previous questions use motions or text-objects?

Exercise 5 - Up and Down Following Indentations

Using the hjkl keys in NORMAL mode, move your cursor on the character 1, at the beginning of the word local:

```
local function restorePosition()
   if vim.fn.line("'\"") > 1 and vim.fn.line("'\"") <= vim.fn.line("$") then
      vim.cmd([[normal! g`"]])
   end
end</pre>
```

How would you:

- 1. Move to the next line, on the i of if, only using one key?
- 2. Move back to the starting position, only using one key?

Beyond the Basics

Exercise x - Motion

- How would you append at the end of the previous word?
 - ge

Playtime Solutions

Exercise A - Horizontal Motions

Question	Keystroke	Result
start		local function restorePosition()
1.	b	<pre>local function restorePosition()</pre>
2.	W	<pre>local function restorePosition()</pre>
3.	\$	<pre>local function restorePosition()</pre>

Exercise B - Vertical Motions

Question	Keystroke	Result	
start		#!/usr/bin/env lu	ıa
1.	5G	vi	m.cmd([[normal! g`"]])
2.	1G or GG	#!/usr/bin/env lu	ıa
3.	G	3	
4.	CTRL-u	V	<pre>im.cmd([[normal! g`"]])</pre>

Note that you can also move at the fifth line of the file in COMMAND-LINE mode, with :5.

Exercise C - Operators and Text Objects

Question	Keystroke	Result
start		<pre>local function restorePosition()</pre>
1.	diw	<pre>function restorePosition()</pre>
2.	u	<pre>local function restorePosition()</pre>
3.	daw	function restorePosition()
4.	ciw	function restorePosition()
5.	dw	<pre>locfunction restorePosition()</pre>

5. All the NORMAL mode keystrokes dw , daw , and diw use the operator d (for d elete).

The difference between these keystrokes: w is a motion, aw and iw are text-objects.

If you:

- Use a motion without operator, you'll move your cursor.
- Use a motion with an operator, you'll operate from the cursor position to the destination of the motion.

That's why dw, in this example, delete from the cursor position to the beginning of the next word function.

Said differently, the start and the end of a text-object are always the same: for example, the start of aw will be the first letter of the word the cursor is on, and the end will be the potential space after the word.

Another difference: a text-object always need to be prefixed by an operator. A motion can be used on its own.

Exercises Solutions

Exercise 1

Question	Keystroke	Result
start		<pre>if vim.fn.line("'\"") > 1</pre>
1.	^ or w	<pre>if vim.fn.line("'\"") > 1</pre>
2.	W	<pre>if vim.fn.line("'\"") > 1</pre>
3.	W	<pre>if vim.fn.line("'\"") > 1</pre>
4.	0	<pre>if vim.fn.line("'\"")</pre>

Exercise 2

Question	Keystroke	Result
start		<pre>if vim.fn.line("'\"") > 1</pre>
1.	f(or t"	<pre>if vim.fn.line("'\"") > 1</pre>
2.	da(Or dab	<pre>if vim.fn.line > 1</pre>
3.	u	<pre>if vim.fn.line("'\"") > 1</pre>
4.	Fi	<pre>if vim.fn.line("'\"") > 1</pre>
5.	;	<pre>if vim.fn.line("'\"") > 1</pre>

Exercise 3

Question	Keystroke	Result
start		local function restorePosition()
1.	dap Or dip	<pre>local function deleteTrailingWS()</pre>
2.	dab	local function restorePosition
3.	das Or dis	g ("]])

- 4. You first need to go to COMMAND-LINE mode by typing : . Then, you need to use the help Ex-command help, followed by what you want to find, i.e sentence, or text-object. In short: :help sentence or :help text-boject.
- 5. The text object at or it stand for "around an HTML tag" and "inside an HTML tag", respectively. Add whatever operator as a prefix to edit your HTML easily.

Exercise 4

1. You'll find this info by running the Ex-command :help object-motions. You can use the motion { to move up a paragraph, } to move down.

Other than that, you could try :help cursor-motions and search for "paragraph", by running /paragraph.

Question	Keystroke	Result
start		local function restorePosition()
2.	dl or x	local function estorePosition()
3.	dh	<pre>local functionrestorePosition()</pre>

4. We're using motions here: 1 is the motion to move one character to the right, h to move one character to the left.

The letter 1 is also a useful mnemonic for 1 etter. Even if it's easier to use \times to delete a character, you can use other operators with the motion 1, like y1 if you want to copy it.

Exercise 5

Question	Keystroke	Result
start		<pre>local function restorePosition()</pre>
1.	+	<pre>if vim.fn.line("'\"") > 1</pre>
2.	-	<pre>local function restorePosition()</pre>

Rank II - Vim for Beginners

Greetings, Vim explorer! Welcome to the second chapter. How do you feel? Good? Tired? Neutral? Happy? Ecstatic?

Don't forget that if you feel overwhelmed, just stop and do something else. You can take the knowledge you've already assimilated in these first chapters and try to apply it while editing some plain text files, to really understand Vim's basics. They are important! Many other advanced concepts are build on these first keystones.

Now, it's time to dive deeper into the concepts we saw in the first chapter, as well as adding even more modes to our toolkit. More specifically, we'll see:

- We'll dive more into the NORMAL and INSERT modes, and we'll introduce the VISUAL mode.
- We'll see more ways to delete some text, and the curious side-effects doing so.
- I know you'll panic. They all do. That's why we'll see more in depth how to use Vim's internal help. It's great, apple-like awesome.
- Finally, we'll continue to build Vim's personal configuration.

Take your bag full of modes, motions, operators, text objects, and let's adventure in the Holy Land of Vim once more.

The Modes of Vim, the Return

Vim's modes are like mountains: they're quite easy to understand (it's a big pile of rocks), but you can always dig deeper. To continue my stupid analogy, the NORMAL mode is the Everest of all the modes: we'll dig it in this section as well as in the whole book, until the end of time.

We'll also try to ascend a new mode: the **visual** mode.

Counting Motions or Text-Objects in NORMAL mode

Throughout this book, we'll see that Vim is the champion for automating mundane, annoying tasks we need to do too often. What doesn't require our human brain should be automated, to use what's left of our neurones to more important thoughts and tasks.

Most NORMAL mode motions we've seen in the previous chapter can be repeated as many times as we want. To do so, we need to add add a *count* before the motion. The count determines how many times the motion will be applied.

Actually, there's always a count applied to a motion when you used them; it just happen that the default, in most case, is 1. It's also true for many NORMAL mode keystrokes.

For example, you can open the file "rank_01/functions.lua" from the book companion in Vim, and you can try to put your cursor on the first character of the 3rd line, as follows:

```
local function restorePosition()
```

Now, try to hit 2w in NORMAL mode. Here's the result:

```
local function restorePosition()
```

You've applied the motion "word" two times! You can try to add a count to the motions you already know to get used to it.

If you apply a count to both a motion and an operator, they are multiplied. For example, if you hit 2d3w, you'll delete 6 words.

Now you might wonder: is it really useful? Do you see yourself counting words or other text-objects, to move where you want? If you experiment a bit with counts, you'll see that it's not easy to look ahead and know exactly what count you need.

That said, when you don't want to move your cursor too far away, count can be useful. Not so much when you need to do an operation multiple lines away.

We'll come back to this functionality throughout the book, to see more practical applications of counts.

Many keystrokes described in this book will often be preceded with <code>[count]</code>; it means you can (but don't have to) add a count before hitting the keystroke.

More Keystrokes to Switch to INSERT Mode

As we saw in the first chapter, Vim requires you to switch between NORMAL and INSERT mode quite often. That's why there are many NORMAL mode keystrokes to switch to INSERT mode in slightly different ways.

Here are the main ones:

Keystroke	Description
i	To <u>i</u> nsert content before the current character.
a	To insert content <u>a</u> fter the current character.
A	To insert content <u>a</u> fter everything, at the end of the line.
[count]o	To open a new line below the current one and switch to
	INSERT mode.
	This inserted new line will be repeated if there is a [count].
[count]0	To open a new line above the current one.
ESC or CTRL-c or CTRL-[Switch back from INSERT mode to NORMAL mode.

The more you'll use these keystrokes, the more seemliness your switch between NORMAL and INSERT mode will be. Soon, you won't even notice that you switch between both. Guaranteed!



Help Yourself

:help Insert-mode

The VISUAL Mode

There is a third important mode in Vim you'll often use: VISUAL mode. Its goal? Selecting a piece of text.

How to switch from NORMAL mode to VISUAL mode? You might already have guessed it: you need to hit v in NORMAL mode. You'll see the indicator --VISUAL-- appearing in the bottom left corner of Vim. The selection will start at the cursor position; you can then hit some motions to extend the selection, with an operator to operate on the selected text.

When hitting ν in NORMAL mode, you enter the VISUAL mode per character. If you want to select entire lines at once, you can enter VISUAL mode linewise. To do so, you need to use SHIFT- ν (or V) (in uppercase), and then going up and down will select what you want.

To summarize:

Keystroke	Description
V	Switch to $\underline{\mathbf{v}}$ ISUAL mode per character.
V	Switch to $\underline{\mathbf{v}}$ ISUAL mode linewise.

There is also the VISUAL mode blockwise, but we'll see that later in the book.

A last thing: VISUAL mode is convenient because it allows us to operate upon a specific portion of text, in a very... visual way. That said, it's often quicker for Vim experts to only use motions, text-objects, and count, because you don't spend your time selecting text. So, if you want an advice, try to train yourself not using VISUAL mode. Not necessarily at the beginning, but when you'll be more comfortable with Vim.

As usual, to come back from VISUAL mode to NORMAL mode, press ESC.



Help Yourself

:help visual-mode

The REPLACE Mode

The last mode I'd like to highlight is the REPLACE mode. As it names indicate, it's a mode where you can replace some text.

To replace a first character by a second one:

- 1. Move your cursor on the first character.
- 2. Hit r in NORMAL mode.
- 3. Hit the second character.

And voila! As usual, you can also use a count to replace a specific number of characters with one character.

If you want to replace more than one character, you can hit R in NORMAL mode. You'll then replace everything you're typing.

To summarize:

Keystroke	Description
R	Switch to <u>r</u> eplace mode.
[count] r	Switch to $\underline{\mathbf{r}}$ eplace mode for $[\mathtt{count}]$ characters (default: 1).

You already know how to come back from REPLACE mode to NORMAL mode: yes, with the ESC key!



It's Playtime!

Exercise A - Vim's Modes

Using the hjkl keys in NORMAL mode, move your cursor at the following position:

For each question, undo all your changes and come back to the initial cursor position.

How would you:

- 1. Move to the third v on the line, using only a NORMAL mode keystroke of 3 characters?
- 2. Replace the first vim of the line with nop using REPLACE mode?
- 3. Delete the whole line using VISUAL mode?
- 4. Change the first v of vim by a w, by using a NORMAL mode keystroke of 1 character followed by one letter in INSERT mode?
- 5. Create a new line below and enter INSERT mode, using only a NORMAL mode keystroke of 1 character?
- 6. Insert a semi colon ":" at the end of the line, by using a NORMAL mode keystroke of 1 character followed by the semi-colon in INSERT mode?



Deleting In Vim (NORMAL mode)

In the first chapter, we've already seen how to delete text with the operator d. With a motion or a text-object, we've already have a powerful tool to get rid of all these annoying character.

Cross What You Don't Want

Here are two useful keystrokes to delete single characters in NORMAL mode:

Keystroke	Description
x	Delete the character under the cursor
X	Delete the character before the cursor

Both are aliases for d1 and dh, which are the combinations of the $\underline{\mathbf{d}}$ elete operator and the motions "1" (right) and "h" (left).

Deleted Content To

There is something many Vim beginners find surprising, and quite annoying. For example, let's say that you've the following content:

```
local function restorePosition()
```

Then, let's try the following in NORMAL mode:

- 1. Hit y to yank (copy) the word local.
- 2. Then, move to the f of function by hitting the motion w.
- 3. Let's then hit diw for delete inside word.
- 4. Let's now hit p to paste (or put) our content. What will be inserted here?

Here's the result

```
local functionrestorePosition()
```

If you did all the steps above, you'll see that function will be added to the line, even if you copied local before!

For now, you could think of this behavior as throwing whatever you delete into your clipboard, as it was a trash bin. This will make more sense when we'll look at Vim's registers, a great functionality. From there, we'll have the opportunity to choose what we want to paste (what we've yanked, what we've deleted, and more). Till then, bear with me; it might be quite annoying at first, but it's worst the pain.



It's Playtime!

Exercise B - Deleting in Vim

Using the hjkl keys in NORMAL mode, move your cursor at the beginning of restorePosition:

```
local function restorePosition()
   if vim.fn.then
      vim.cmd([[normal! g`"]])
   end
end
```

Use Vim's search for each question, from the cursor position you moved on for the previous question. How would you:

- 1. Move to the r of restorePosition?
- 2. Delete the word restore using a three letter keystroke in NORMAL mode?
- 3. Replace the uppercase P of position with a lowercase p?
- 4. Move to the n of position?
- 5. Paste what you've deleted, to end up with positionrestore?
- 6. Delete the e of positionrestore with one-letter NORMAL mode keystroke?



Help Yourself

:help deleting

Searching (COMMAND-LINE Mode)

We can hit : in NORMAL mode to switch to COMMAND-LINE mode and run Ex commands, as we saw in the first chapter. We can do much more in COMMAND-LINE mode, however.

If you hit / or ? in NORMAL mode, you'll also switch to COMMAND-LINE mode. But this time, it's not for running Ex commands; it's for searching a pattern in our text.

Here's the basics:

Command	Description
/{pattern}	Search {pattern} forward from the cursor position.
?{pattern}	Search {pattern} backward from the cursor position.

When you hit ENTER after typing your {pattern}, you'll switch back to NORMAL mode and you'll move directly to the first match in your text. If there is no match, Vim will display the error "Pattern not found".

If there are more than one match, you can move to them by repeating the search. Here are the different NORMAL mode keystrokes to do so:

Keystroke	Description
n	Repeat the last search forward (move to the $\underline{\mathbf{n}}$ ext match).
N	Repeat the last search backward.

That's not all: searching is a motion. It means that you can use an operator (as we saw in the first chapter), and then type your search. The operator will operate from the cursor position to the first match.

Let's take an example in our favorite file:

```
local function restorePosition()
   if vim.fn.line("'\"") > 1 and vim.fn.line("'\"") <= vim.fn.line("$") then
      vim.cmd([[normal! g`"]])
   end
end</pre>
```

If you then hit / in NORMAL mode, you'll move to the bottom of the screen. From there, you can type the pattern you want to search for. For example: vim . Hitting ENTER will bring your cursor to the next string "vim" and switch back to NORMAL mode.

Then, hitting n two times will bring you at the end of the line:

```
local function restorePosition()
   if vim.fn.line("'\"") > 1 and vim.fn.line("'\"") <= vim.fn.line("$") then
      vim.cmd([[normal! g`"]])
   end
end</pre>
```



It's Playtime!

Exercise C - Vim Search

Using the hjkl keys in NORMAL mode, move your cursor at the beginning of restorePosition:

Using Vim's search, how would you:

- 1. Move your cursor to this position: if $v \text{ im.fn.line}("' \setminus "") > 1$
- 2. Move to the next match of your search pattern?
- 3. Move back to the first match of your search pattern?
- 4. Move to the 4th match of your search pattern, using a count?

Navigating Vim's Help (COMMAND-LINE Mode)

As we saw in the first chapter, Vim's help is your most previous ally in your Vim's adventure. I don't know everything regarding Vim, and this book won't even explain everything I know. Many details are left out, for this book not being too boring, and for you not to be lost in details.

In contrast, vim's help knows almost everything. That's why, if you need to dig deeper into a functionality, it should be your first reflex. Stack overflow won't cut it.

Basics

Here are the most useful help commands you can use:

Ex Command	Short Name	Description
:help	: h	Open the main help file.
<pre>:help {subject}</pre>	<pre>:h {subject}</pre>	Open the help about {subject} in a split window.
<pre>:helpgrep {pattern}</pre>	<pre>:helpg {pattern}</pre>	Search the {pattern} in the help files.

I always reach for <code>:help {subject}</code> first when I need to look at a specific functionality. If I don't find what I want, I reach then for <code>:helpgrep {pattern}</code>; this Ex command will have a bigger chance to find a false positive, however.



Help Yourself

:help helphelp
:help helpgrep

Follow The Definition

Vim's help are simple text files. Thanks to a tag file, you can also jump to the definition of some specific words. We'll come back to this concept of tags much later in this book.

For now, let's just say that you can get to the definition of some keywords in this set of help files. This is really useful to look at related subjects you might want to dive into. If you're syntax highlighting is on (running <code>:syntax on in COMMAND-LINE mode</code>, as we saw in the first chapter), these keywords will appear in different colors in Vim. If not, you can find them between two pipes <code>|</code>.

If you want to follow one of these keywords, you can place your cursor on it and hit CTRL-]. You'll then *jump* to another part of the same file, or even to another file. To jump back, you can hit CTRL-o.

Let's try it. First, run the Ex command :help . A new window will open vertically. Welcome to the main help file!

Then, place your cursor on bars, as follow:

```
Close this window: Use ":q<Enter>".

Get out of Vim: Use ":qa!<Enter>" (careful, all changes are lost!).

Jump to a subject: Position the cursor on a tag (e.g. | bars|) and hit CTRL-].

With the mouse: Double-click the left mouse button on a tag, e.g. |bars|.

Jump back: Type CTRL-O. Repeat to go further back.
```

Finally, hit CTRL-]. You'll jump to the end of the same file. To come back to where you were, hit CTRL-o. To close the window, you can run :q in COMMAND-LINE mode.

This is a very effective, and useful way to navigate Vim's help files!

A last tip: to display the table of content, you can hit go in NORMAL mode. It will open a new window where you can select the subsection you want to read. If the window is already open, hitting go will move your cursor inside it.

Again, to close the window, you can run :q in COMMAND-LINE mode.

To summarize:

Keystroke	Description
CTRL-]	Jump to the definition of the keyword under the cursor.
CTRL-o	Jump back to your <u>o</u> lder cursor position.



Help Yourself

:help
:help CTRL-]
:help CTRL-0

Finding What Your Heart Desire

We've seen just above that you can search for different functionalities using $help \{subject\}$. But what should be this $\{subject\}$?

You can try to find it by yourself of course, but it's sometimes difficult to really know where to find the information you seek, especially when you get familiar with Vim. That's why this book gives you many help Ex commands for each section.

Here are other, more precise patterns you can use depending on what you want to find:

Ex Command	Description
:help CTRL-{char}	Open the help for a keystroke with CTRL and a {char} in NORMAL mode.
<pre>:help i_CTRL-{char}</pre>	Open the help for a keystroke with CTRL and a {char} in INSERT mode.
<pre>:help c_CTRL-{char}</pre>	Open the help for a keystroke with CTRL and a {char} in COMMAND-LINE mode.
:help -{arg}	Open the help for the {arg} we can pass to the "vim" command.

To illustrate, here are some examples of Ex commands you can run in Vim:

Ex Command	Description
:help CTRL-o	Help for the NORMAL mode keystroke CTRL-0.
:help i_CTRL-o	Help for the INSERT mode keystroke CTRL-0.
:help c_CTRL-f	Help for the COMMAND-LINE mode keystroke $$ CTRL-f $$.
:help -u	Help for the Vim's command line option -u .

There are many more pattern for help's subjects. We'll discover them as we introduce the concepts they are about.



It's Playtime!

Exercise X - Reference Manual (Vim's Help)

- 1. In this command, what is optional and what is not?
 - [range]move {line}

Vimscript or Lua? (Customization)

The content of your vimrc and the different plugins for Vim are Ex-commands, as we saw. But Ex-commands are just a part of larger programming language called Vimscript.

At the beginning of time, Vimscript was created to configure Vim, and nothing more. It's straightforward to use for this limited job; Vimscript has useful constructs to customize Vim as we see fit. That's why we use it.

But, over the years, more and more constructs were added to Vimscript. Step by step, it became a general programming language. It's where things become to get ugly. As a programming language, Vimscript has many pitfalls and weird design decisions. It's painful to understand, use, and debug.

Vimscript is not the only option, nowadays. It's also possible to use Lua to configure Vim. It's where the difference between Vim and Neovim is the most obvious: while Neovim allow you to configure almost everything in Lua, Vim doesn't allow you as much.

The next versions of Vim will come with Vimscript 9, a new and faster version of the language which is trying to fix Vimscript quirks. But, as I write these lines, it's not officially out yet.

All in all, most of the configuration of this book will be in Vimscript. When we'll need a more general programming language to push the customization further, I'll give both Vimscript and Lua versions.

Why not doing everything in Lua at the first place, you might wonder? For different reasons:

- 1. Vim users can't configure as much in Lua as Neovim's users.
- 2. Many good resources and useful functions available out there are written in Vimscript. Knowing a bit about it can help you understand them, and even modify them for your needs.
- 3. Vimscript will be likely supported for a long time by both Vim and Neovim.
- 4. Many Vimscript functions can be called from Lua script. Knowing them is essential, even if you prefer doing everything in Lua.

If you really want to write your whole configuration in Lua, you can find 43270724983 resources online to do exactly that. I'm not kidding; it's trendy nowadays to switch your vimrc from Vimscript to Lua.

Vim's Options (Customization)

We've began to configure our vimrc in the next chapter. Let's now look at a cornerstone of Vim configuration, the famous Vim's options.

Defining Options

You can modify Vim's general behavior by modifying its options. You can think of options as variables with some scope; you can display their values, or modifying them using Ex commands in COMMAND-LINE mode. An option can be either a boolean, a string, or a number.

If you want to permanently modify Vim's options, you need to assign their new values directly in your vimre, as we did with the option clipboard in the previous article for example.

Here are the commands you can use: to manage these options:

Ex Command	Description	Туре
<pre>:set no<option> :set <option>! :set <option>?</option></option></option></pre>	Unset the {option} Toggle the {option} Return the {option} 's value Reset the {option} to its default value	Boolean Boolean All All
<pre>:set <option>& :set {option}={value} :set {option}+={value}</option></pre>	Set the {value} to the {option} Add the value {value}	String or number Number
<pre>:set {option}-={value}</pre>	Append a string {value} Subtract the value {value} Delete the string <value></value>	String Number String

For example, if you want to display the filetype of the current file open, you can run this Ex command:

```
:set backspace?
```

The string option backspace is made of multiple substrings separated with commas ','. You can delete one of these substrings as follow:

```
:set backspace -= indent
```

You can now look at its new value:

```
:set backspace?
```

To come back to its default value, you can run:

```
:set backspace&
```

Remember: you don't need the prefix : if you want to put some Ex commands in your vimrc, like setting some options.

Here's an important tip to remember: if you want to get some information about options in Vim's help, you need to surround your pattern with single quotes. For example:

```
:help 'number'
:help 'swapfile'
```

Lastly, it's a bit difficult to set values for options of type "string". You'll have to escape all spaces and other characters for it to work. Here's another way to set this options we'll see again as we progress through the book:

```
:set list listchars=tab:\ \ ,trail:.
:let &listchars='tab: ,trail:.'
```

Using :let here allow us to use single quote for our string, and not bother with escaping characters with a backslash \land .



Help Yourself

:help options
:help set-options
:help option-list
:help :let-option

Setting Options The Interactive Way

There's another way to modify Vim's options: using the Ex command :options.

If you run it, another window will open. From there, you can see all the options you have access to, and you can set them by hitting ENTER on the value you want. For example:

```
compatible behave very Vi compatible (not advisable)
set noocp cp
```

The cursor here is on <code>nocp</code>, so if you hit <code>ENTER</code> you'll set the option <code>compatible</code> to false. For string values, you can modify them and then hit <code>ENTER</code> to set them.

These options are grouped by general functionality, so it can be a good way to experiment and discover new options. The number of options can be a bit daunting, however; don't try to set all of them at once, or you'll burn out by so many... possibilities!

Useful Options

Here are the options we've seen in the previous chapter:

Option	Description	Туре	Recommended value
compatible	Vim becomes compatible with Vi	Boolean	False
number	Display line numbers	Boolean	True
wildmenu	Enhanced completion in COMMAND-LINE mode.	Boolean	True

Here are three more interesting options related to the search we've seen above:

Option	Description	Type	Default
'ignorecase'	Search is not case sensitive	Boolean	off
'smartcase'	Search is case sensitive if the pattern has one	Boolean	off
	more uppercase. Needs ignorecase to be true.		
'showcmd'	Show partial NORMAL mode keystrokes	Boolean	on
	on the right of the command-line window		
'hlsearch'	Highlight the matching search pattern.	Boolean	on on Neovim, off on V
	Use :nohlsearch (or :noh) to turn the highlight off.		
	The next matching pattern will be highlighted, however.		
autowriteall	Automatically write open files.	Boolean	off
incsearch	Display the matches in the current buffer while searching.	Boolean	on

We'll see many more options throughout the book; you'll have the occasion to set them up for your own specific needs.



It's Playtime!

Exercise D - Manipulating Options

How would you:

- 1. Get the value of the option 'filetype'?
- 2. Toggle the value of the option 'number'?
- 3. Switch the value of the option 'compatible' to false?
- 4. Append the substring "S" to the existing value of the option 'shortmess'?
- 5. Only delete the substring "S" to the existing value of the string option 'short-mess'?
- 6. Set back the option 'shortmess' to the default value?

Options are useful, and we'll use them all the time in the following chapters. Be prepared to fill your life with them!

Exercises

Basics

Exercise 1 - Help Yourself!

Using Vim's help, how would you:

- 1. Find information about the REPLACE mode?
- 2. Find information about the keystroke * in NORMAL mode?
- 3. Find information about the keystroke CTRL-a in INSERT mode?
- 4. Find information about the keystroke CTRL-f in COMMAND-LINE mode?
- 5. Find information about the option iskeyword?

Exercise 2 - Search Highlighting

Using what you've seen in this chapter and Vim's help, how would you:

- 1. Set the option 'hlsearch'?
- 2. Search for the word vim.
- 3. The matching pattern vim should now be highlighted. How would you turn off the highlight, but still highlight the next search you'll do?
- 4. How can you totally disable matching pattern highlight?

Exercise 3 - More Vim Search

Using the hjkl keys in NORMAL mode, move your cursor at the beginning of restorePosition:

```
local function restorePosition()
   if vim.fn.then
      vim.cmd([[normal! g`"]])
   end
end
```

Use Vim's search for each question, from the cursor position you moved on for the previous question. How would you:

```
    Move your cursor to this position: vim.fn. line("$") then
    Move your cursor to this position: vim.cmd([[normal! g`"]])
    Move your cursor to this position: if vim.fn. line("'\"") > 1
```

- 4. Yank everything until the word then at the end of the line.
- 5. Delete everything until the word then at the end of the line.

Beyond the Basics

Exercise xxx - Vim Modes

- 1. Here's a table. How would you fill the third column?
 - Using shift+r to start replace mode and not disturb the table...

Exercise xxx - Options

Better to put string like that, especially regexes; we'll speak about it again in another "beyond the basics" exercises in rank when we speak about Vim regexes

Exercise 4 - Swapping

Using the hjkl keys in NORMAL mode, move your cursor on the character 1 at the beginning of the word local:

```
local function restorePosition()
   if vim.fn.line("'\"") > 1 and vim.fn.line("'\"") <= vim.fn.line("$") then
      vim.cmd([[normal! g`"]])
   end
end</pre>
```

Undo all your changes after each question to come back to the starting position above.

How would you:

- 1. Swap the character 1 you're on, with the following character o, only using two keys?
- 2. Swap the current line you're on, with the line just below, only using three keys?
- 3. Swap the word local with the word function using five keys?

Playtime Solutions

Other solutions than the one presented here are possible.

Exercise A - Vim's Modes

Question	Keystroke	Result
start		<pre>if vim.fn.line("'\"")</pre>
1.	2fv	<= vim.fn.line("\$") then
2.	Rnop	<pre>if nop.fn.line("'\"")</pre>
3.	Vd	<pre>vim.cmd([[normal! g`"]])</pre>
4.	rw	<pre>if wim.fn.line("'\"")</pre>
5.	0	
6.	A:	<pre><= vim.fn.line("\$") then:</pre>

Exercise B - Deleting in Vim

Question	Keystroke	Result
start		<pre>local function restorePosition()</pre>
1.	fr	<pre>local function restorePosition()</pre>
2.	dtP	local function Position()
3.	rp	<pre>local function position()</pre>
4.	fn	<pre>local function position()</pre>
5.	p	local function positionrestore()
6.	X	local function positionrestor()

Exercise C - Vim's Search

Question	Keystroke	Result
start		#!/usr/bin/env lua
1.	/vim then ENTER	<pre>if vim.fn.line("'\"") > 1 and vim.fn.line("'\"")</pre>
2.	n	<pre>if vim.fn.line("'\"") > 1 and vim.fn.line("'\"")</pre>
3.	N	<pre>if vim.fn.line("'\"") > 1 and vim.fn.line("'\"")</pre>
4.	3n	<pre>vim.cmd([[normal! g`"]])</pre>

Exercise D - Manipulating Options

```
    :set filetype? or :set ft?
    :set number! or :set nu!
    :set nocompatible or :set nocp
    :set shortmess+=S or :set shm+=S
    :set shortmess-=S or :set shm-=S
    :set shortmess& or :set shm&
```

Exercises Solutions

Exercise 1 - Help Yourself

```
    :help replace-mode
    :help *
    :help i_CTRL-a
    :help c_CTRL-f
    :help 'iskeyword'
```

Exercise 2 - Search Highlighting

```
    :set hlsearch
    /vim
    :nohlsearch Or :nohl
    :set nohlsearch Or :set hlsearch!
```

Exercise 3 - More Vim Search

Question	Keystroke	Result
start		<pre>local function restorePosition()</pre>
1.	/line $then\ nn$	vim.fn.line("\$") then
2.	/vim	<pre>vim.cmd([[normal! g`"]])</pre>
3.	?line	<pre>if vim.fn.line("'\"") > 1</pre>
4.	y/then	<pre>if vim.fn.line("'\"") > 1</pre>
5.	d/then	if vim.fn.then

Beyond The Basics Solutions

Exercise 4 - Swapping

Question	Keystroke	Result
start		local function restorePosition()
1.	хр	olcal function restorePosition()

Question	Keystroke	Result
2.	ddp	<pre>local function restorePosition()</pre>
3.	dwelp	<pre>function local restorePosition()</pre>