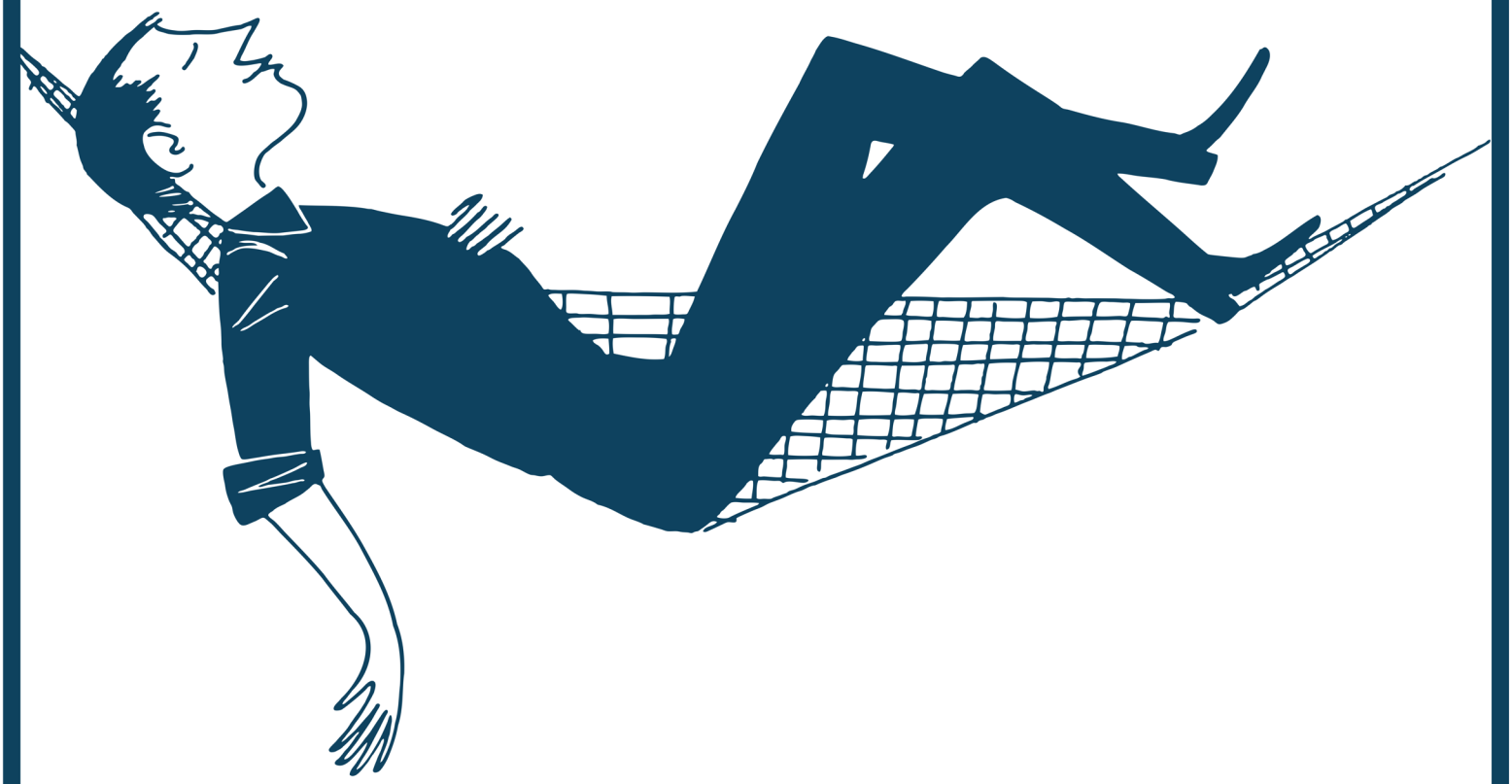
Painless Vim

A Sane Person's Guide to the World's Most Powerful Editor



by Nate Dickson

Painless Vim

A Sane Person's Guide to the World's Most Powerful Editor

Nate Dickson

This book is for sale at http://leanpub.com/painless_vim

This version was published on 2018-07-30

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Nate Dickson by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I'm taking the sane route to learning vim with Painless Vim

The suggested hashtag for this book is #painlessvim.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#painlessvim

Also By Nate Dickson

Painless Tmux

Painless Git

Contents

Preface

Why? Why This?

Look, we've all been through this: someone says, "You should learn vim. I did, and now I'm 10000% faster at everything and people like me more." So you download the latest version for your OS and start it up and, hey look, there's that screen with a bunch of tildes down the side asking you to donate to kids in Uganda. Just like the last time you tried this. You know from experience that typing `:help` isn't going to be super helpful, because you'll get stuck in the help window and you'll never be able to get it to go away. You start typing and the terminal beeps at you or says it's in some weird mode now or –if you're lucky– starts putting letters on the screen at some point. Then it's a quick trip to Google to figure out how to get back *out* of Vim, and you shudder, vowing not to go back.

It doesn't have to be this way.

At its heart, Vim is about making life easier for developers. But most of the "beginners" advice you get sounds like there's a big trial by fire initiation that you have to pass before you can get to that programmer's nirvana. People tell you that the "only way" you'll ever get good at using the power of vim is to throw your mouse out the window, disable all the hotkeys you're used to, and just spend a few months doing nothing but learning how to use an editor.

Bless their pointy little heads. It's true that once you're *good* at vim you will start to see all the system shortcuts and mouse shortcuts as inefficient. But when you're starting out they make it feel like you're suddenly in a foreign country where you don't speak the language. They mean well, but that sort of initiation is exactly wrong for a newcomer. It's advice like that that is impeding vim, keeping it from being *the* editor that everyone uses[1] .

I'm a Stranger Here, Myself

The fact is, you *can* learn vim at your own pace and be fairly productive with it right out of the box. And that's what this book is about. If vim were a foreign country, I would be the guy who walks up to you in the hotel lobby and says, "Hey, how's it going? Just get here? I'm a stranger here myself." And then I would show you all the good restaurants and stuff, because I understand what you're going through.

So instead of lecturing you on learning all the underlying `ex` and `vi` commands that you "have to know to *truly* experience the *raw power* of vim" I'm going to show you how to get in, get comfortable, and start branching out into bigger and better things. I'm going to take it as read that you're not

[1] You and I aren't the only ones that feel this way. Even very smart people can get caught by the "throw the newbie in the deep end" problem.

100% committed to leaving all other text editors behind. Heck, I'd strongly suggest having a copy of Sublime Text (more on that later) available for the times where you're just sick to the eye teeth of vim. It happens! But come back to vim, because I plan to show you that vim isn't that hard, and that learning it can be *painless*.

Introduction

Why Learn Vim

Vim, for all its eccentricities, is designed to be *fast* and *developer friendly*. Smart people have been using and improving it for decades and it's gotten to the point that it's pretty darn good in the right hands. As a former vim critic who has started using it, I can tell you that it is *that* fast when you get going. You can make a surprising number of changes quickly and efficiently in vim, and once you start navigating, making changes, and issuing commands using just the home row keys using anything else will feel clunky by comparison.

It is *that* fast

Vim, used well, is a speed *demon*. Once you get used to the slightly weird controls every other text editor will feel unwieldy. Vim was designed to keep your hands where they belong, without using the mouse or the arrow keys.

Now, like I said before, if you want to use the mouse and arrow keys while you get used to vim, go right ahead! You're not being productive if all you're doing is trying to remember how to edit the third character in the line below your cursor. But the *promise* is that eventually you'll get comfortable with the vim way of doing things and be much happier. There are other reasons, too:

Vim *commands* get around

A lot of unix-y things use the vim keybindings for moving around. Even web services like gmail use the `h j k l` magic keys to move between emails. If you get comfortable with vim idioms you'll be more comfortable with a lot of other tools as well.

Vim *itself* gets around

Let's face it, vim is *everywhere*. If you install git on windows vim comes along for the ride. If you have a Linux box or a Mac you've already got vim. And if you're like me[2] it's always slightly bugged you that there was this editor on your machine that you don't like or know how to use. If you get comfortable with vim then you don't have to panic when you have to enter a commit message from the command line in git.

[2] A few months ago

What This Book Is

This book aims to be:

Short

You've got better things to do than read books about editors. The main goal of this book is to get you comfortable with the editor and feeling like you're ready to move on, bending vim to your wishes. If you're like me, you get annoyed at making changes someone else forced on you after a while, and you just want to strike out and do it yourself. I'll let you do just that.

Cheap

Vim is free. While I don't want to sell my work for nothing, I'm coming pretty darn close. This is a beginner's book, and my goal is to lower the bar to entry. To start with, I'll price it low and hope that helps you along.

Easy to understand

Vim is complex; this book shouldn't be. My main goal here is to help you build confidence and get your feet under you. If you're reading this during the beta period and you've got suggestions, please send them on over to me! I'm more than willing to listen and change.

Friendly!

My high horse died a while back, and I never bothered to buy myself a soapbox, so I won't be doing any preaching. Remember that friendly face in the hotel who shows you all the great spots in town? That's this book. I'm not here to judge.

Opinionated (but in a good way)

I'll keep the tone friendly, but I got *opinions* people. There are things I think are good and things I think aren't so good. But I'll let you know *why* I think some things are good or not so good and let you make up your mind on your own.

Lighthearted, but serious

The *tone* of the book is meant to be light and approachable, but the *advice* is meant to be rock solid. I'm not going to have you do things to your configuration files that I wouldn't do, even temporarily, or to prove a point. The goal is to make it easy for you to get things *right* [3] the first time.

[3] According to me, see the whole "Opinionated" thing up above.

What This Book Isn't

This book definitely is *not*:

The end-all-be-all reference on vim

I'm here to help you get started, not to be the only book you'll ever buy on the subject. The main reason for this isn't laziness, thank you very much. Well, okay, it kind of is. There are a *ton* of books, websites, wikis, and other documentation out there that will help you along, once you get over the initial learning curve. I'm here to help you over that first curve.

A list of tips

A lot of the stuff in the later chapters won't make a ton of sense if you haven't been through the earlier chapters. The chapters are laid out iteratively, starting with small changes that build up to a final configuration that should work well. Then you'll have a good jumping-off point for your further modifications.

A deep dive

The goal here is to give you enough information to be comfortable, but not all the underlying technologies and line editor commands and other such things. Again, these are available online and in many other reference guides all over the place. When you decide that you want to know more about something you'll be able to find it. I have faith in you.

Who am I?

I'm Nate Dickson, it says so right on the cover.

But beyond that, I'm a professional programmer by day, blogger, writer, and daddy by night. You can check out a fairly large amount of my writing on my personal blog[4], or my less-maintained Apple blog[5]. You can also check me out on Twitter[6], or Github[7].

In terms of my vim credentials, I don't have many. I use vim in my day job and I'm a command line junky, with the `.zshrc` and `.tmux.conf` files to prove it.

But my quasi newbie status is part of why I'm writing this book right now. I've spent the last few months combing blogs and websites, buying or borrowing books on vim and grabbing onto the arm of anyone I know who uses vim and begging them to help me fix whatever mess I've most recently gotten myself into. Now that I'm fairly comfortable, I'd like to pass what I've learned along to ease the learning curve for everyone else.

[4] http://natedickson.com
[5] http://crazyapplenews.com
[6] @poginate
[7] http://github.com/PogiNate

References

I will refer you to a lot of other books, posters, blogs, wikis, etc. etc. over the course of this book. For the most part these are things I've used extensively myself, and can heartily recommend. Again, I'll try to provide reasons for my references and let you decide if it sounds like something you'd like to investigate on your own.

Sample Documents

Throughout the book there will be exercises to try out. For convenience, I have created a repository of sample documents for use with these exercises. They are stored on Github[8], and the readme file has instructions for how to get them in place, if you're not familiar with git and Github.

The Obligatory "Conventions Used in This Book" Section

First off, regarding the actual word "vim". In general I will write it all lowercase, because that's how most people think of it, like "perl" or "ruby". That's the command you enter to start the editor, and that's really the name of the program.

However, when it's in a book, chapter, or section title I will capitalize it, just as I would any other word in a title: *Painless Vim*

Ironically, even though it's probably the most correct, I will never write it as "VIM", even though it is a acronym (kinda) for "Vi IMproved"

Other than that, I use the same conventions everyone else uses. Code appears in `a monospaced font`, and longer blocks of code will appear

```
in a large block
of monospaced text.
```

Special keys like `<shift>`, `<enter>`, `<esc>`, will have angle brackets around the name to make it real obvious that you shouldn't be typing that word, you should be pressing that key. If you need to press `<ctrl>` and another key at the same time (and you *will* need to do this, quite frequently) I'll just write them like so: `<ctrl>d`.

Vim uses both upper- and lower-case characters, so remember that "`S`" is `<shift>s` and "s" is just `s`.

[8] https://github.com/PogiNate/Painless-Vim-Samples

Those are the only two times you should ever need to press two keys at the same time: either to hit `<ctrl>` and a character, or to press `<shift>` and a character to make something upper-case.

Which is good, because there will be a lot of times where you have to hit multiple characters in sequence. For example, let's say you need to delete the next character after your cursor. You would press `d` (for delete) and then `l` (for the next character. Don't worry about it now; we'll get into it later). When I write this out in the book I will just write the two letters in a row, like so :

```
dl
```

I realize that looks exactly the same as my "type them at the same time" notation. Just remember that you'll only ever need to press `<ctrl>` or `<shift>` at the same time as another character. In all other situations, two characters in sequence means type them one after the other.

Okay, got it? It's pretty natural, I promise. Let's move on.

From time to time I will need to show you *exactly* where your cursor needs to be when you perform a command. In these cases I will put a pipe symbol where the cursor resides, like so:

```
The cursor is in the middle of the word pi|pe
```

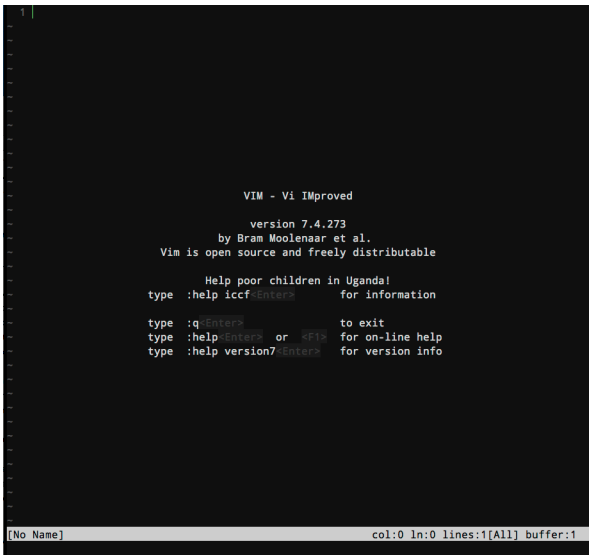
Again, this will all make more sense in a bit. For now let's get moving.

1. Get In, Get Out, Get Comfortable

Okay, you've gotten through the introduction. Good for you. I'd like to say that the style will settle down at this point, but I can't promise that.

Here's where we get started on the real meat of learning vim. We're going to start simple: *open the editor*. For now I'd recommend doing things from the command line, not because I have some big problem with GUI's, but because the graphical versions of vim tend to make little changes that can get in the way of the things we're trying to do here. So, open a Terminal [1] and enter the following:

```
1  $ vim
```



I have been here before. I don't like it.

There! That's not so scary, is it? Just a bunch of tildes and some version information! Not scary at all!

Well, okay, yeah it is. You're in this editor with no discernible way out. So let's cut straight to the quick and talk about how to get back out. For now we're going to use the absolute "oh crap let me out" method. Type `:q!<enter>`[2] and you should be right back to the command prompt, which looks like this:

[1] I'm going to use lowest common denominator words like "Terminal" a lot. If your OS calls it something different like a shell or a command line or (heaven help you) `cmd.exe` then just substitute that in your head.

[2] Later on you'll want to know that this is the "quit and don't save anything" way of getting out. There are a few other ways to exit vim, but we'll get there later.

```
$
```

Okay, breathe. This time we're going to go in armed a little better. Before you dive back in, find a file you've been working on lately. HTML files work well, but pretty much anything that is structured text or code will work just fine. If you don't have anything that springs easily to mind you can use `sample.html` from my sample repository. Switch to the directory where that file is kept:

```
cd /path/to/where/the/file/lives
```

And do this:

```
vim file1.html
```

And you should see…a boring representation of your file. But we're going to fix that right now. Enter the following:

```
:syntax on<enter>
```

And the file should brighten up a bit. We'll talk more about colorful files in a bit, but for now let's get back out of vim, because it's still scary:

```
:q!<enter>
```

Okay, you're safe. Let's take a minute out here in the real world to formalize what we've just learned:

1. When you start vim, the keyboard makes things happen instead of making letters appear.
2. When you want to send a command to vim, you preface it with a colon (`:`).
3. `:q!<enter>` gets you back out of vim without saving any changes.

Cool.

Line numbers and Moving Around

All right, now that we've had a breather, let's dive back into vim. Open your test file again with

```
$ vim file1.html
```

And you'll see you're right back to boring plain ol' text. You can tell vim to color it again with

```
:syntax on<enter>
```

But it seems like a bummer to have to do this by hand every time you start vim. You're right. Don't worry, we'll fix that.

But there's more. We've learned that when you send commands to vim that start with a `:` you have to press `<enter>` at the end. This is useful information, because I'm going to stop putting `<enter>` after every command from here on out. Remember that if you start a vim command with a colon it always ends with an `<enter>`. There are other commands that *don't* need an `<enter>` at the end. We'll get to them in time. For now let's look around this file.

2. Moving Around in Vim: Baby Steps

Now that you're in a file, it's time to move around a little bit. If you've ever played with vim before, you've probably seen something that looks like this:

```
        k
h               l
        j
```

This is a super-unintuitive way of telling you which keys will move your cursor which directions. In English it goes like this:

- `h` moves your cursor one space **left**
- `l` moves your cursor one space **right**
- `k` moves your cursor one line **up**
- `j` moves your cursor one line **down**

So play with those four keys for a while. The layout is weird at first, I'll grant you, but it serves the purpose of keeping your hand firmly on home row, and after a while you'll start to just believe that `h j k l` should *always* move your cursor around. This is great when you're using a program that understands vim keys. It also means that you'll end up in an instant message conversation with someone, and you'll inadvertently send them a message that says "jjjkl" as you try to figure out why you can't move your cursor. It's a curse.

Okay, moving on. Once you've gotten comfortable moving your cursor around, it's time to *edit* something. Use the movement keys to get your cursor where you want it and press `i`.

At the bottom of the screen you should see a message appear:

```
--INSERT--
```

This is vim's way of telling you that pressing keys now will make things change in your text. In vim parlance, you have just entered *insert mode*.

Type something witty in the document. I would put "So long, and thanks for all the fish" because it's better than "hello world". But I know it doesn't matter what I tell you to enter; you'll put whatever you want. That's just how you are. You're independent and I respect that.

Once your witticism is entered it's time to save it up. Hit `<esc>` and the `--INSERT--` message at the bottom of the screen should disappear. This means that the keyboard is back to issuing commands to vim instead of changing the text on the screen. Vim calls this *normal mode*, but don't worry too much about that yet. Type

```
:w
```

And your changes are *saved*, friend.[1]

Now let's do something a little more daring and, dare I say it, *useful*. Moving around with your fingers on home row is nice and all, but it's not worth all this hassle and bother. Let's look at some slightly larger jumps in the text.

Position your cursor somewhere on a line of text–anywhere is fine– and make sure the `-- INSERT --` thingy is gone by pressing `<esc>` a bunch of times until vim starts beeping and/or flashing at you.[2] Now press `A`. Note that was an uppercase `A`, not a lowercase `a`. This handy little shortcut moves you to the end of the line and puts you back into insert mode, all at once. Think of it as "append" or "add". Nifty, huh? Okay, press `<esc>` again and let's do another one. Press `o`. (Again, note the case.) Vim adds an empty line, right under the line you were on, and once again you're in Insert mode, ready to enter new text into the file. Write something if you want, and then press `<esc>` again to get out of Insert mode.

What Are All These Modes You Keep Mentioning?

I'm getting there, little by little. If you want to skip ahead to the chapter on Modes feel free, but the path I'm taking will be a little nicer. For now you only need to know that when vim says

```
 -- INSERT --
```

you're in *insert mode* (so called because that's how you insert text) and when it doesn't you're (probably) in *normal mode*. There are other modes, and a lot more to say on the topic, but right now we're getting used to moving around, and we don't want to be bothered.

We're going to talk more about moving the cursor around later, but for now look how awesome you already are! Since you don't need to worry about changes this time, you can just type

```
:q!<enter>
```

To exit vim.

[1] Remember that commands that start with a colon `:` require you to press `<enter>` to make them execute. Okay, this is definitely the *last* time I'm going to repeat that.

[2] I've heard it said that some people call Normal Mode "Beep Mode" because you'll try to switch back to it and the terminal will beep at you and you'll realize you're already in it. I don't know if I believe that anybody actually calls it that, but since I'm writing a vim book I'm honor bound to pass this bit of information on to you. Just like if I were writing *Beginner's Big Book o' Computer Facts* I would feel obligated to mention that four bits is called a "nibble" (because it's *half a byte* and "byte" sounds like "bite"! Get it? *GET IT???*), even though I'm pretty sure that's just a joke that got put in some textbook years ago and has since gotten out of hand.

More Ways Out

Vim is a strong believer in the "TIMTOWTDI" [3] principle. There are five main ways to get *out* of vim:

- `:q` Quits. This will fail if you have unsaved changes in your file.
- `:q!` Force quits, even if you have unsaved changes.
- `:wq` Writes (saves) and quits.
- `ZZ` Same as `:wq`, but much easier to type.
- `ZQ` Same as `:q!`, but *way* easier to type. [4]

I would strongly recommend getting used to the `ZZ` and `ZQ` versions, since they are the easiest to type and remember. But hey, it's up to you.

Zoom!

Let's go back into vim and play with three simple (but useful) commands that I call the *zoom* commands. To enter vim this time type

```
1 $ vim sample1.html
```

So that we've got a document to play with. Press `j` repeatedly to move your cursor down a few lines, it doesn't matter where it ends up. What we're going to do is tell vim to scroll the window so that the line the cursor is on is positioned where we want it.

The first "zoom" command is `zt`, which (I say) stands for "zoom top." When you press `zt` in command mode it will move your current line to (you guessed it) the top of the screen. So, if your cursor is on line 15 and is one third of the way down the screen it will now be at the top of the screen.

Next up is `zb`, or "zoom bottom." As you can guess, it moves your current line to the bottom of the screen, letting you see everything that came before.

The third and arguably most useful zoom command is `zz`, which positions the line your cursor is on in the *middle* of the screen, so that you can see that line in context. I use all three commands fairly frequently, but I find this is the one I like the best. It's my secret favorite child.

All right, we've done some good work here. You've started vim and gotten around a bit, and more importantly, you made your code all pretty. In the next chapter we'll get down to the single most important thing you ever have to do with an editor: choose your color scheme.

[3] "There is more than one way to do it" usually pronounced "Tim-Toady"

[4] Here are two secret bonus ways to exit vim that you won't need to know about until you know how to open multiple files: `:qa` closes all open files and quits vim, and `:qa!` does the same thing even if any file has unsaved changes. Don't worry, we'll review them again when we talk about opening multiple files.