Extracted from:

Modern Vim

Craft Your Development Environment with Vim 8 and Neovim

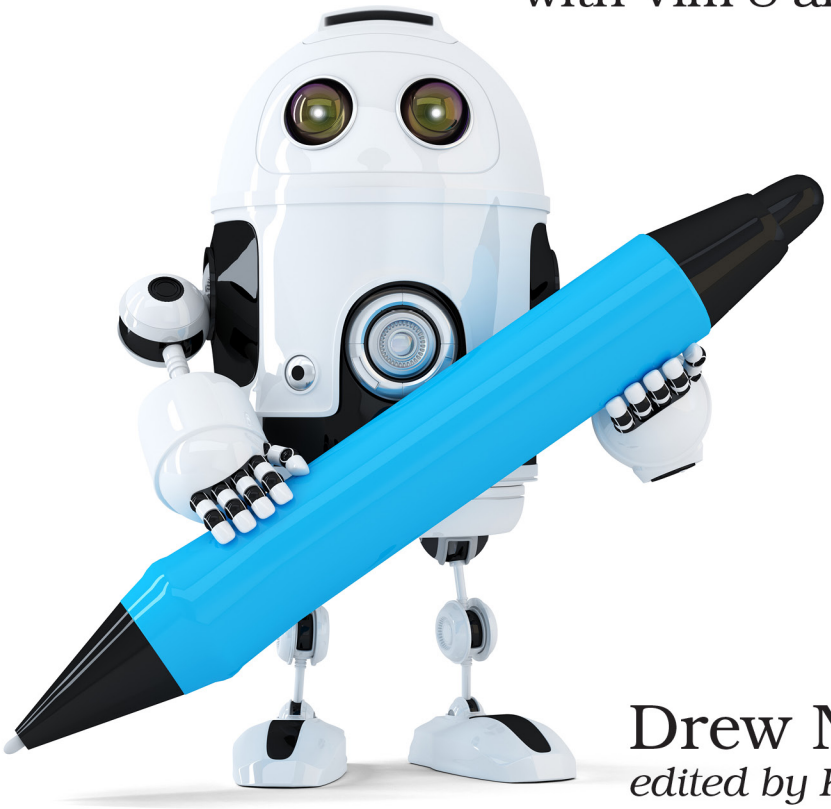
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Modern Vim

Craft Your Development Environment with Vim 8 and Neovim



Drew Neil

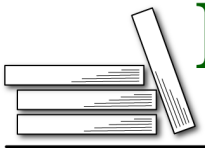
edited by Katharine Dvorak

Modern Vim

Craft Your Development Environment with Vim 8 and Neovim

Drew Neil

The Pragmatic Bookshelf
Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Brian MacDonald
Supervising Editor: Jacquelyn Carter
Development Editor: Katharine Dvorak
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-262-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—May 2018

Introduction

Vim's core functionality makes it a good programmer's text editor. Turning Vim into a full-blown development environment means combining it with other tools and extending its capabilities with plugins. In *Practical Vim [Nei15]*, I focused on the core features of the editor. In this book, I show you how to extend Vim and make it the centerpiece of a Unix-based IDE.

How This Book Is Structured

Modern Vim is a recipe book. It's not designed to be read from start to finish. Each chapter is a collection of tips that are related by a theme, and each tip demonstrates a particular feature in action. Some tips are self-contained. Others depend upon material elsewhere in the book. Those tips are cross-referenced so you can find everything easily.

Modern Vim doesn't progress from novice to advanced level, but each individual chapter does. A less-experienced Vim user might prefer to make a first pass through the book, reading just the early tips in each chapter. A more advanced user might choose to focus on the later tips or move around the book as needed. If it helps, you can think of this as a "Choose Your Own Adventure" book.

A Note on Vim Versions

To follow the tips in this book, you're going to need an up-to-date installation of Vim. (The clue is right there in the book's title!) You have two options: use version 8 of Vim or version 0.2 of Neovim.

Vim Version 8

Version 8 of Vim was released in September 2016. It introduced some new features that you'll learn about in this book, such as packages and job control. As a minimum requirement, you'll need to be running version 8 of Vim,

compiled with the huge feature set. You'll find instructions on how to install Vim 8 in Tip 1, *Installing Vim 8*, on page ?.

All of the tips in this book have been tested with version 8.0 of Vim, apart from a handful of tips which have been written especially for Neovim.

Neovim

Neovim is a community-run fork of Vim that can be used as a drop-in replacement for Vim. It supports all of the same features Vim 8 offers and more. You'll find instructions on how to install Neovim in Tip 2, *Switching to Neovim*, on page ?.

All of the tips in this book have been tested with Version 0.2.2 of Neovim.

Terminology

In many ways, Vim 8 and Neovim are interchangeable. When I use the word "Vim" by itself, you can read that as "Vim 8," or you can read it as "Neovim." If I want to make a specific point about one particular version of Vim, then I will specify "Vim 8" or "Neovim" to make that clear.

If you see this signpost at the start of a tip, it means that the tip is relevant only for Neovim:


Neovim only

If a tip only applies to Vim 8, you'll see a signpost like this:


Vim 8 only

If you see no such signposts at the start of a tip, then that tip should work just as well in both versions. Most of the tips in this book work in both Vim 8 and Neovim.

Contextual Instructions Using $VIMCONFIG

Vim 8 and Neovim follow different conventions on where to keep their configuration files. Vim 8 typically places them in a ~/.vim directory, whereas Neovim uses the ~/.config/nvim directory. These are important details, but it would get distracting if I mentioned them every time I referenced a runtime file.

To avoid this problem, we'll refer to certain files and directories using environment variables $MYVIMRC, $VIMCONFIG, and $VIMDATA. When you see $VIMCONFIG,

you can interpret that as ~/.vim if you are using Vim, or ~/.config/nvim if you are using Neovim. You'll find complete instructions on how to interpret these variables in *Contextual Instructions for Vim*, on page ? and *Contextual Instructions for Neovim*, on page ?.

Other Software Requirements

In *Modern Vim*, many of the lessons are illustrated with practical examples. You'll learn best if you actually follow the examples, and in some cases that means you're going to need to run other programs besides Vim.

JavaScript, Node.js, and npm

Many examples in this book are illustrated using JavaScript, which has become something of a universal language in recent years. Even if JavaScript is not your first choice for a programming language, you probably know enough "pidgin JavaScript" to be able to follow the examples in this book. All of the Vim features that are demonstrated for JavaScript can be adapted for other languages.

If you want to execute the JavaScript examples in this book, you'll need to install the Node.js[1] runtime, as well as the package manager npm.[2] Check out their websites for installation instructions.

Bash Shell (Or Any Shell)

Some of the tips in this book involve running commands in a shell. The examples are written assuming that you use the bash shell, because this is the default shell on many systems.

I don't mean to suggest that you *should* be using bash. If you prefer to use zsh, fish, or another shell, that's cool. You've invested time customizing your shell, so you should be prepared to spend a little bit more time adapting my instructions to make them work for your setup. You shouldn't have any trouble with this, since we only use basic features of the shell.

Git

Throughout this book you'll find instructions for running git commands, such as clone, init, and commit. You'll need an up-to-date installation of Git. You can find instructions for installing Git online.[3]

1. https://nodejs.org
2. https://www.npmjs.com
3. https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Ripgrep

The Ripgrep tool by Andrew Gallant[4] makes a couple of appearances. Much like grep, the primary purpose of Ripgrep is to search files for a pattern, and you'll see it used this way in Tip 13, *Searching Files with Grep-Alikes*, on page ?. Ripgrep also has a neat bonus feature: running `ripgrep --files` lists all the files beneath the current working directory, minus those that are ignored by your version control system. You'll see this feature put to use in Tip 7, *Finding Files Using Fuzzy Path Matching*, on page ?.

Depending on which platform you're using, you may be able to install Ripgrep using your package manager. If that doesn't work, take a look at the release page on GitHub.[5] There, you'll find pre-built binaries for Linux and Mac.

Don't worry if you can't get Ripgrep to work on your machine. It's nice to have, but you can get by fine without it.

Notation for Simulating Vim on the Page

`Ctrl-s` is a common convention for representing chordal key commands. It means "While holding down the `Ctrl` key, press the `s` key." But this convention isn't well suited to describing Vim's modal command set. In *Modern Vim*, I use a specific notation to illustrate Vim usage, which I outline here.

Playing Melodies

In Normal mode, commands are composed by typing one or more keystrokes in sequence. These commands appear as follows:

Notation	Meaning
`x`	Press `x` once
`dw`	In sequence, press `d`, then `w`
`dap`	In sequence, press `d`, `a`, then `p`

Most of these sequences involve two or three keystrokes, but some are longer. Deciphering the meaning of Vim's Normal mode command sequences can be challenging, but you'll get better at it with practice.

4. https://github.com/BurntSushi/ripgrep
5. https://github.com/BurntSushi/ripgrep/releases

Playing Chords

When you see a keystroke such as `<C-p>`, it doesn't mean "Press `<`, then `C`, then `-`, and so on." The `<C-p>` notation is equivalent to `Ctrl`-`p`, which means "While holding down the `Ctrl` key, press the `p` key."

I didn't choose this notation without good reason. Vim's documentation uses it (:help key-notation), and we can also use it in defining custom key mappings. Some of Vim's commands are formed by combining chords and keystrokes in sequence, and this notation handles them well. Consider these examples:

Notation	Meaning
`<C-n>`	While holding `Ctrl` press `n`
`g<C-]>`	Press `g`, then while holding `Ctrl` press `]`
`<C-r>0`	While holding `Ctrl` press `r`, then release `Ctrl` and press `0`
`<C-w><C-=>`	While holding `Ctrl` press `w` then `=`

Placeholders

Many of Vim's commands require two or more keystrokes to be entered in sequence. Some commands must be followed by a particular kind of keystroke, while other commands can be followed by any key on the keyboard. I use curly braces to denote the set of valid keystrokes that can follow a command. Here are some examples:

Notation	Meaning
`f{char}`	Press `f`, followed by any other character
`` `{a-z} ``	Press `` ` ``, followed by any lowercase letter
`m{a-zA-Z}`	Press `m`, followed by any lowercase or uppercase letter
`d{motion}`	Press `d`, followed by any motion command
`<C-r>{register}`	While holding `Ctrl` press `r`, then release `Ctrl` and press the address of a register
`<C-v>{nondigit}`	While holding `Ctrl` press `v`, then release `Ctrl` and press any nondigit key

Showing Special Keys

Some keys are called by name. This table shows a selection of them:

Notation	Meaning
<Esc>	Press the Escape key
<CR>	Press the carriage return key (also known as <Enter>)
<Tab>	Press the Tab key
<S-Tab>	While holding Shift press <Tab>
<M-j>	While holding Meta press j
<Up>	Press the up arrow key
<Down>	Press the down arrow key
<Space>	Press the space bar
<Leader>g	In sequence, press <Leader> then g

Note that the Meta key goes by other names such as Alt and Option.

The Leader Key

The <Leader> key can be customized to suit your preference. The default <Leader> key is \, but lots of people prefer to set it to the , key. You can set the leader key by putting this in your vimrc file:

```
let mapleader = ','
```

When you see the <Leader>g notation, you can translate the meaning to ,g, or \g, or whatever is appropriate for your configuration.

Interacting with the Command Line

In some tips you'll execute a command line, either in a shell or from inside Vim. For example, you might be instructed to change to a directory from the provided source code examples, before opening a particular file. The $ prompt in these examples indicates that the commands are to be run in an external shell:

```
⇛ $ cd code/terminal/
⇛ $ nvim readme.md
```

Inside of Vim, pressing the : key switches from Normal mode to Command-Line mode. In this mode, you can type out Ex commands such as :write and :quit, using the <CR> key to execute the command. In the following examples, the : prompt indicates that the commands are to be executed using Vim's Command-Line mode:

```
⇛ :s/cool/awesome/g
⇛ :write
```

Any time you see an Ex command listed inline, such as :write, assume that the <CR> key is pressed to execute the command. Nothing happens otherwise, so consider <CR> to be implicit.

In Neovim, you can run a shell inside of a terminal buffer using the :terminal command. (This is covered in detail in Chapter 5, *Neovim's Built-In Terminal Emulator*, on page ?.) In the following examples, the » prompt indicates that the commands are to be executed in a shell within a terminal buffer:

```
⇛ » cat readme.md
⇛ » top
```

This table summarizes the meaning of these different prompts:

Prompt	Meaning
:	Use Command-Line mode to execute an Ex command
$	Enter the command line in an external shell
»	Enter the command line in an internal shell (within a terminal buffer)

Minimal Configuration

To follow the examples in this book, you'll need to make sure that 'nocompatible' is set and that filetype detection is enabled. Prior to version 8 of Vim, you had to specify these settings in your vimrc file:

```
set nocompatible
filetype plugin indent on
```

With the release of Vim 8, these are now default settings (:help defaults.vim). That means you don't have to include those lines in your vimrc, unless you want to keep your configuration backward compatible with older versions of Vim. You can check that filetype detection is enabled by running:

```
⇛ :filetype
❮ filetype detection:ON  plugin:ON  indent:ON
```

Make sure that you can see detection:ON, otherwise you'll have trouble following some of the tips in this book.

Using Factory Settings

Some of the tips in *Modern Vim* are written on the assumption that you're running Vim with the "factory settings." If you want to follow the steps in these tips, you can do so by temporarily moving your Vim configuration to a

location where it will be ignored when you start up your editor. For example, you could rename your Vim 8 configuration files like this:

```
⇛ $ mv ~/.vim ~/.xvim
⇛ $ mv ~/.vimrc ~/.xvimrc
⇛ $ mkdir ~/.vim
```

After following the tip, you can restore your Vim configuration by moving the files back to their original locations:

```
⇛ $ rm -r ~/.vim
⇛ $ mv ~/.xvim ~/.vim
⇛ $ mv ~/.xvimrc ~/.vimrc
```

For Neovim, you could switch to the factory settings by running:

```
⇛ $ mv ~/.config/nvim ~/.config/xnvim
⇛ $ mkdir ~/.config/nvim
```

Then you could switch back again by running:

```
⇛ $ rm -r ~/.config/nvim
⇛ $ mv ~/.config/xnvim ~/.config/nvim
```

Downloading the Examples

The examples in *Modern Vim* usually begin by showing the contents of a file *before* we change it. These code listings will include a file path that will look similar to the following:

green-bottles.txt

```
10 green bottles hanging on the wall.
```

Each time you see a file listed with its file path in this manner, it means you can download the example. I recommend that you open the file in Vim and try out the exercises for yourself. It's the best way to learn!

To follow along, download the examples and source code[6] from the *Modern Vim* book page at The Pragmatic Bookshelf,[7] which is where you will also find a place to post any errata. If you're reading on an electronic device that's connected to the Internet, you can also fetch each file one by one by clicking the filename. Try it with the previous example.

Now, let's get started!

6. https://pragprog.com/titles/modvim/source_code
7. https://pragprog.com/titles/modvim