Vim for humans

Release 1.0

Vincent Jousse

Aug 21, 2017

CONTENTS

CHAPTER

ONE

PREAMBLE

1.1 Disclaimer

I'm not a native english speaker, so if you spot errors, feel free to tell me. Here is my email address: vincent@jousse.org

1.2 Paid book but free price

What does it mean? I am convinced that culture and education should be accessible for everyone. Asking people to pay a price fixed in advance would put a barrier that limits the access to the book. That's why I let you choose your price.

And what if you don't have any euro to give in exchange for this book? Well, just download it freely and make the promise you will use it for the best. But don't tell me you have nothing to offer in exchange. An email with a thank-you would be just fine.

Whatever you give, it will be a motivation for me to write more books like this one.

If you give in euros, 20% of the earnings are donated to Framasoft, an association promoting free software.

You can give on the official website http://vimebook.com/en

Thanks for listening, and have fun!

1.3 Free license

I'm releasing the book under the Creative Commons Attribution 4.0 Internationa lLicence. So, you can do (almost) whatever you want with this ebook.

1.4 Thanks

A big thank to my lovely sister Sandra for her reviews!

CHAPTER

TWO

INTRODUCTION

When you need to write or to code, you have to choose a text editor, and a very good one. They are many text editors available out there, but very few of them are more than 40 years old. It's the case of *Emacs* (http://www.gnu.org/software/emacs/), *Vi*, and its improved successor *Vim* (http://www.vim.org). They were created in the 70's and are still used a lot nowadays. You may have already noticed that it's not thanks to the beauty of their website or the efficiency of their communication. Here are some **reasons for their success**:

Forever You learn them once and you use them forever. In a world where languages and technologies are constantly changing, learning vim is a real chance to invest in a skill you'll be able to use forever.

Everywhere They are available for each and every possible platform : Mac Os X, Windows, GNU/Linux, BSD, and so on, and it's always been that way.

Efficient Thanks to their features (like the extensive use of the keyboard), you can edit and write text as fast as your thoughts.

For everything They allow you to edit everything and anything. When you'll use another programming language, or another markup language, you won't have to change your editor. Of course, this book has been written using *Vim* (and the ReST Markup).

Yet, these text editors are difficult to learn. Not that they are harder than anything else, not that you can't handle it, but rather because there is no smart way out there to learn them for now. So, here we are.

The aim of this book is to address this gap by guiding you through your discovery of *Vim*. I'll put *Emacs* aside from now and I'll focus on *Vim*. If you want to know more about this **Editor war**, be sure to check the Wikipedia page. This book doesn't claim to be a reference book about *Vim*. There are already a lot of good references on the subject like A byte of Vim. However, it claims to reduce the entry barrier to get used to *Vim*. In my opinion, the most difficult thing about learning *Vim* is not getting discouraged while finding a way to use it, learning *Vim* step by step. We all have to get things done with our text editor on a daily basis, that's why losing all your productivity when switching to *Vim* is not an option.

I'm sure you'll find a lot of people who will tell you: "Just do it cold turkey", "You'll see, it's hard at the beginning, but time will help". True, but you'll still have the problem of trying to remain productive on a daily basis. The approach of this book is the following:

- Have a modern *Vim*: syntax highlighting and nice colors.
- Use *Vim* as any other text editor: easily edit code and switch between files using the mouse.
- Learn keyboard shortcuts and go without the mouse step by step.
- Install the *best* plugins to start using *Vim* to its full potential.

Starting from bullet number 2, you'll already be able to use *Vim* on a daily basis without losing a lot of productivity. It's where the magic will happen: if you can integrate *Vim* in your daily habits, you have won. You'll then have the rest of your life to learn all the shortcuts and the tip and tricks of *Vim*.

You're tired of trying a new editor each year? You're tired of having to relearn everything from scratch every time? You're tired having to change your editor when you're using your Mac, Windows or Linux? So, just stop it, and join the community of people happy with their text editor!

2.1 For who?

Every person having to produce text (code, book, reports, slideshows, . . .) regularly. Developers are of course concerned, but it's not only about them.

For example, if you are a:

Student If you want to impress your future boss with your resume, it's a must. It's a proof of seriousness to see that a student took the time to learn *Vim* on his or her own. Moreover, you'll have a unique tool to write all what you'll have to write (and that you'll be able to use for the rest of your career): your LaTeX reports, your slideshows, your code (if you need Word or LibreOffice to write you reports, it's time to use LaTeX, Markdown or reStructuredText).

Friendly advice: for your slideshows, don't hesitate to use something like impress.js. It's using HTML/JS/CSS and I highly recommend that you use it to do awesome presentations based on non-proprietary technologies. You can have a look at reveal.js too, and its online editor slide.es.

Teacher It's time to set an example for your students and to teach them a tool they will use during their entire life. *Vim* is something they'll be able to use a lot more than any programming language.

Coder It's essential to invest time in your daily tool. You'll be learning keyboard shortcuts anyway, so you should do it for something useful. If this investment is still profitable 10 years from now, it's the perfect investment. If you put time into learning *Vim*, you will be able to use it for many decades.

System and network administrator If you use *Emacs*, then I can forgive you. If you use nano/pico, there is nothing I can do for you. Otherwise, it's time to get some work done, folks! Remote administration of a Unix system is the perfect use case for *Vim* (a powerful text editor without the need of a graphical interface).

Writer If you write using Markdown/reStructuredText/WikiMarkup or LaTeX, *Vim* will save you a lot of time. You'll not be able to go back to another editor after it, or you'll want to *Vimify* it at all costs.

Trust me, I have done and still do all these 5 roles, and my best investment has always been, by far, *Vim*.

2.2 What you will be learning

- How to use *Vim* as a "usual" editor first (you know, the type of text editors having syntax highlighting, allowing you to open files, to click using the mouse, . . .). In short, we will be demystifying *Vim* to allow you to go further.
- How to move from classical text editing to the power of *Vim*, baby step by baby step (it's where addiction begins).
- How to do without the mouse and why it's the best thing that can happen to you when you're programming/writing text.
- How you can easily deduce keyboard shortcuts with some simple rules.

To sum up: if you consider yourself a craftsman, act like one. Learn how to use your tool, once and for all.

2.3 What you will not be learning

- You will not be learning how to install and to configure *Vim* for Windows. It's doable, but I have very limited knowledge about Windows. It may happen, but not yet. Only Linux/Unix will be discussed (and by extension Mac OS X).
- You will not be learning how to use *Vi* (notice the lack of "*m*"). I'll only teach you how to be productive writing text with *Vim*, I won't be teachng you how to impress your friends with *Vi* (and anyway, *Vim* is enough for that). For those who don't get what I'm talking about, *Vi* is the "ancestor of *Vim* (which stands for *Vi - IMproved*)" and is installed by default on all Unix-like systems (even on Mac OS X).
- You will not be learning to know *Vim* by heart: this book is not a reference it's a pragmatic smart way to learn *Vim*.
- You will not learn how to pimp the colors of your *Vim*, although I will go over how to change your theme. I'll use the Solarized theme, it's the best theme for your eyes.

2.4 The hardest part is to get started

So, your are ready for the adventure? Ready to sacrifice one hour to start using *Vim*, one week to be familiar with it, and the rest of your life to be happy with your choice? So here we go! Well, almost, we need to talk a little bit before.

With *Vim* you'll have to struggle. No matter how big your willpower is, you will struggle. Be prepared. The goal of this guide is to diminish this struggle as much as possible, but be aware that you will struggle anyway. No pain, no gain. Here is the method I recommend to tame the beast:

- Try to make using *Vim* a habit. Be sure to follow this guide until the chapter about *The NERD Tree* (the file explorer). Then you'll be able to use *Vim* as you would do with Notepad++, Textmate or Sublime Text for example. You'll be using only 1% of the capacities of *Vim*, but whatever. What really matters is to use *Vim* on a daily basis.
- Be sure to have a printed sheet with all the main *Vim* shortcuts near you. The goal here is not to learn them by heart, but only to have somewhere to look when you'll ask yourself: "There must be a better way to do this".
- Keep the faith. At the beginning you'll be sceptical regarding the usefulness of learning everything from scratch with *Vim*. And then, one day, you'll have that "a-ha!" moment. You'll be asking yourself why all the software you're using can't be controlled using *Vim* shortcuts.
- Keep in mind that it's an investment for your next 20 years. As you know, investments are rarely profitable immediately.

So, enough talking, let's get started!

CHAPTER

THREE

HAVING A USABLE *VIM*

This may be a surprising approach for you, but for me, the first thing to do is to have a *Vim* usable by a normal human being. It seems that everybody agrees that *Vim* is a very **powerful editor**. And I think that you will agree too if I say that, by default, *Vim* is totally unusable. Let's be honest, without a decent minimal configuration, using *Vim* is **counterproductive**.

In my humble opinion, it's the first obstacle to tackle before anything else. This is what all the trendy editors like TextMate, Sublime Text, Notepad++ or Netbeans are proposing: a default environment usable as it is, even if we don't use its full potential for now.

Here is what a default *Vim* is missing (and why **most of people are giving up** before they can really see the power of *Vim*):

Default configuration You can configure *Vim* thanks to a file named `~/.vimrc`. This file is, obviously, totally empty by default. The first thing to do will be to have a `~/.vimrc` file with a minimal configuration.

Syntax highlighting By default, *Vim* is white and ugly. To fix that, we will use the Solarized theme. If your goal is to be efficient, it's the best theme available out there (across all text editors), period. The beautiful image below will give you an idea of what it looks like (with the dark and the light theme). Personally, I'm using the dark theme.

File explorer If you are using *Vim* along with a graphical interface (I suppose it's the case for 99% of you) you will by default have a `File` menu available. This menu should allow you to open a file. It is, for sure, a good start. But having a file explorer a la Netbeans or Textmate can be very handy. To mimic the same behavior, we will be using The NERD Tree. Be aware that, once you will have read this guide, you will not need the mouse anymore.

This chapter is mandatory if you have very few (or not at all) experience with *Vim*. By the end of the chapter, you will have a *Vim* usable on a daily basis. It should be enough to then be able to learn it gradually. Because, of course, there is no magic, you will have to practice to be able to learn *Vim*, and the sooner, the better.

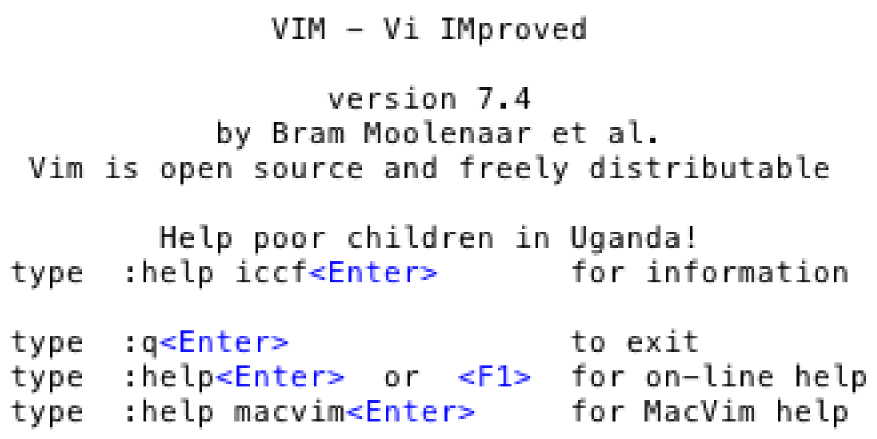
However, if you are already familiar with *Vim* and don't use the mouse anymore, you can skip this chapter. But be sure to give *Solarized* a try, as you would be missing something otherwise.

3.1 Essential preamble: the insert mode

Let's be totally crazy. We will try to create the `~/.vimrc` configuration file with *Vim* itself. As I said earlier, the sooner you start to use *Vim*, the better. I told you it would be totally crazy!

The first thing to do will certainly be to install a *Vim* version for your operating system. If you are using a Mac, give MacVim a try, it's the best *Vim* port for Mac without a doubt. If you are using GNU/Linux or any other "Unix-like" system, you should have *gVim* available, or at least easily installable using your package management system. Be sure to install the __full__ version (ie. with ruby and lua support). For ubuntu, the package is called *vim-nox*. For Mac OS X, MacVim has already all what you need builtin. For Windows, it seems that there is a version available on the official *Vim* website (http://www.vim.org/download.php), but I haven't tested it.

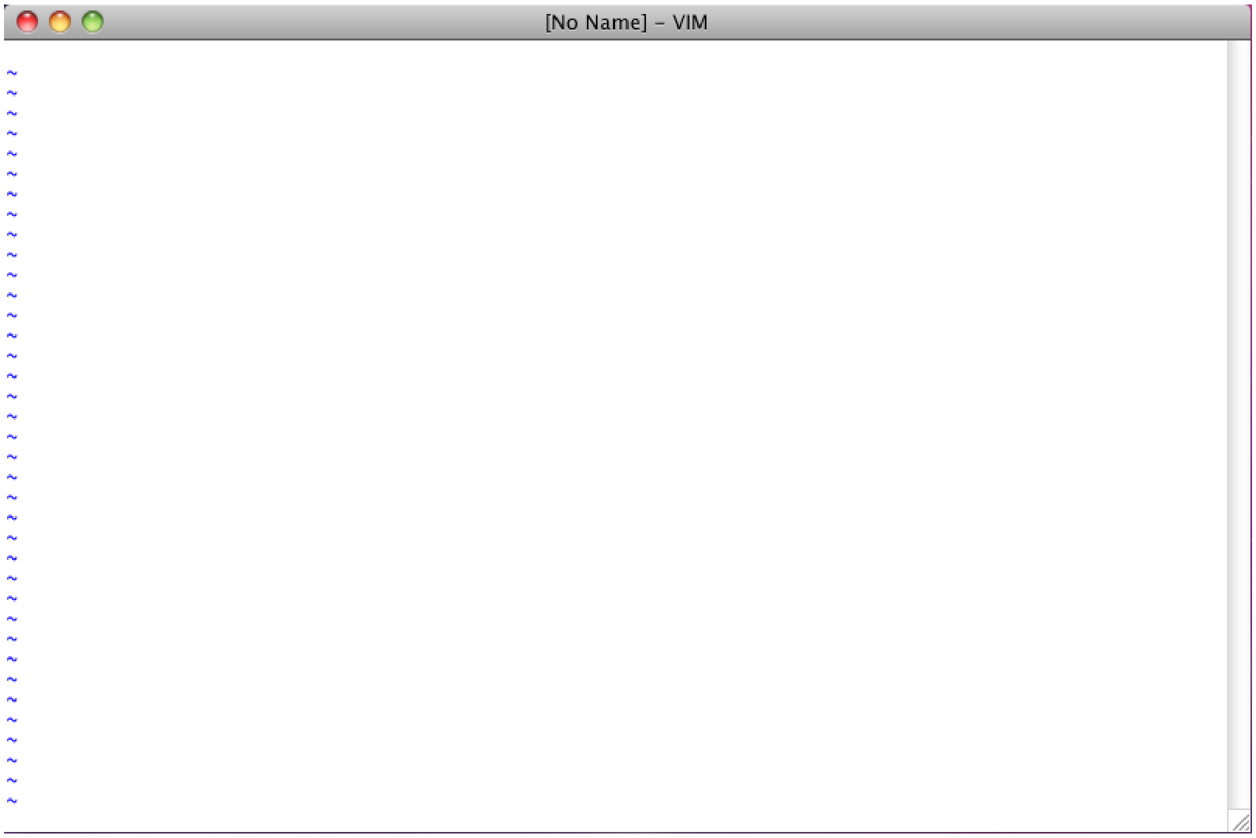
When you will start *Vim*, you should see a welcome text asking you to help poor children in Uganda (or something along the lines).



```
                VIM - Vi IMproved

                   version 7.4
             by Bram Moolenaar et al.
   Vim is open source and freely distributable

          Help poor children in Uganda!
  type  :help iccf<Enter>       for information

  type  :q<Enter>               to exit
  type  :help<Enter>  or  <F1>  for on-line help
  type  :help macvim<Enter>     for MacVim help
```

This text will disappear as soon as we start writing text in *Vim*.

We will start by adding a comment in the header of the file to specify the author of the document (this should be you). To be able to type text, the first thing to do will be to press the `i` key (the cursor should have changed). The welcome text should have disappeared and you should have *a blank page* looking like the image below:

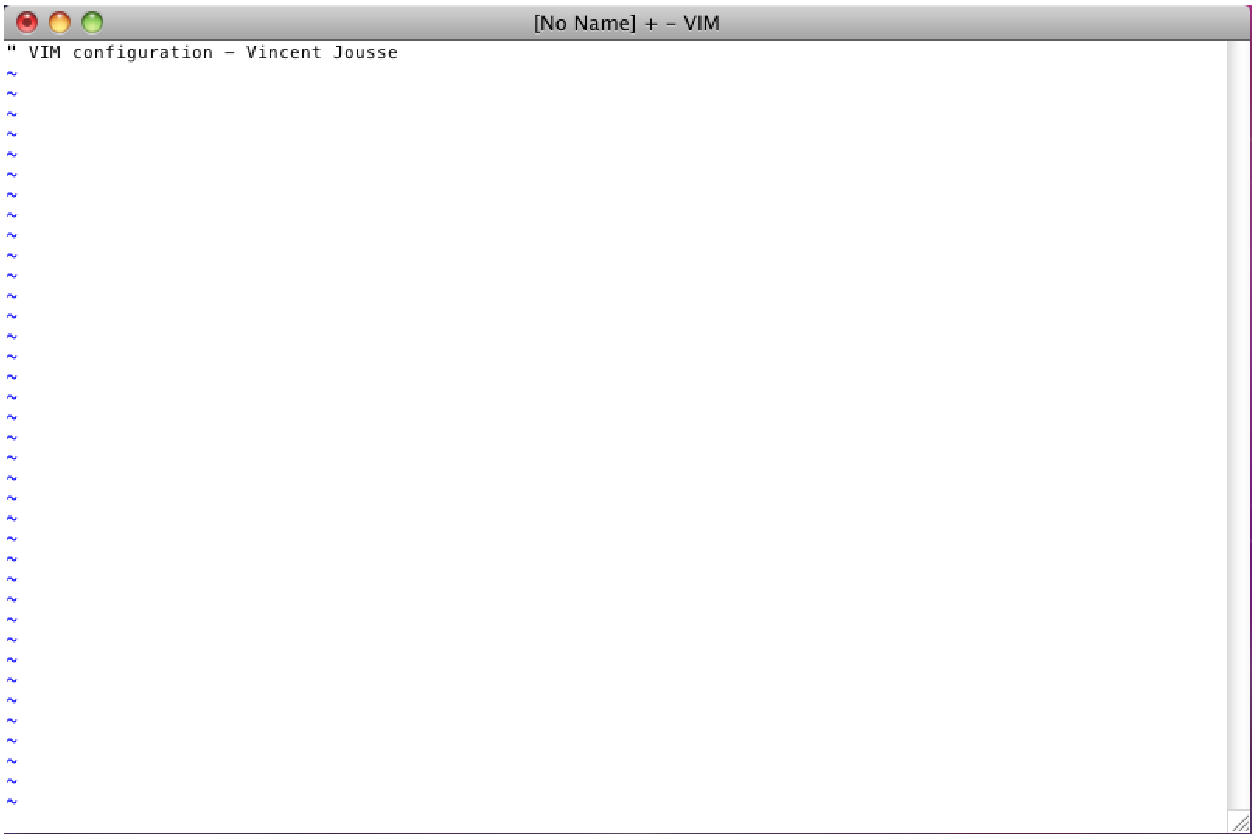


On a side note: if you don't really understand what you have done and *Vim* is displaying red messages at the bottom left or doesn't seem to react as it should when you press the `i` key, don't panic. Pressing multiples times on the `Esc` key (two times should be enough) should bring *Vim* to its default mode, the *Normal mode*. Then it should behave as you would expect again.

You should know be able to write down *the comment below*:

```
" VIM Configuration - Vincent Jousse
```

You will have noticed that comments in *VimL* (the name of the *Vim* programing language) start with a ". Then press the `Esc` key to come back to the default mode (the normal mode) of *Vim*. That's all, you are done. Here is a screenshot of what your *Vim* should look like now:



```
" VIM configuration - Vincent Jousse
~
~
~
```

I can already hear you: all that fuss for that? Well, yes. And you even don't know how to save a file. But all these things that I'm about to explain to you are logical. One of the advantages of *Vim* is that, usually, it is logical. Once you will have understood the logic behind it, all will be crystal clear for you (at least I hope so).

By default, when you start *Vim*, you are presented with its default mode. This mode is called the *Normal mode*. The purpose of this mode is not to write text (for that, you will have the *Insert mode*), but only to move the cursor and to manipulate text. The power of *Vim* is coming from the combination of these two modes (other modes exist, but it's not the topic for now). You will need some time and some practice to realize the power it's giving to you, so you will just need to trust me in the meantime.

If you are asking yourself why those modes exist, why can't we even write down some text by default, and why we are making things more complicated than they should be, the next chapter is for you.

3.2 Modes: the powerful *Vim* secrets

I suppose you will agree if I say that, if you want to learn *Vim*, it's to be more efficient when writing/manipulating text or code. To be more efficient when writing text, there are not many solutions. There is only one actually: you need to move your hands as less as you can (even not at all) and only move your fingers.

To do so, of course, you will need to do without your mouse. In addition to being slow, the move keyboard -> mouse and then mouse -> keyboard is really bad for your joints. It's often the cause of musculoskeletal disorders. Maybe you are still young and don't know what I'm talking about, but believe me, you will have such problems one day or another (often sooner than you may think). If you are in front of your computer all day long, don't neglect those possible troubles, you may regret it someday. According to Wikipedia, it's actually the most common professional disease.

You will need to forget the movement of your right hand toward the directional keys (left/right/bottom/top) too. It's a waste of time and it's completely unneeded with *Vim*.

So what do you have the ability to do? Not a lot to be honest (but it's for your own good), you can only leave your hands on the home row *as you can see on the picture below*.

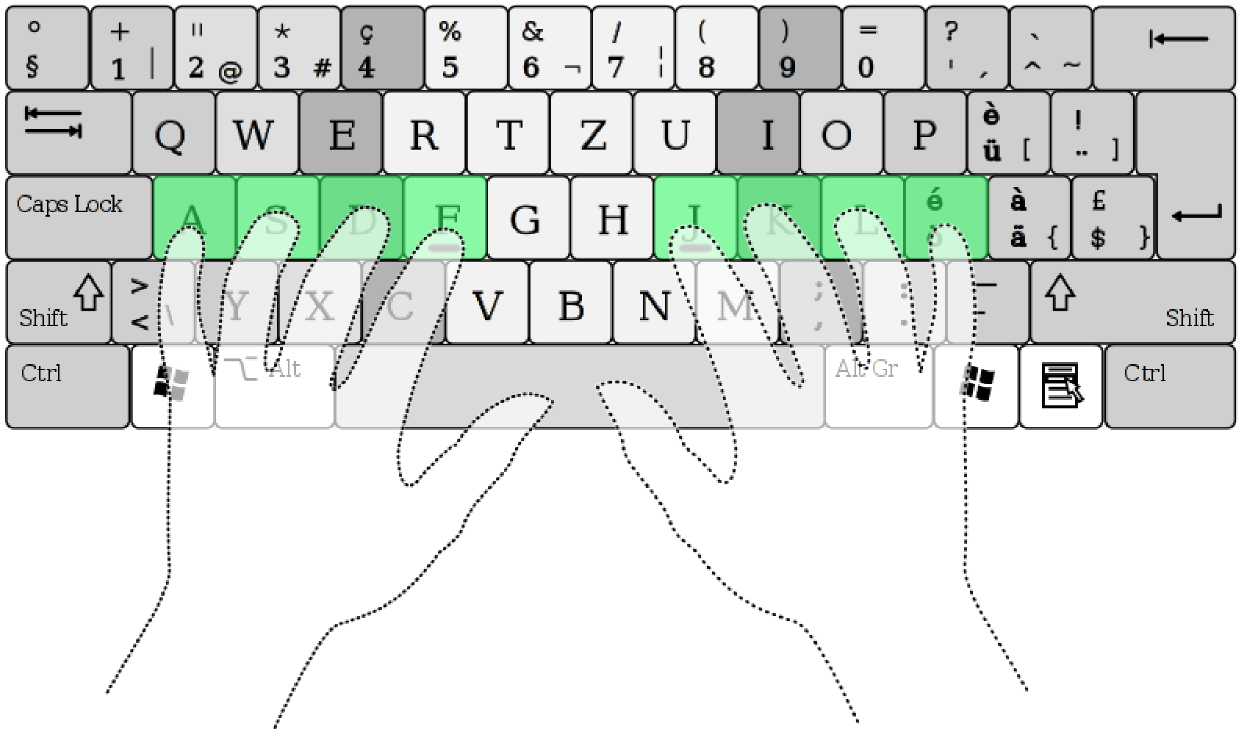


Fig. 3.1: Home row, QWERTY keyboard
Illustration by Cy21 - CC-BY-SA-3.0 (http://www.creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html), via Wikimedia Commons http://commons.wikimedia.org/wiki/File:Typing-home-keys-hand-position.svg

You will also probably find on your keyboard some marks on the letters F and J. The goal of these marks is to give a landmark for the position of your fingers (the indexes) on the home row of the keyboard.

Trying to move as less as possible the hands from the keyboard is the reason for having a *normal* mode and an *insert* mode in *Vim*. When switching from one to the other, the keys under your fingers will sometimes allow you to move the cursor and to manipulate text: copy/paste, deletion, ... (it's the normal mode), sometimes they will allow you to select some text (it's the *visual mode*) and sometimes to insert some text (it's the *insert* mode). And of course, all of that is possible without the extensive use of keys combinations like *Ctrl + key* that are very bad for your fingers (*Emacs*, this one is for you).

By default, we can switch from the *insert* mode to the *normal* mode by pressing the `Esc` key, but it will be one of the first things we will change: the `Esc` key is to far from your fingers on current keyboards.

To switch from *normal* mode to *insert* mode, we can press the `i` key. We will later learn that there are other ways to do so too. For example, to enter the *insert* mode and to then create a new line below the current one (no matter where is your cursor on the line), we will use the `o` key while in *normal* mode.

I will talk again about this subject later in "*Learning how to move: the copy/paste use case*", but if you are not ready, at some point, to do without your mouse and the directional keys to edit text, I would recommend you to stop learning *Vim* right now. It's as simple as that. You can leverage the full power of *Vim* only by getting rid of the mouse and by moving your hand as little as possible.

If you want to go further, you can buy an orthogonal keyboard like TypeMatrix. It's the keyboard I'm currently using, and my fingers are thanking me everyday.

The ultimate change would be to switch your keyboard layout to a more efficient one like Colemak, but that's another story.

3.3 The lifesaver default configuration

Let's get serious and try to have a usable *Vim*. We will start by editing the default configuration file `~/.vimrc` and by entering default values that any sane person would love to find in it.

You have to place this file in your home directory. It should be */home/your_user/.vimrc* if you are using Linux, */Users/your_user/.vimrc* if you are using Mac OS X. Generally speaking, it should be in your home directory under *~/.vimrc*. If you are using Windows, you'll need to create a file named *_vimrc* that you have to put in your *%HOME%* directory. This directory is obviously not the same across the different Windows versions. Usually, it's the directory just before your *My Documents* directory. More information is available on Wikipedia if you want.

I've directly commented all the lines in the code itself. Nothing fancy here, you should just be asking yourself why all of this is not available by default.

For those who have done a copy/paste, you just have to save your newly created file. We want to put it in our home directory, so you have to save it as *~/.vimrc*. When using Mac OS X and Linux, ~ is the home directory of the current user. But be careful, when using Linux and Mac OS X the files starting with a **.** are hidden files. Don't be surprised when you don't *~/.vimrc* in your file explorer by default.

For people using the mouse, you will just have to click the *File* menu, and then click *Save as*. Save the file under the name *.vimrc* in your home directory. For those who want to use the keyboard you will just have to type :sav ~/.vimrc (be sure to be in the *Normal* mode by pressing the `Esc` key first). To save your further modifications, just use the menus with the mouse or type :w when in *Normal* mode.

I have uploaded this configuration file directly on *Github*. You can download or copy/paste it directly from: http://vimebook.com/link/en/firstconfig.

Below is a screenshot of *Vim* (macvim) *after your first configuration*.

Notice the line numbers on the left and the position (coordinates) of the cursor at the bottom right.

Well, it's a good start, but we now need more colors. Let's go!

3.4 And now, the color!

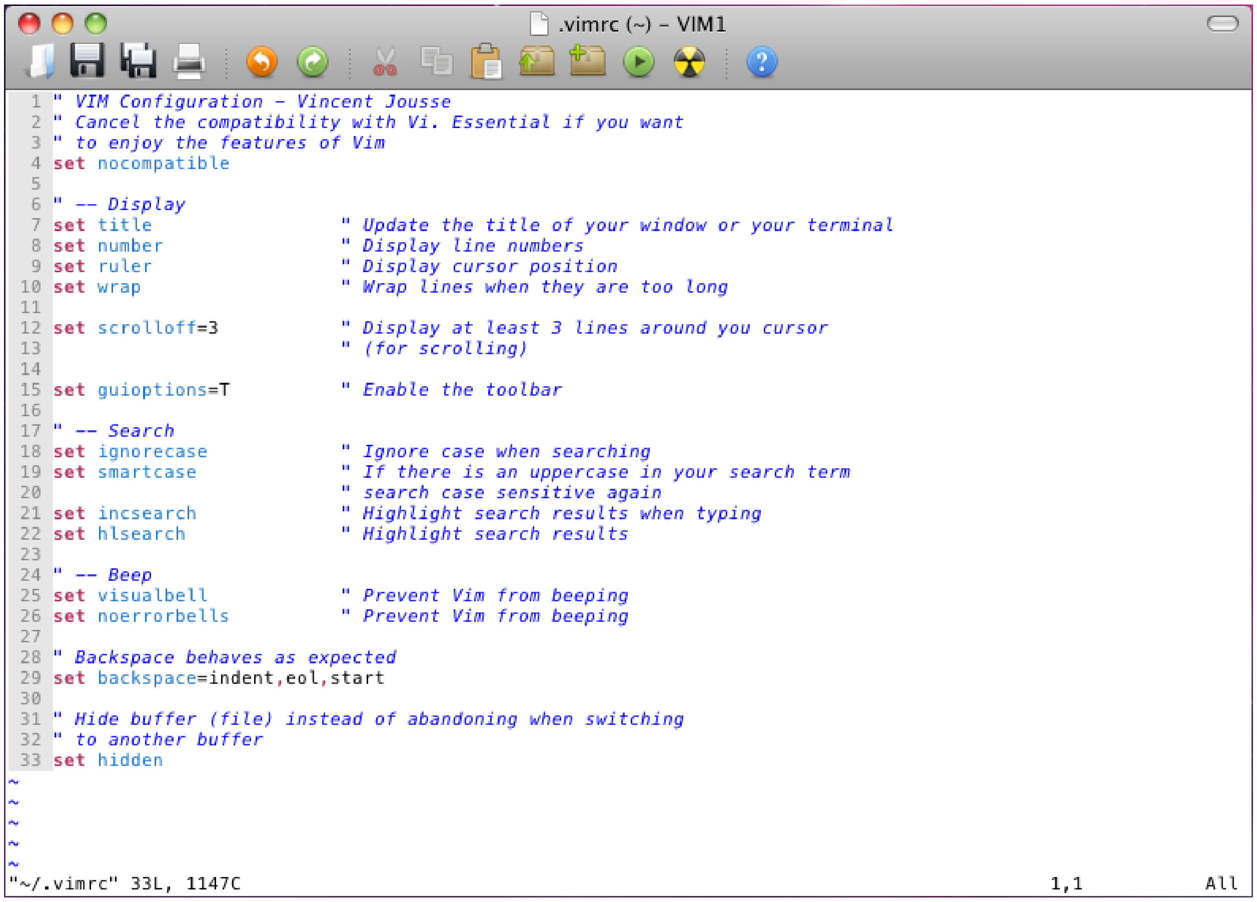
First, we need to enable syntax highlighting in the configuration file. Add these lines at the end of your `~/.vimrc` configuration file:

```
" Enable syntax highlighting
syntax enable
" Enable file specific behavior like syntax highlighting and indentation
filetype on
filetype plugin on
filetype indent on
```

You should have a *Vim* looking like the picture below.

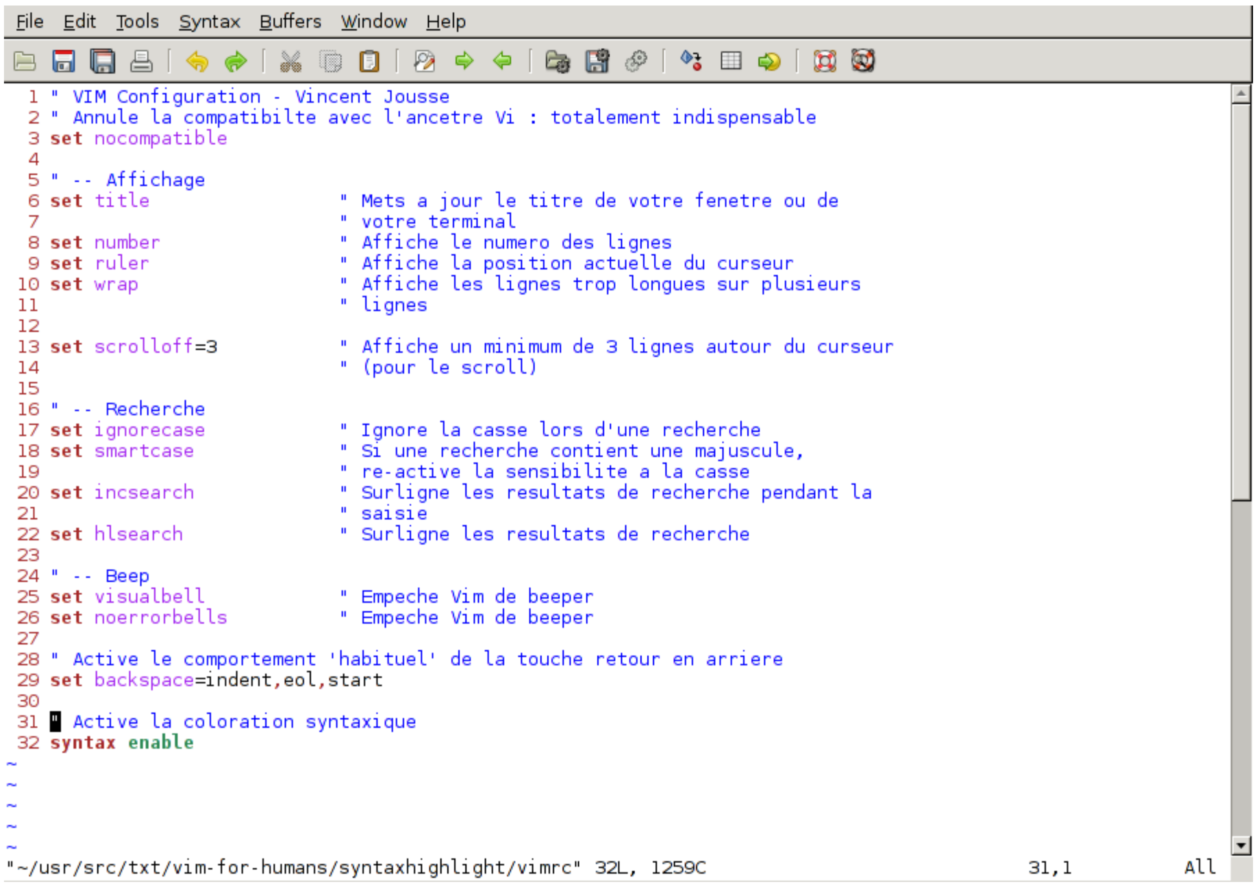
For the time being, the easiest way to test the modifications you made to your `~/.vimrc` file is to restart *Vim*. If you want to use *Vim* like a boss right now, you can type in normal mode :so ~/.vimrc or :so $MYVIMRC. It will reload the configuration without the need to restart *Vim*. :so being a shortcut for vimcmd:*:source*.

This is a good first step, but now it's time to start using a theme.



```
" VIM Configuration - Vincent Jousse
" Cancel the compatibility with Vi. Essential if you want
" to enjoy the features of Vim
set nocompatible

" -- Display
set title                 " Update the title of your window or your terminal
set number                " Display line numbers
set ruler                 " Display cursor position
set wrap                  " Wrap lines when they are too long

set scrolloff=3           " Display at least 3 lines around you cursor
                          " (for scrolling)

set guioptions=T          " Enable the toolbar

" -- Search
set ignorecase            " Ignore case when searching
set smartcase             " If there is an uppercase in your search term
                          " search case sensitive again
set incsearch             " Highlight search results when typing
set hlsearch              " Highlight search results

" -- Beep
set visualbell            " Prevent Vim from beeping
set noerrorbells          " Prevent Vim from beeping

" Backspace behaves as expected
set backspace=indent,eol,start

" Hide buffer (file) instead of abandoning when switching
" to another buffer
set hidden
~
~
~
~
~
"~/.vimrc" 33L, 1147C                                                1,1           All
```

Fig. 3.2: *Vim* after your first configuration.

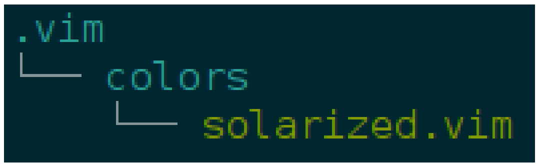


```
" VIM Configuration - Vincent Jousse
" Annule la compatibilte avec l'ancetre Vi : totalement indispensable
set nocompatible

" -- Affichage
set title                 " Mets a jour le titre de votre fenetre ou de
                          " votre terminal
set number                " Affiche le numero des lignes
set ruler                 " Affiche la position actuelle du curseur
set wrap                  " Affiche les lignes trop longues sur plusieurs
                          " lignes

set scrolloff=3           " Affiche un minimum de 3 lignes autour du curseur
                          " (pour le scroll)

" -- Recherche
set ignorecase            " Ignore la casse lors d'une recherche
set smartcase             " Si une recherche contient une majuscule,
                          " re-active la sensibilite a la casse
set incsearch             " Surligne les resultats de recherche pendant la
                          " saisie
set hlsearch              " Surligne les resultats de recherche

" -- Beep
set visualbell            " Empeche Vim de beeper
set noerrorbells          " Empeche Vim de beeper

" Active le comportement 'habituel' de la touche retour en arriere
set backspace=indent,eol,start

" Active la coloration syntaxique
syntax enable
~
~
~
~
~
"~/usr/src/txt/vim-for-humans/syntaxhighlight/vimrc" 32L, 1259C        31,1        All
```

Fig. 3.3: Default syntax highlighting.

Themes will allow you to have a nicer *Vim* than the default one. A theme will change the background color of *Vim* and the colors used for the syntax highlighting. As I said earlier, we will use the *Solarized* theme http://ethanschoonover.com/solarized (with dark or light background, it will be up to you).

To install it, you will first need to create a directory called *.vim* in the same directory than your `~/.vimrc` (that is to say, in your home directory). Note that when using Windows, the *.vim* directory is called *vimfiles*. Each time I'll be speaking of the *.vim* directory, it will be the *vimfiles* directory for people using Windows. In this *.vim* directory, create a sub directory named *colors*. Then, download the *Solarized* theme file https://raw.github.com/altercation/vim-colors-solarized/master/colors/solarized.vim (it's the same file for the light and the dark version) and copy it in your *vim/colors/* directory. You *.vim* directory should look like the picture below.



```
.vim
└── colors
    └── solarized.vim
```

Fig. 3.4: Content of the .vim directory with Solarized.

Then enable the Solarized theme in your `~/.vimrc` like shown in the code below.:

```
" Use the dark version of Solarized
set background=dark
colorscheme solarized
```

To test the light theme, you just have to change *dark* with *light* (for the *background* property).

Here is a preview of the two versions (personally, I prefer the dark one).

A bonus (if you don't use *Vim* directly in your terminal) would be to choose a font that suits your needs a little bit better. This is of course optional, but I suppose that some of you may wish to do this.

If you are using Mac OS X, I recommend the *Monaco* font that is quite friendly. Add the following lines to your `~/.vimrc` to use it:

```
set guifont=Monaco:h13
set antialias
```

You can of course change *h13* with *h12* if you want a smaller font (or with *h14* if you want a bigger one).

Under Linux I am using the *DejaVu Sans Mono* font:

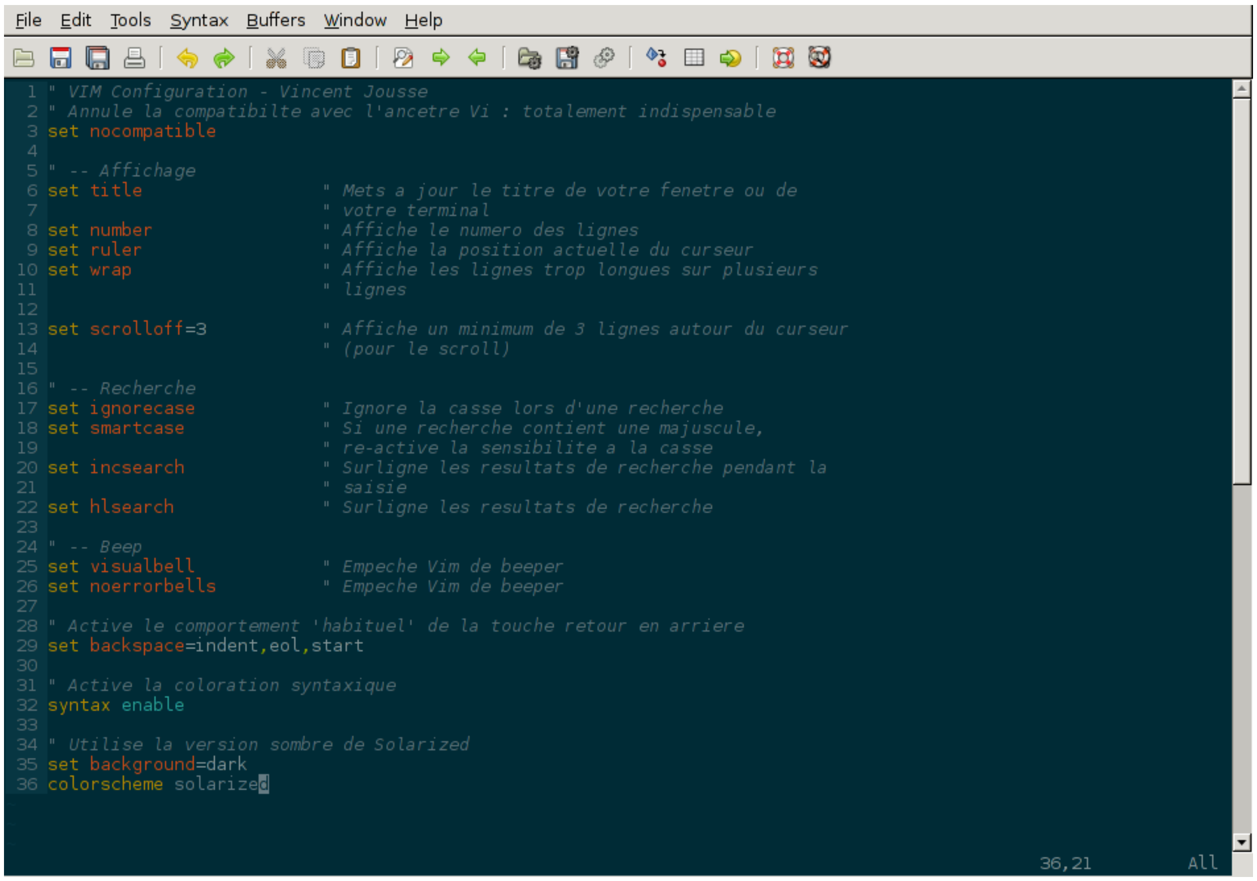
```
set guifont=DejaVu\ Sans\ Mono\ 10
set antialias
```

You can of course change the font size as you wish. To have the list of all the available fonts for your system type :set guifont:* in normal mode.

You will find the full version of the configuration file for this chapter online http://vimebook.com/link/en/syntaxhlconfig. I will not spend more time talking about the fonts as it's dependant of your operating system and not of *Vim*.

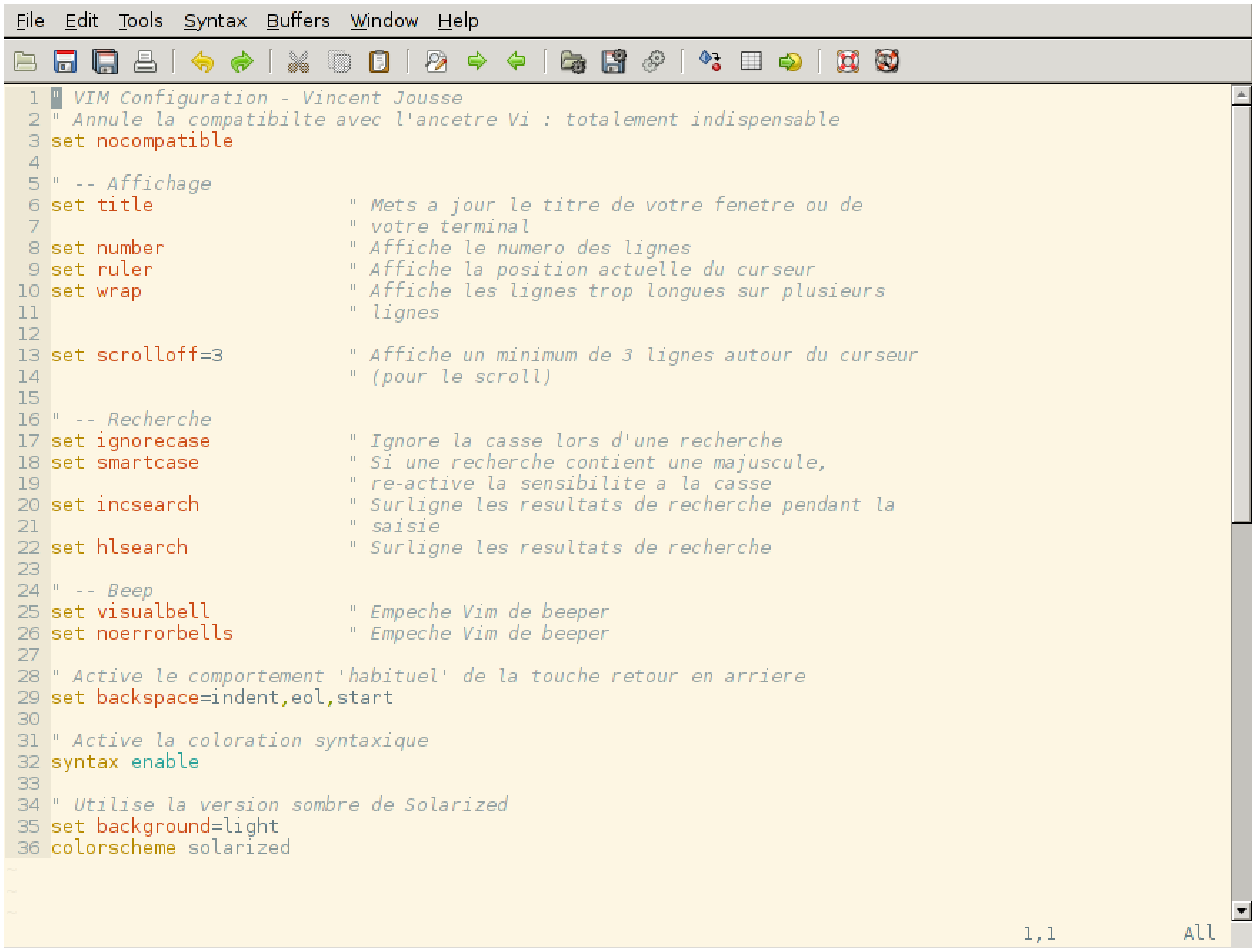
3.5 Our first plugin: the file explorer

Now we have a nice *Vim* that we can actually use with pretty colors. It's a good first step, but now, it would be cool to be able to open files without having to do *File -> Open* using the menu bar. This is the perfect opportunity to install our first plugin: a file explorer. We will use a two-step approach here: first we will install a plugin manager and then



```
" VIM Configuration - Vincent Jousse
" Annule la compatibilte avec l'ancetre Vi : totalement indispensable
set nocompatible

" -- Affichage
set title                    " Mets a jour le titre de votre fenetre ou de
                             " votre terminal
set number                   " Affiche le numero des lignes
set ruler                    " Affiche la position actuelle du curseur
set wrap                     " Affiche les lignes trop longues sur plusieurs
                             " lignes

set scrolloff=3              " Affiche un minimum de 3 lignes autour du curseur
                             " (pour le scroll)

" -- Recherche
set ignorecase               " Ignore la casse lors d'une recherche
set smartcase                " Si une recherche contient une majuscule,
                             " re-active la sensibilite a la casse
set incsearch                " Surligne les resultats de recherche pendant la
                             " saisie
set hlsearch                 " Surligne les resultats de recherche

" -- Beep
set visualbell               " Empeche Vim de beeper
set noerrorbells             " Empeche Vim de beeper

" Active le comportement 'habituel' de la touche retour en arriere
set backspace=indent,eol,start

" Active la coloration syntaxique
syntax enable

" Utilise la version sombre de Solarized
set background=dark
colorscheme solarized
```

Fig. 3.5: The dark *Solarized* theme.



```
" VIM Configuration - Vincent Jousse
" Annule la compatibilte avec l'ancetre Vi : totalement indispensable
set nocompatible

" -- Affichage
set title                  " Mets a jour le titre de votre fenetre ou de
                           " votre terminal
set number                 " Affiche le numero des lignes
set ruler                  " Affiche la position actuelle du curseur
set wrap                   " Affiche les lignes trop longues sur plusieurs
                           " lignes

set scrolloff=3            " Affiche un minimum de 3 lignes autour du curseur
                           " (pour le scroll)

" -- Recherche
set ignorecase             " Ignore la casse lors d'une recherche
set smartcase              " Si une recherche contient une majuscule,
                           " re-active la sensibilite a la casse
set incsearch              " Surligne les resultats de recherche pendant la
                           " saisie
set hlsearch               " Surligne les resultats de recherche

" -- Beep
set visualbell             " Empeche Vim de beeper
set noerrorbells           " Empeche Vim de beeper

" Active le comportement 'habituel' de la touche retour en arriere
set backspace=indent,eol,start

" Active la coloration syntaxique
syntax enable

" Utilise la version sombre de Solarized
set background=light
colorscheme solarized
~
~
~
                                                   1,1           All
```

Fig. 3.6: The light *Solarized* theme.

we will install the plugin using this plugin manager. By default *Vim* doesn't come with a plugin manager and, believe me, if you don't install one your *~/.vim* directory will soon be a real mess. So let's get started.

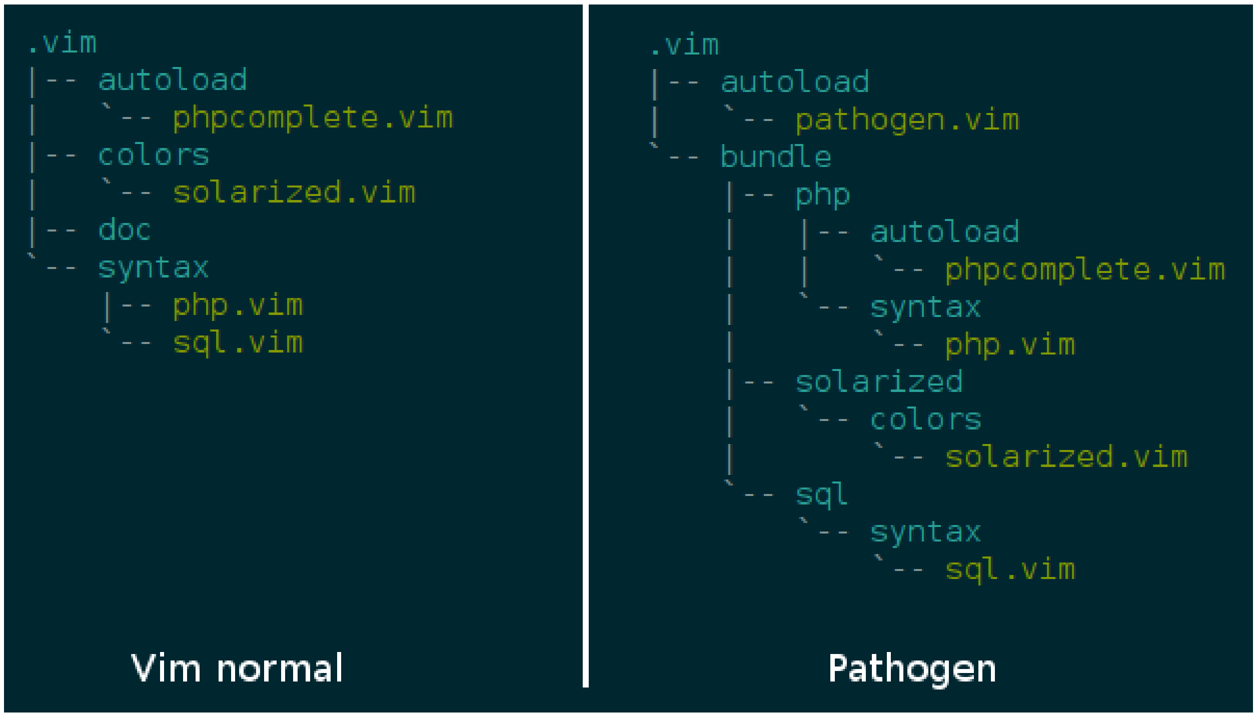
3.5.1 Plugin manager: Pathogen

Pathogen (https://github.com/tpope/vim-pathogen/) is typically the kind of plugin that you discover after having already configured your *Vim*. Then you ask yourself, "Why didn't I start this way?". Fortunately, I have a good news for you: we will be starting the right way.

First of all, let's start with a little explanation about how to install plugins using *Vim*. Plugins are installed by copying files (most of the time with the **.vim* extension) in subdirectories of your *~/.vim* directory. By the way, we've already created a subdirectory called *colors* that contains our first coloration plugin using the Solarize theme.

The main problem with this approach is that the plugins are not isolated. So you will have to copy files from different plugins in the same directory and you will soon not be able to know from what plugin a file is coming from. As a result, when you will want to remove or update a plugin, it will be a nightmare to know where the files are located.

That's why *Pathogen* is especially useful, it will allow each plugin to be located in a separate directory. Here is an example of a *~/.vim* directory before and after the usage of *Pathogen*:



```
.vim
|-- autoload
|   `-- phpcomplete.vim
|-- colors
|   `-- solarized.vim
|-- doc
`-- syntax
    |-- php.vim
    `-- sql.vim
```

Vim normal

```
.vim
|-- autoload
|   `-- pathogen.vim
`-- bundle
    |-- php
    |   |-- autoload
    |   |   `-- phpcomplete.vim
    |   `-- syntax
    |       `-- php.vim
    |-- solarized
    |   `-- colors
    |       `-- solarized.vim
    `-- sql
        `-- syntax
            `-- sql.vim
```

Pathogen

Fig. 3.7: *~/.vim* before and after Pathogen

You are totally right if you find that the version with *Pathogen* is using more directories. But believe me, those directories will save your life later. You will be able to easily remove and update plugins and you will be able to use *git* (or any other SCM software) to manage your plugins / submodules / dependencies.

Let's start by installing *Pathogen*. Create a directory called *autoload* in your *~/.vim* directory. Download *pathogen.vim* (https://raw.github.com/tpope/vim-pathogen/master/autoload/pathogen.vim) and copy it to your *autoload* directory. For the Unix/Mac OS X/Linux user, here is how to install it (if you don't have *curl*, you can use *wget -O -* instead:

```
# Create the autoload directory
mkdir -p ~/.vim/autoload
```

```
# Download and install Pathogen
curl -so ~/.vim/autoload/pathogen.vim \
    https://raw.github.com/tpope/vim-pathogen/master/autoload/pathogen.vim
```

From now on, we will install our plugins directly in the *.vim/bundle* directory that you will create right now:

```
# Create the bundle directory used by Pathogen
mkdir -p ~/.vim/bundle
```

All you have to do is to activate *Pathogen* in your `~/.vimrc` file and voila. We will put the following lines at the very beginning of the `~/.vimrc` file, right after the *set nocompatible* line. It is mandatory to put the code at the **beginning** of your `~/.vimrc` file, otherwise it will not work.:

```
" Activate pathogen
call pathogen#infect()
```

Since charity begins at home, we will tidy up our install by using *Pathogen* with our *Solarized* plugin. We just have to create a directory called *solarized* in your newly created *.vim/bundle* directory. You could name it as you want as any subdirectory of the *bundle* directory will be considered a plugin directory. Then, we will move our previously created *colors* directory in the *solarized* directory as follows:

```
# Create the plugin directory for solarized
mkdir ~/.vim/bundle/solarized
# Move solarized to the bundle dir
mv ~/.vim/colors ~/.vim/bundle/solarized
```

Currently, *Pathogen* is still the most used plugin manager for *Vim*. But recently, a new challenger has arrived, it's called Vundle https://github.com/gmarik/vundle. I have chosen to use *Pathogen* here because it's the one you will hear about the most often. But you have to know that Vundle is an interesting alternative: it's compatible with Pathogen and you can manage your plugins directly from internet (from github, the vim website, ...). For those familiar with Ruby, it's Bundler (http://gembundler.com) for *Vim*.

Our *Vim* is almost ready to be used on a daily basis. We are just missing an handy way to explore the files of a project. We will use *The NERD Tree* for that.

3.5.2 The NERD Tree: a file explorer

The NERD Tree is a plugin that will allow you to display your directory and file tree directly in *Vim*, just like in *TextMate*, *Sublime Text* or *Eclipse/NetBeans*. This is not a mandatory plugin if you want to control everything using the keyboard (I don't use it anymore myself), but it's very handy when you are starting with *Vim*.

The other solution that we will see in the *Essential plugins* chapter will be to use the *Ctrl-p* or *Command-t* plugins to find files and to use the *LustyExplorer* and *LustyJuggler* plugins to navigate between the files. Indeed, having to visualize the whole file tree to find a file is a lot slower than to find a file by its name. In the meantime, The NERD Tree will allow us to use *Vim* with a *normal* file explorer where you can click with the mouse.

First, we will prepare *Pathogen* in order to install all the files needed by *The NERD Tree*.

```
# Create the directory for The NERD Tree
mkdir ~/.vim/bundle/nerdtree
```

Then, download the latest *.zip* available on the plugin page http://www.vim.org/scripts/script.php?script_id=1658. At the time of writing, the latest version is 4.2.0. You can directly download it using this url: http://www.vim.org/scripts/download_script.php?src_id=17123.

Unzip the file and copy the content in the newly created `~/.vim/bundle/nerdtree` directory. The directory structure of your `nerdtree` directory should look like the one below:

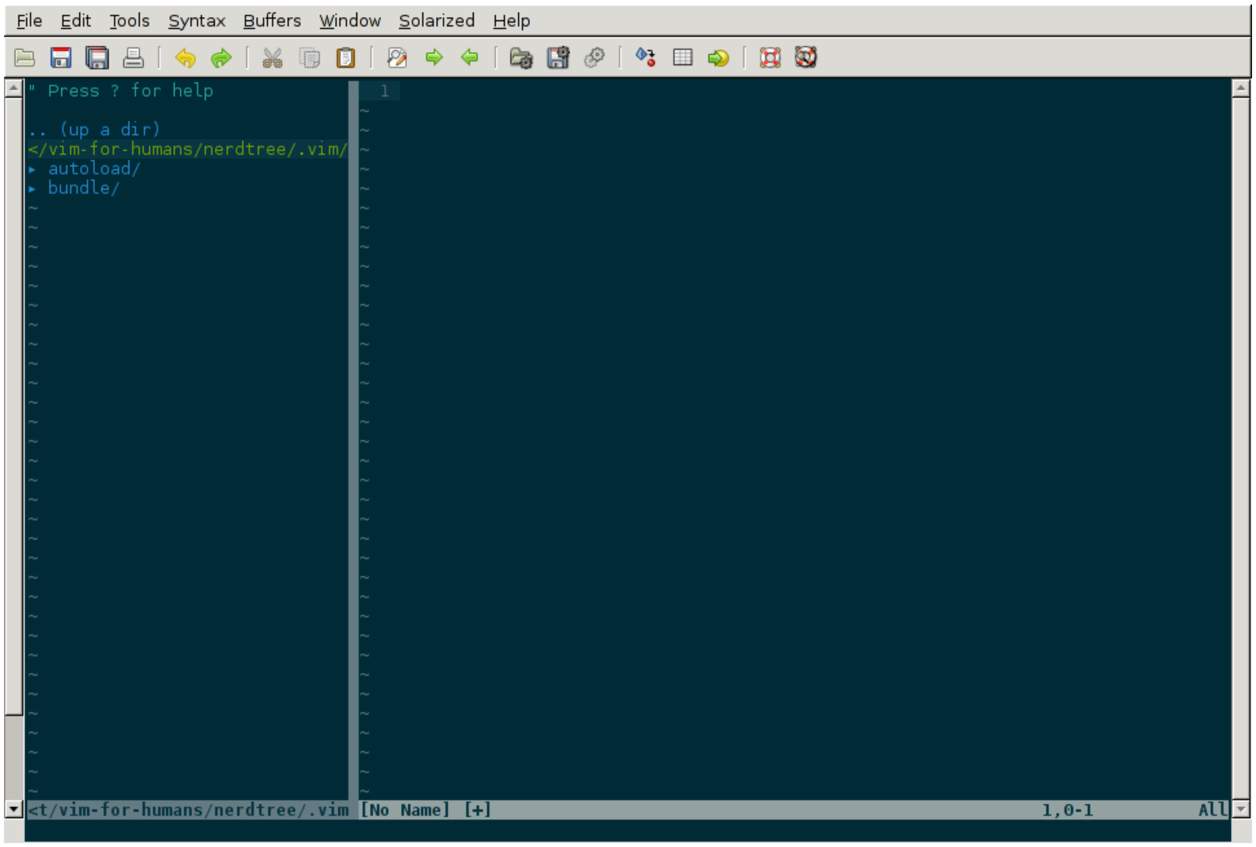
```
nerdtree
|-- doc
|   `-- NERD_tree.txt
|-- nerdtree_plugin
|   |-- exec_menuitem.vim
|   `-- fs_menu.vim
|-- plugin
|   `-- NERD_tree.vim
`-- syntax
    `-- nerdtree.vim
```

Then, you will need to activate the plugin. You can do it manually by typing :NERDTree in normal mode. If you prefer to activate *The NERD Tree* every time you open your *Vim*, add these lines to your `~/.vimrc`:

```
" Activate the NERDTree when launching vim
autocmd vimenter * NERDTree
```

I have to admit that this command is a little bit obscure. Actually it means: every time you open vim (`vimenter`), no matter what is the type of the file (`*`), launch *The NERD Tree* (`NERDTree`).

Nothing special then, *The NERD Tree* will display the tree of the directory where you've launched *Vim* as you can see on the picture below. You can use your mouse and/or your keyboard to move inside *The NERD Tree*.



```
" Press ? for help

.. (up a dir)
</vim-for-humans/nerdtree/.vim/
▸ autoload/
▸ bundle/
```

```
<t/vim-for-humans/nerdtree/.vim [No Name] [+]          1,0-1          All
```

Fig. 3.8: *Vim* with *The NERD Tree*.

You can also run commands (create, copy a file) by typing the `m` key when you are inside *The NERD Tree*. To switch between the *NERD Tree* window and your file window with your keyboard, use `Ctrl + w` and then `w`. That is to say, hold the `Control (Ctrl)` key and at the same time press the `w` key. You can then release everything and press `w` again. This shortcut is valid to switch between any *Vim* window (it's not a shortcut specific to *The NERD Tree*).

3.6 Here we go

Now, you've done the hardest part. Well, almost. We've just covered what is sorely lacking in *Vim*: a sensible default configuration. I'm not saying that you now have the best editor out there, but at least, you should be able to use *Vim* as any other *normal* text editor that you do not yet know all the possibilities. I recommend, at this stage, to start using *Vim* in your everyday life. Feel free to use the mouse and the menus for now. The primary goal here is to reduce the negative impact that *Vim* could have on your daily productivity, if not configured properly. You will gradually learn the keyboard shortcuts when the time will come.

We will now discuss what makes the uniqueness of *Vim*: the way modes are handled and the shortcuts to manipulate text. The ball is in your court now: either you are willing to change your habits and move to another level of efficiency with *Vim*, or using *Vim* as an improved notebook is the best option for you (in this case, you can stop here). It's up to you !

CHAPTER

FOUR

THE TEXT EDITOR YOU'VE ALWAYS DREAMED OF

I confess, I have some weird dreams. But hey, maybe my dreams are not as weird as they seem: dreaming of a tool that can improve all the professional areas of my life as a programmer, writer, teacher, and more doesn't seem that weird after all.

The success of *Vim* is due to its ability to **ease the text manipulations**. Certainly, it will provide you with functionalities dedicated to specific tasks (often via plugins) as syntax highlighting, spell checking, and so on, but in the end, it's always writing/fixing/handling/moving text that takes most of your time.

This is where the difference lies between *Vim* and IDEs like Eclipse/Netbeans/PhpStorm and others. Such an IDE will put the focus on the particularities of your programming language while providing basic text manipulation functionalities. *Vim* takes the opposite approach: you will by default be **very effective** at manipulating/writing text, no matter what kind of text. But then, when you feel the need, you will be able to enrich *Vim* with plugins specific to your needs and programming languages.

This chapter will cover how to use *Vim* the right way (you will begin to forget your mouse) and the logic behind all these obscure commands. By the end of this chapter, you should be able to **completely avoid using your mouse** to edit/handle text. In any case, you should force yourself not to use the mouse when learning *Vim*. It's not that hard to only use the keyboard, and it will make a huge diffrence in your day to day life.

4.1 Learning how to move: the copy/paste use case

We have already seen in the "*Essential preamble: the insert mode*" section how to switch between the insert mode (to write text) and the normal mode (for the moment, you don't need to totally understand the purpose of this mode). When you press the `i` key your cursor will switch to the insert mode (when you are already in the normal mode) and when you press the `Esc` key it will switch back to the normal mode. Well, that's cool. So now, what else?

4.1.1 Preamble

Now is the time to learn our first text manipulation: the famous copy/paste. I can already hear some of you saying that it is useless: you already know how to do that. You switch to the insert mode, you use your mouse (or you move using the directional keys while holding the `Shift` key) to select some text. Then you go to the `edit` menu and you select `copy`. Then, `edit` menu and `paste`. That works, so why not do that?

If you have understood the "*Modes: the powerful Vim secrets*" section about the ideal position of your hands on the keyboard, you should know that you did things you are not supposed to do:

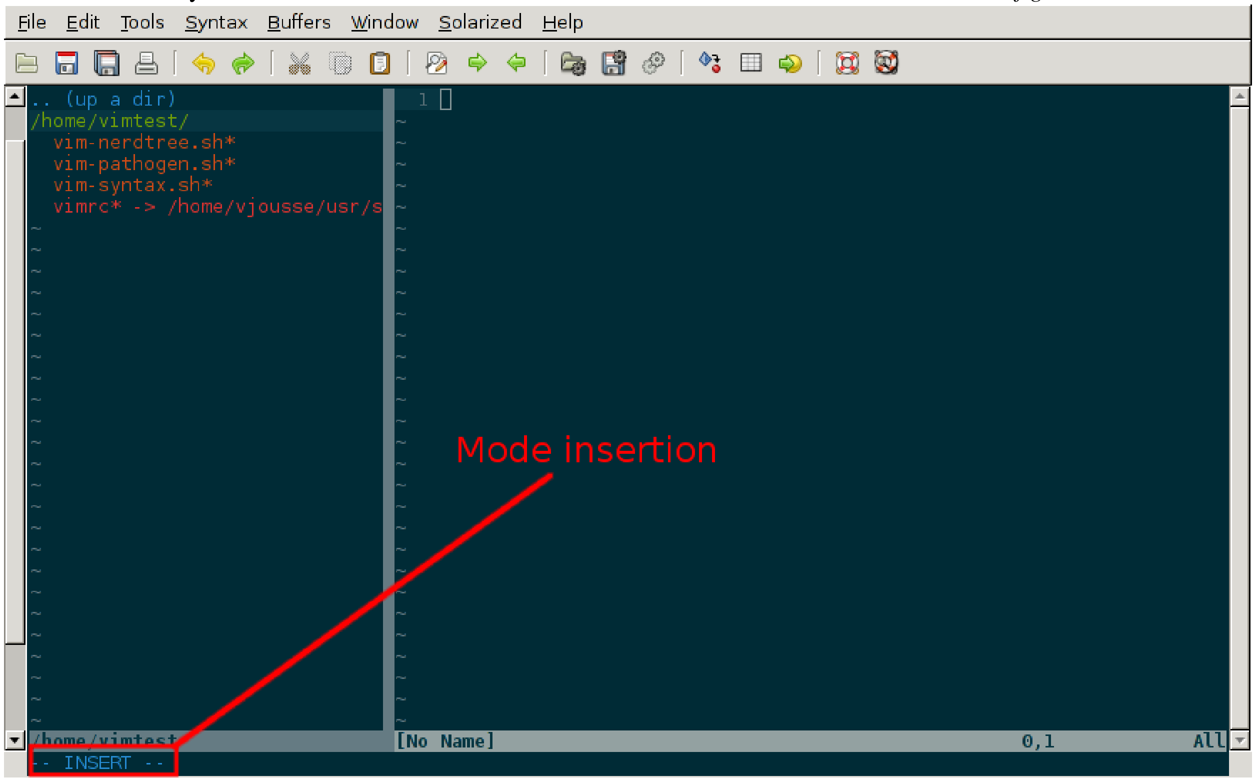
- You used your mouse
- You moved your right hand a lot from its rest position, to press the directional keys that are far away from your fingers

Of course, it doesn't really matter, but it's totally ineffective (using your mouse or moving your right hand toward the directional keys is very slow) and harmful for your hands. This is your last chance: if you are not ready to force yourself to not do it, **|vim| is not for you**. *Vim* does a perfect job at keeping your hands on the keyboard and at keeping you away from your mouse. If you don't do it, you will not be using *Vim* to its fullest. And, one day or another, **you will quit for another editor** that was made to be used with a mouse. So, should we continue?

4.1.2 Forget the mouse

If you are reading thoses lines it means that you have answered "yes" to the question above, so let's go! Our first step will be to get rid of the mouse. Then we will do the same with directional keys, but first things first.

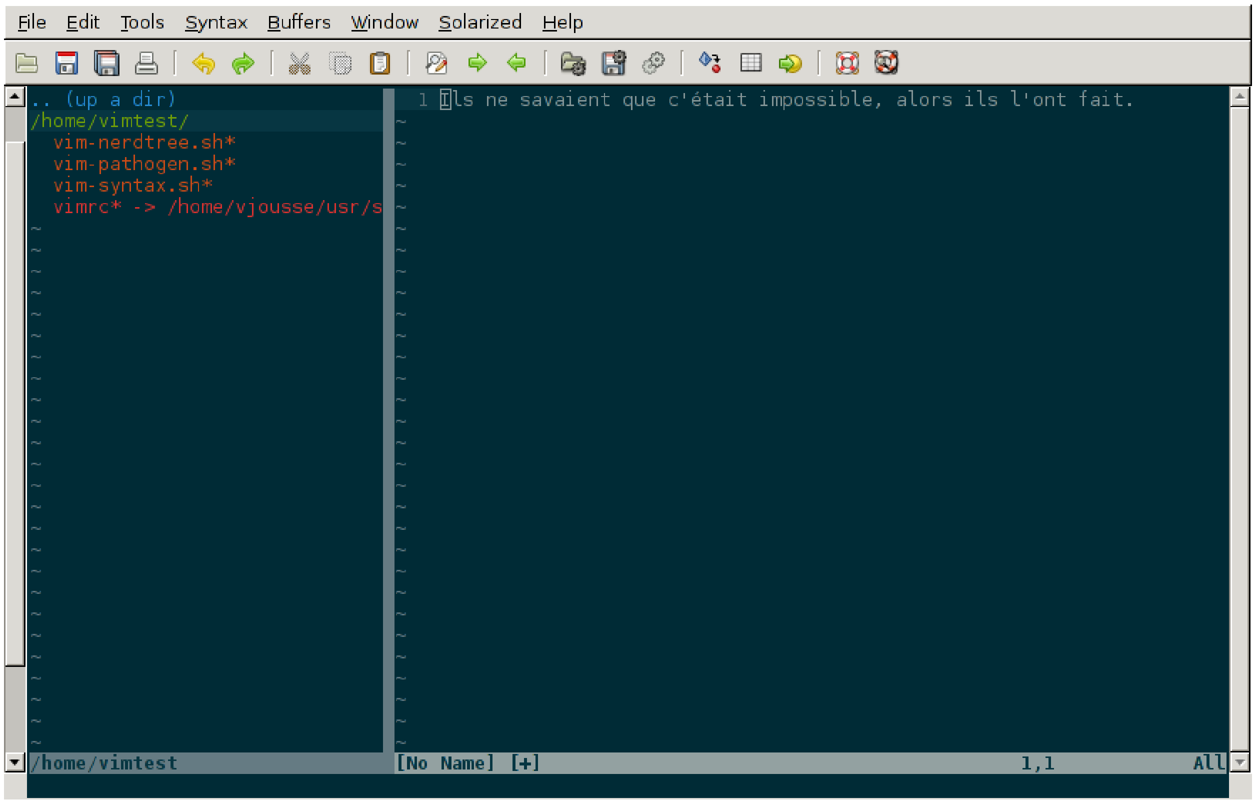
To copy/paste using *Vim*, you will have to switch to the "normal" mode (the default one when you open *Vim*). To know what the current mode is, just have a look at the bottom left of your *Vim*. You can see *Vim* in "insert" mode in *the figure below*.



```
.. (up a dir)
/home/vimtest/
  vim-nerdtree.sh*
  vim-pathogen.sh*
  vim-syntax.sh*
  vimrc* -> /home/vjousse/usr/s
/home/vimtest       [No Name]                    0,1       All
-- INSERT --
```

When there is nothing displayed at the bottom left, it's because you are currently in "normal" mode. In order to quit a mode to return to the normal one, you just have to press the `Esc` key. You may already have noticed that pressing the `Esc` key is a pain for your fingers. Don't worry, this is just temporary. I will explain why in the "*Doing without the Esc key*" section.

Let's say that you are currently in the "normal" mode and that you already have some text in you *Vim*. For example, it could be this beautiful quote from Mark Twain: "They did not know it was impossible, so they did it.". Your *Vim* should like the one in the figure below. Notice that there is nothing displayed at the bottom left.



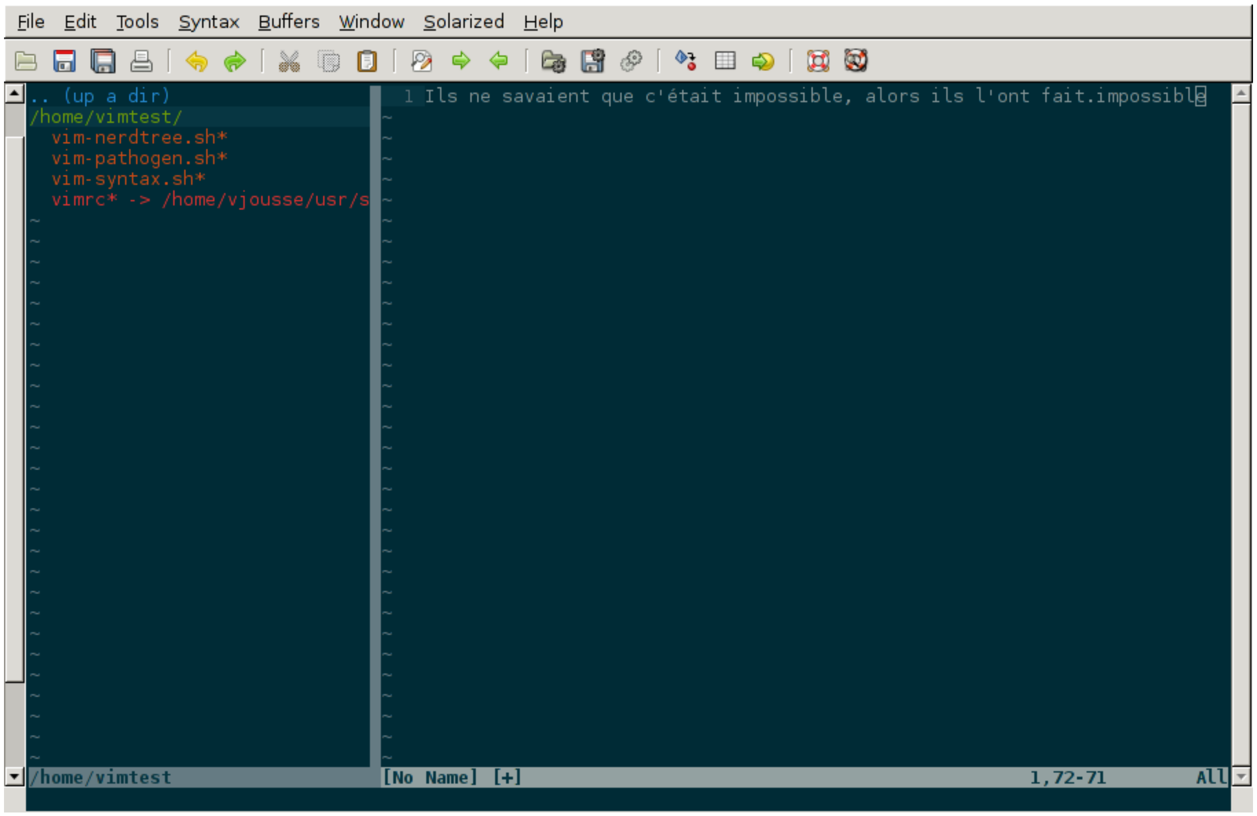
```
.. (up a dir)
/home/vimtest/
  vim-nerdtree.sh*
  vim-pathogen.sh*
  vim-syntax.sh*
  vimrc* -> /home/vjousse/usr/s
```

```
Ils ne savaient que c'était impossible, alors ils l'ont fait.
```

```
/home/vimtest        [No Name] [+]                                  1,1           All
```

The most intuitive way (but not the most efficient, we will see why a little bit later) to copy/paste the "impossible" word is to move the cursor at the first letter of the word using the directional keys, to press the `v` key (to switch to the "visual" mode), to move to the last letter of the word (you should have the word "impossible" highlighted) and then to press the `y` key (the `y` key stands for *yank*). You've just copied your first word using *Vim*. Hooray!

Then, move to the end of the sentence (in "normal" mode) and press the `p` key (the `p` key standing for *paste*). The word should now be pasted at the end, and you should have something like the figure below.



```
.. (up a dir)
/home/vimtest/
  vim-nerdtree.sh*
  vim-pathogen.sh*
  vim-syntax.sh*
  vimrc* -> /home/vjousse/usr/s
```

```
Ils ne savaient que c'était impossible, alors ils l'ont fait.impossible
```

```
/home/vimtest          [No Name] [+]          1,72-71          All
```

We can see that *Vim* uses the mode switching trick (including the "normal" mode for moving) in order to not have to use the mouse. When you will be used to switch quickly from one mode to another (and in order to do so, going without the `Esc` key will be mandatory), using the mouse will appear like a pure waste of time. But obviously, you will first need to train yourself.

4.2 Forgetting the directional keys

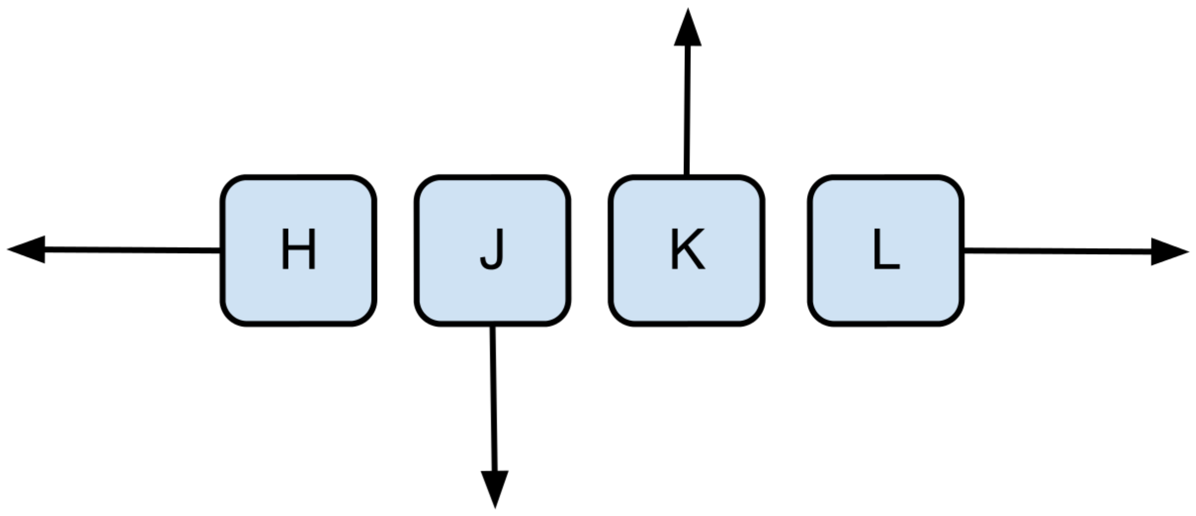
Here we are. Even if forgetting the mouse is a first good step, the real goal when using *Vim* is to forget the directional keys too. You will be faster and better when using *Vim* on the sole condition that you don't use the directional keys anymore. It will indeed force you to keep your hands on the home row and you will have to switch to the normal mode to move around. This is a prerequisite to use *Vim* at its fullest.

In this section, I will explain how to move without using the directional keys. Then, when you will know how to do it, I will give you the code that you need to put in your `~/.vimrc` to totally disable the directional keys. It was the only way I found to force me to not use the directional keys anymore.

4.2.1 Moving without using the directional keys

When in normal mode, 4 keys will allow you to move your cursor:

- the `h` key to move to the **left**
- the `j` key to move to the **bottom**
- the `k` key to move to the **top**
- the `l` key to move to the **right**



As you can notice, those keys are located on the home row so that you don't have to move your hands. Your index finger has two moves (left and bottom) while your little finger doesn't have any. You will see that this not a problem, it's even a feature: your index finger is stronger than your little finger. By checking the keyboard that was used to develop *Vi* in the "*Doing without the Esc key*" section, you will understand why.

On a side note, once you will be used to *Vim*, you will not use the left and right moves a lot. You will primarily move the cursor word by word, paragraph by paragraph or by using the search function. Here are some "fast moves" that I often use:

Key	Move
`e`	**to the end of the current word**
`b`	**to the beginning of the current word**
`w`	**to the beginning of the next word**
`^`	**to the first non white character of the line**
`$`	**to the end of the line**
`0`	**to the start of the line**

This is the mininum to move your hands in normal mode. They are also commands allowing you to first move and then to enter the insert mode directly. They are very handy because they will allow you to save a few keystrokes. Here are some that I often use:

Key	Action
`i`	enter insert mode just **before the cursor**
`a`	enter insert mode just **after the cursor**
`I`	enter insert mode **at the beginning of the line**
`A`	enter insert mode **at the end of the line**
`o`	insert a new line **below the current line**
`O`	insert a new line **above the current line**
`r`	**replace the character** under the cursor by a new one

Let's discuss that a little bit. The secrets of *Vim* rely on the contents of this chapter. There is one thing that you have to do when learning *Vim*: **use the hjkl keys** to move. If you can manage to do that, you will learn everything else on the go.

You'll find a lot of websites with all the possible commands, combinations and so on. You will learn and forgot them (depending on how useful they are to you). If you have a single effort to do: it is to use the directional keys and thus to force you to use the normal mode. Everything else will then be perfectly obvious.

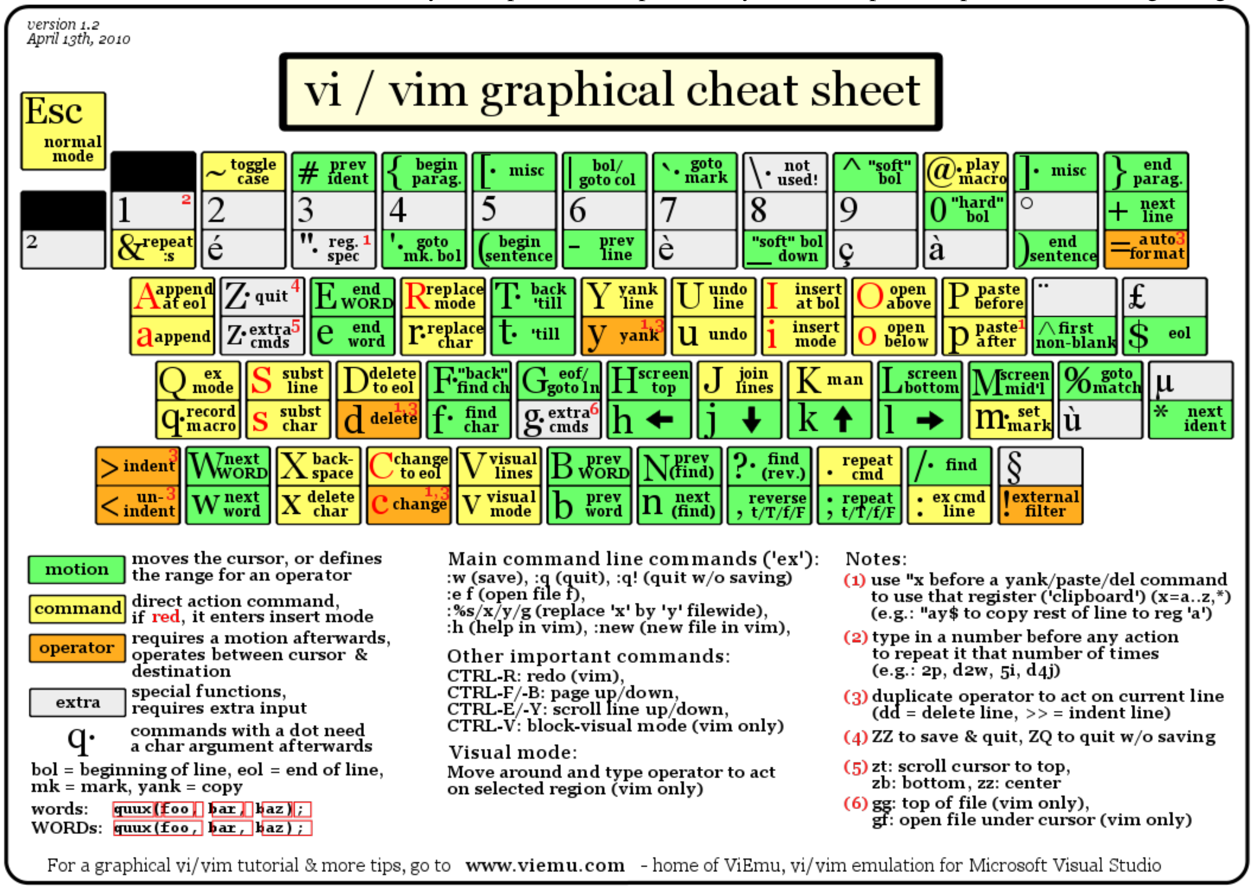
Here is the ultimate configuration that you will need to put in your `~/.vimrc` to achieve your goal: disabling the directional keys:

```
" Disabling the directional keys
map <up> <nop>
map <down> <nop>
map <left> <nop>
map <right> <nop>
imap <up> <nop>
imap <down> <nop>
imap <left> <nop>
imap <right> <nop>
```

Here we are. Believe me, this will be a little bit hard at the beginning. It was the case for me during the first two days. But then, you just forget about it and get used to it. Besides, if you are not ready to struggle for two days in order to learn *Vim* properly, then what are you doing here?!

I will not go into details on all the possible keys to move inside *Vim*, other resources do a better job at it. “A byte of *Vim*” is a good resource that you can freely download here: http://www.swaroopch.com/notes/vim/. Of course, you will also learn in *Combining keys and moves* how to use the keys wisely.

Here is an handy graphical cheat sheet that you can download on http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html. I recommend that you to print it and put it on your desktop: it helps a lot at the beginning.



motion — moves the cursor, or defines the range for an operator

command — direct action command, if red, it enters insert mode

operator — requires a motion afterwards, operates between cursor & destination

extra — special functions, requires extra input

q· — commands with a dot need a char argument afterwards

bol = beginning of line, eol = end of line, mk = mark, yank = copy

words: quux(foo, bar, baz);
WORDs: quux(foo, bar, baz);

Main command line commands ('ex'):
:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file f),
:%s/x/y/g (replace 'x' by 'y' filewide),
:h (help in vim), :new (new file in vim),

Other important commands:
CTRL-R: redo (vim),
CTRL-F/-B: page up/down,
CTRL-E/-Y: scroll line up/down,
CTRL-V: block-visual mode (vim only)

Visual mode:
Move around and type operator to act on selected region (vim only)

Notes:
(1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*) (e.g.: "ay$ to copy rest of line to reg 'a')
(2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
(3) duplicate operator to act on current line (dd = delete line, >> = indent line)
(4) ZZ to save & quit, ZQ to quit w/o saving
(5) zt: scroll cursor to top, zb: bottom, zz: center
(6) gg: top of file (vim only), gf: open file under cursor (vim only)

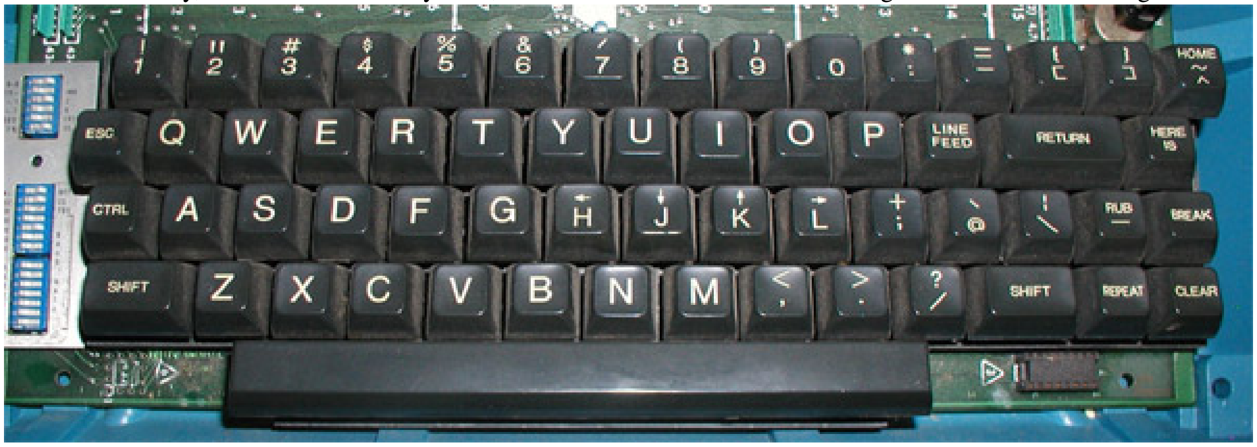
For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

Keep in mind that the main goal here is to increase your speed while keeping your hands on the “home row” and using the “normal mode”. Get down to work!

4.3 Doing without the Esc key

Let's be honest: having to use the `Esc` key to exit the "insert mode" seems to be totally counterproductive. The key is very far from the home row, you have to move your full left hand to reach it and you have to torture your little finger to press it.

To understand why the `Esc` key is used by default to exit the "insert mode", we have to go back in time a little bit. We need to find the keyboard used to program *Vi*. You can see on the picture below that the `Esc` key was very easy to reach. You can notice that the directional keys were on the home row, on the famous h, j, k, l keys. But unfortunately, it's not the case anymore, so we will have to do some changes to the default configuration.



So we agree that we need another key to exit the insert mode. There are many solutions, here are some possibilities that you can try in your `~/.vimrc`:

```
" Press the j 2 times in row
:imap jj <Esc>

" Press the j key followed by the k one
:imap jk <Esc>

" Press the i 2 times in row
:imap ii <Esc>

:imap ` <Esc>

" Shift-Space
:imap <S-Space> <Esc>

" Alt-Space.
:imap <M-Space> <Esc>
```

You can have a look at the discussion here if you want more information: http://vim.wikia.com/wiki/Avoid_the_escape_key.

4.4 Combining keys and moves

Now that we are able to move properly by using the normal mode, it's time to see how to perform other useful operations. We have already seen how to copy/paste in the *Learning how to move: the copy/paste use case* chapter, we will now have a look at how to delete/edit more easily.

In *Forgetting the directional keys* we have seen that if we want to move to the start of the next word we just have to use the `w` key. We will combine that with some new keys of the "normal mode":

- the `d` key is used to "delete"
- the `c` key is used to "delete and switch to insert mode"

Something good to know: by default, everything that is deleted is placed in the clipboard. Delete behaves in the same way cut does in other applications.

The particularity of these keys is that they are waiting for a "move order" to know what should be deleted. As a result, you will need to provide one of the keys that we have discussed in the *Forgetting the directional keys* chapter.

Here are some examples:

Action	Result
the `d` key then the `w` key	deletes all the characters until the next word
the `c` key then the `w` key	deletes all the characters until the next word and switch to the "insert mode"
the `d` key then the `$` key	delete everything until the end of the line
the `d` key then the `^` key	delete everything until the start of the line

To copy, you can use:

Action	Result
the `y` key then the `w` key	copy the characters until the next word
the `y` key then the `$` key	copy everything until the end of the line
the `y` key then the `^` key	copy everything until the first non blank character of the line

All you have to do next is to press the `p` key to paste the text you have copied above. By default, the `p` key will paste the text after the current position of the cursor. If you want to paste before the position of the cursor, use the `P` key.

From time to time, you may also want to be able to delete some text. Here are some useful commands to do so:

Action	Result
`dd`	delete the current line and put it in the clipboard
`x`	delete the character under the cursor
`X`	delete the character before the cursor

Most of the moves can be prefixed by a multiplier number. Here are some examples:

Action	Result
`2dd`	delete 2 lines
`3x`	delete 3 characters forward
`3X`	delete 3 characters backward
`2yy`	copy 2 lines in the clipboard
`5j`	move 5 lines downward

4.5 Search / Move quickly

Now that we know the basic commands for editing text with *Vim*, let's see how we can move faster in our document. We have already mentioned the `w`, `b`, `^` and `$` keys that allow us to move to the end of a word, to the beginning of a word, to the beginning of a line, and to the end of a line, respectively. First, let's see how to "scroll" without using the mouse. Note that all these commands are for the "normal mode".

4.5.1 Scrolling pages

To scroll page by page, you must use:

- `Control` + `f` to move to the next page (`f` = forward)
- `Control` + `b` to move to the previous page (`b` = backward)

These shortcuts will allow you to move quickly in your document.

You can also:

- Move to the top of the file by typing `gg`
- Move to the end of the file by typing `G`
- Move to the line number 23 by typing `:23`

4.5.2 Marks

When I'm moving inside a file, I often need to go back to some previous points. For example, when I go to the beginning of the file while I am working in the middle of it, I like to come back directly to where I was working before moving to the beginning. Fortunately, *Vim* has everything for it through the use of **markers**. Markers are simply "bookmarks" that allow you to move quickly through the file.

You can put a marker by pressing `ma`. To move your cursor to the position of the marker, just press `'a`. You can place as many markers as you want by changing `a` with any letter of the alphabet (this is called a register in *Vim*'s language). To place another marker you can, for example, use the letter `d`. Thanks to `md` you will put the marker and with `'d` you will move to the position of the marker.

4.5.3 Search

In "normal mode", you can start a search by using the `/` key followed by the text you want to search and the `Enter` key. Thanks to our *Vim* configuration you should see your search occurrences highlighted at the same time as you type. By default, the search is not case sensitive (no difference between upper and lower case). However, as soon as you will type a capital, the search will become case sensitive. You can move forward to the next search result with the `n` key. To move backward, use the `N` key.

As a reminder, here are the corresponding lines of your configuration:

```
" -- Search
set ignorecase            " Ignore case when searching
set smartcase             " If there is an uppercase in your search term
                          " search case sensitive again
set incsearch             " Highlight search results when typing
set hlsearch              " Highlight search results
```

Be careful, by default, the search is using POSIX regular expressions. If you want to search for characters commonly used in regular expressions (like [ ] ^{ } $ /) do not forget to prefix them with .

You can also search for the word that is directly under your cursor through the `*` key. the `*` key will search forward, the `#` key will search backward.

4.6 Visual mode

I have already mentioned the "visual mode" when explaining how to Copy/Paste, but I will do a little reminder here, just in case.

When you are in "normal mode", press the `v` key to switch to the "visual mode". You will then be able to select individual characters or entier lines thanks to the various ways of moving that you just learned above. You can then copy the selected text with the `y` key and paste it with the `p` key. To cut it just use the `d` key instead of the `y` key.

In "normal mode" you will be able to use the `v` key to select line per line. And of course, use the `Esc` key or ;; to switch back to "normal mode".

4.7 It's your turn!

You should now be able to only use the keyboard to manipulate and edit text in *Vim*. I have only skimmed over the power of *Vim* here, but it should be enough to survive. I have given you the bare minimum here, but this minimum should allow you to enjoy *Vim* and to not use the mouse anymore.

It's now your turn to read all the many resources available on the Internet describing all the possible moves and combinations.

Here is a list of resources that could be useful to you:

- A byte of *Vim* http://www.swaroopch.com/notes/vim/en
- A beautiful Wiki : http://vim.wikia.com/wiki/Vim_Tips_Wiki
- Videos from Peepcode : https://peepcode.com/products/smash-into-vim-i and https://peepcode.com/products/smash-into-vim-ii
- Derek Wyatt's blog http://www.derekwyatt.org/vim/vim-tutorial-videos/

To awaken the child in you, I urge you to go have fun with http://vim-adventures.com/. This is a role playing online game that aims to teach you to master *Vim*! Here is an overview:

VIM Adventures
vim-adventures.com
Level 1
VIM Adventures Profile
VIM Adventures
Learning VIM while playing a game
Press any key to start!
© All rights reserved to Doron Linder - 2012. Legal issues and credits

Now we will go to the next level: the use of plugins, or how to make *Vim* indispensable.

CHAPTER

FIVE

ESSENTIAL PLUGINS

Let's be clear, using *Vim* without plugins is almost useless. It's the usage of plugins that will allow you to boost your productivity. You don't need a lot of them, but you do need the good ones.

Of course, *Vim* can be used without any plugins and it can be useful to know how to use it without needing to install anything. Indeed, on most servers, you will have zero plugins installed. That's why knowing how to open and save a file and knowing how to switch between files just by using default commands is very useful. However, for your writing or coding needs, plugins are mandatory.

5.1 Managing and switching between files : *Lusty Explorer*

We've already talked about NerdTree in *The NERD Tree: a file explorer* and we have seen that thanks to it, we can have a project explorer in a sidebar. One of the problems of this plugin is that it was not designed to be used with the keyboard. You can still use the keyboard, but it will not be as efficient as a plugin developed with keyboard usage in mind.

The first plugin that I install when I have to use *Vim* is *Lusty Explorer* (http://www.vim.org/scripts/script.php?script_id=1890). This plugin will allow you to navigate between the files on you hard drive in order to open files without using the mouse. Moreover, it will allow you to easily switch between you opened files, called buffers in *Vim* terms.

To install it, first download the latest version of the script here: http://www.vim.org/scripts/script.php?script_id=1890. At the time of writing, it's on version 4.3: http://www.vim.org/scripts/download_script.php?src_id=17529). Once installed, put it in your `.vim/` folder. It should look like this:

```
.vim
|-- autoload
|   `-- pathogen.vim
`-- bundle
    |-- lusty-explorer
    |   `-- plugin
    |       `-- lusty-explorer.vim
```

If you have done everything right since the beginning, your `.vim/` directory should now look like this:

```
.vim
|-- autoload
|   `-- pathogen.vim
`-- bundle
    |-- lusty-explorer
    |   `-- plugin
    |       `-- lusty-explorer.vim
    |-- nerdtree
    |   |-- doc
    |   |   `-- NERD_tree.txt
    |   |-- nerdtree_plugin
    |   |   |-- exec_menuitem.vim
    |   |   `-- fs_menu.vim
    |   |-- plugin
    |   |   `-- NERD_tree.vim
    |   `-- syntax
    |       `-- nerdtree.vim
    `-- solarized
        `-- colors
            `-- solarized.vim
```

Let's see how to use it. If we take a look at the documentation, here is what we find:

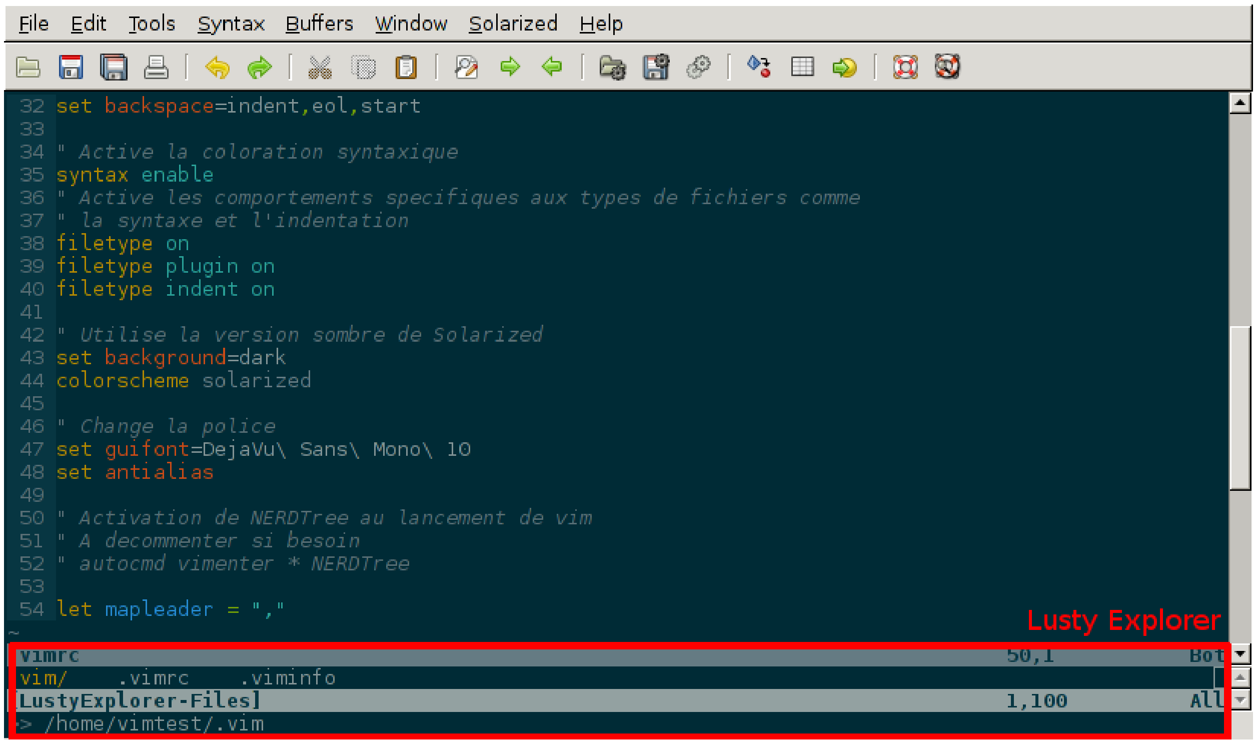
```
<Leader>lf  - Opens filesystem explorer.
<Leader>lr  - Opens filesystem explorer at the directory of the current file.
<Leader>lb  - Opens buffer explorer.
<Leader>lg  - Opens buffer grep.
```

We can see that the documentation is referring to a key named `<Leader>` that you need to combine with keys like *lf*, *lr*, *lb* et *lg*. The `<Leader>` key is a special key that we need to define in our `~/.vimrc`. Almost each plugin will need this special key to be defined, so we will use it a lot. This is a good way to avoid collisions with the default shortcuts of *Vim*.

So, we need to choose a key to be our `<Leader>` key. By default, *Vim* uses \ as a `<Leader>` key. I don't know about you, but for me this is not handy at all. I don't love to use my little finger too much. So I always replace the default `<Leader>` key with the `,` key. You can of course choose another key, but lot of people are using `,`: it's up to you. To tell it to *Vim*, you will need to add a line in your `~/.vimrc` as follows:

```
let mapleader = ","
```

Ensure that the modification has been made and taken into account by *Vim*. This can be done by restarting *Vim*, or by typing :so ~/.vimrc or :so $MYVIMRC in normal mode. Once this is done, you should be able to do `,lr` (if you choose `,` as your `<Leader>` key) and you should see something like the picture below in your *Vim*.

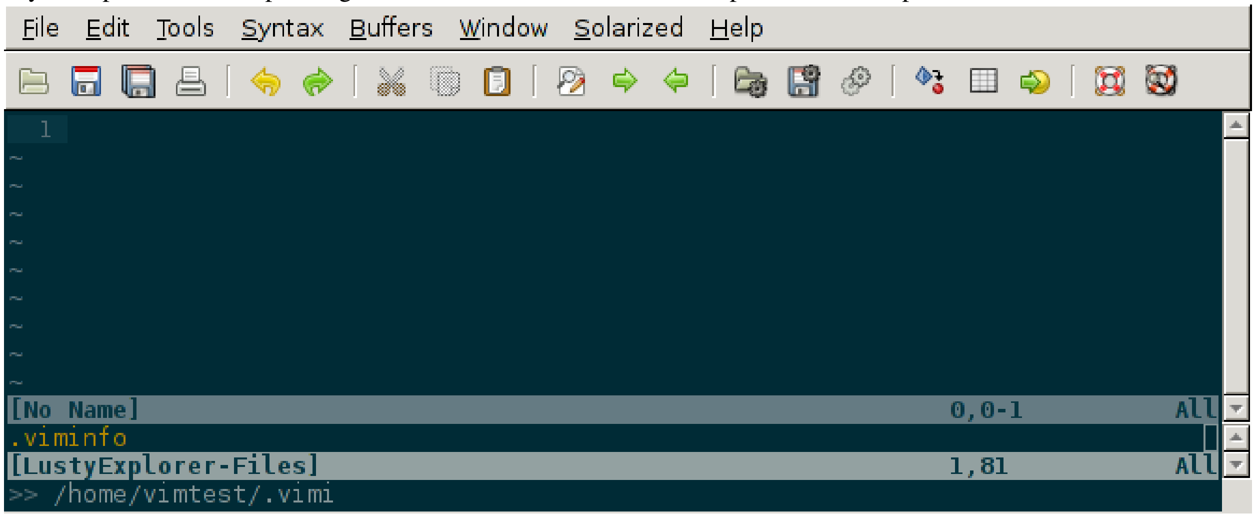


```
set backspace=indent,eol,start

" Active la coloration syntaxique
syntax enable
" Active les comportements specifiques aux types de fichiers comme
" la syntaxe et l'indentation
filetype on
filetype plugin on
filetype indent on

" Utilise la version sombre de Solarized
set background=dark
colorscheme solarized

" Change la police
set guifont=DejaVu\ Sans\ Mono\ 10
set antialias

" Activation de NERDTree au lancement de vim
" A decommenter si besoin
" autocmd vimenter * NERDTree

let mapleader = ","
~
vimrc                                                          50,1        Bot
vim/    .vimrc    .viminfo
[LustyExplorer-Files]                                          1,100       All
>> /home/vimtest/.vim
```

The next thing to do is to deactivate *The Nerd Tree* by commenting the corresponding line like I have done on the screenshot above. It will not be useful anymore as *Lusty Explorer* is a better replacement when using the keyboard.

You can see on the *lusty* screenshot that *Lusty Explorer* is made of two parts. The bottom part is about the current directory you're exploring and the top part is the content of this directory. The current item is highligthed. For example, on the *lusty* screenshot above, the current item is the `.vim/` directory, highligthed in yellow (the color could be different, it depends on your theme).

Lusty Explorer uses something called *Fuzzy matching* that will allow you to type only a small part of the file you want to select. This part can be everything: the begining of the filename, the middle, the end or just letters composing the file to select. In the example above, if I enter `.vimi` in the *Lusty* window, `.viminfo` will be selected without needing to specify the full name. Then I just need to press the `Enter` key to open the corresponding file in *Vim*. You can see this particular example in the screenshot above.



```
  1
~
~
~
~
~
~
~
~
~
[No Name]                                          0,0-1       All
.viminfo
[LustyExplorer-Files]                              1,81        All
>> /home/vimtest/.vimi
```

Here are some handy handy shortcuts of *Lusty Explorer*:

- `Control` + `n` select the next file/directory
- `Control` + `p` select the previous file/directory
- `Control` + `w` go the the parent directory
- `Control` + `e` create a new empty file (unsaved) with the current name entered in *Lusty Explorer*. If you want to save the file, you just have to use :w.

So *Lusty Explorer* can be used for two things: navigate your filesystem with `,lr` and `,lf`, and switch between your opened files (buffers) with `,lb`. Personally, I don't use the `,lg` keys a lot to search in the buffers, but it's up to you.

In order to get familiar with *Lusty Explorer* you should try to open multiple files with `,lr` or `,lf`. Then, try to switch between the opened files with the help of `,lb`. This is the combination I'm using the most on a day to day basis.

This plugin is totally mandatory and adds a lot of value to *Vim* as it allows you to avoid using the mouse to open files. Be sure to take the time to learn how to use it, it's a great time investment.

5.2 Searching files on disk: *Ack*

At some point, you will need to search for a particular pattern inside your codebase. *Vim* can help help you to do so with a plugin that uses *Ack* under the hood.

Ack (http://betterthangrep.com/) is a program written in *perl* that replaces the good old *grep* to search inside files. It's *grep*, but better. However, it has one little disadvantage: OS's rarely have it installed by default. So, as you may have guessed, the first thing to do will be to install it. As it depends on the OS you are running, you will have to refer to the installation instructions to know how to intall it for your particular case: http://github.com/mileszs/ack.vim#installation.

For Debian/Ubuntu, run: `sudo apt-get install ack-grep`. For Mac OS X, first you will need Homebrew (http://mxcl.github.com/homebrew/). Then, you will need to open a terminal and type `brew install ack`. For people using MacPorts the command will be: `sudo port install p5-app-ack`. For Windows, install Strawberry Perl (http://strawberryperl.com/) and in a command shell execute `C:\>cpan App::Ack`. You should now be able to use the **ack** command in your terminal instead of **grep**.

Now, we're ready for the big thing. Go to the ack plugin page (http://www.vim.org/scripts/script.php?script_id=2572) and download the last version (at the moment, it's the 0.3.1 version). Uncompress it in your `~/.vim/bundle/` directory so that you have a structure like the one below:

```
bundle
|-- ack
|   |-- doc
|   |   `-- ack.txt
|   `-- plugin
|       `-- ack.vim
...
```

As always, be sure that your modifications are taken into account by restarting *Vim* or by entering :source ~/.vimrc while in normal mode.

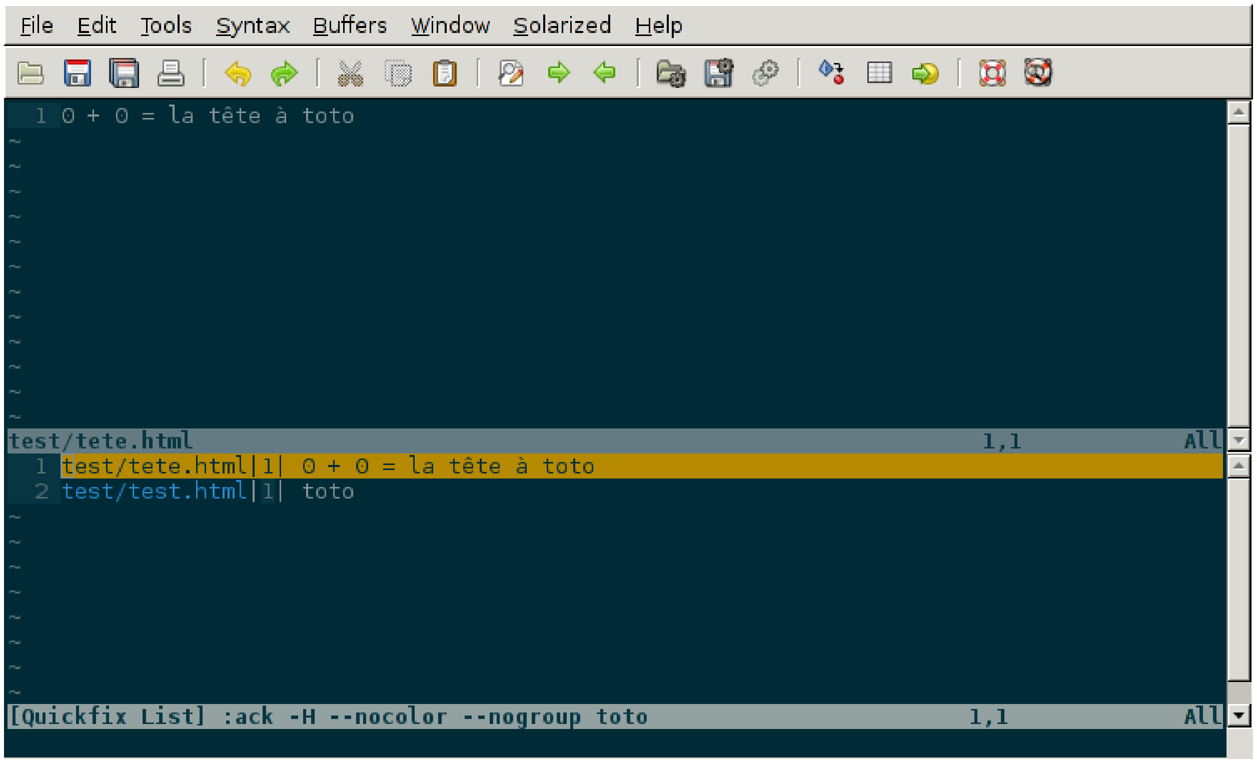
Then we will need to add some lines to our `~/.vimrc` file to ease the use of the plugin ::

```
" Default params for ack
let g:ackprg="ack -H --nocolor --nogroup --column"
" Add a mark and search
nmap <leader>j mA:Ack<space>
" Add a mark and search for the word under the cursor
nmap <leader>ja mA:Ack "<C-r>=expand("<cword>")<cr>"
nmap <leader>jA mA:Ack "<C-r>=expand("<cWORD>")<cr>"
```

Ack will start the search from the directory of the file currently opened. Here are some examples (assuming that your `<Leader>` key is the `,` key):

- `,j` *toto* : will search for *toto* starting from the directory of the current file,
- `,ja` with your cursor on a word will search for this word.

The results will be displayed in a window called the *Quickfix Window*, as you can see below.



```
  1 0 + 0 = la tête à toto
test/tete.html                                          1,1          All
  1 test/tete.html|1| 0 + 0 = la tête à toto
  2 test/test.html|1| toto
[Quickfix List] :ack -H --nocolor --nogroup toto        1,1          All
```

Here are some commands available inside this window:

- **o** : open (same as <Enter>)
- **go** : preview display (open the file but keep the focus on the ack.vim results)
- **t** : open in a new tab
- **T** : open in a new background tab
- **h** : open and split the window horizontally
- **v** : open and split the window vertically
- **q** : close the quickfix window

By default, Ack doesn't search in files that are not relevant. For example, it will not search in temp files or in files used by your favorite revision control system. If you want Ack to search into any file, independantly of its type, you need to specify the `-u` option in your `~/.vimrc` ::

```
" Default params for Ack
let g:ackprg="ack -H -u --nocolor --nogroup --column"
```

5.3 Searching files on disk: Ctrlp

Here we will not search inside files like we did with Ack. Instead, we will search for files to open with *Vim*. This can be very handy when you're working on a project where files are everywhere in the directory tree.

As always, we will start by installing the plugin. For once, this plugin has a dedicated page that you will find here: http://kien.github.com/ctrlp.vim/. Scroll to the bottom to download the latest version in the "Direct Downloads" section. For the laziest, here is the direct link: http://github.com/kien/ctrlp.vim/zipball/master. Uncompress the archive in your `~/.vim/bundle/` directory, so that you get something like this:

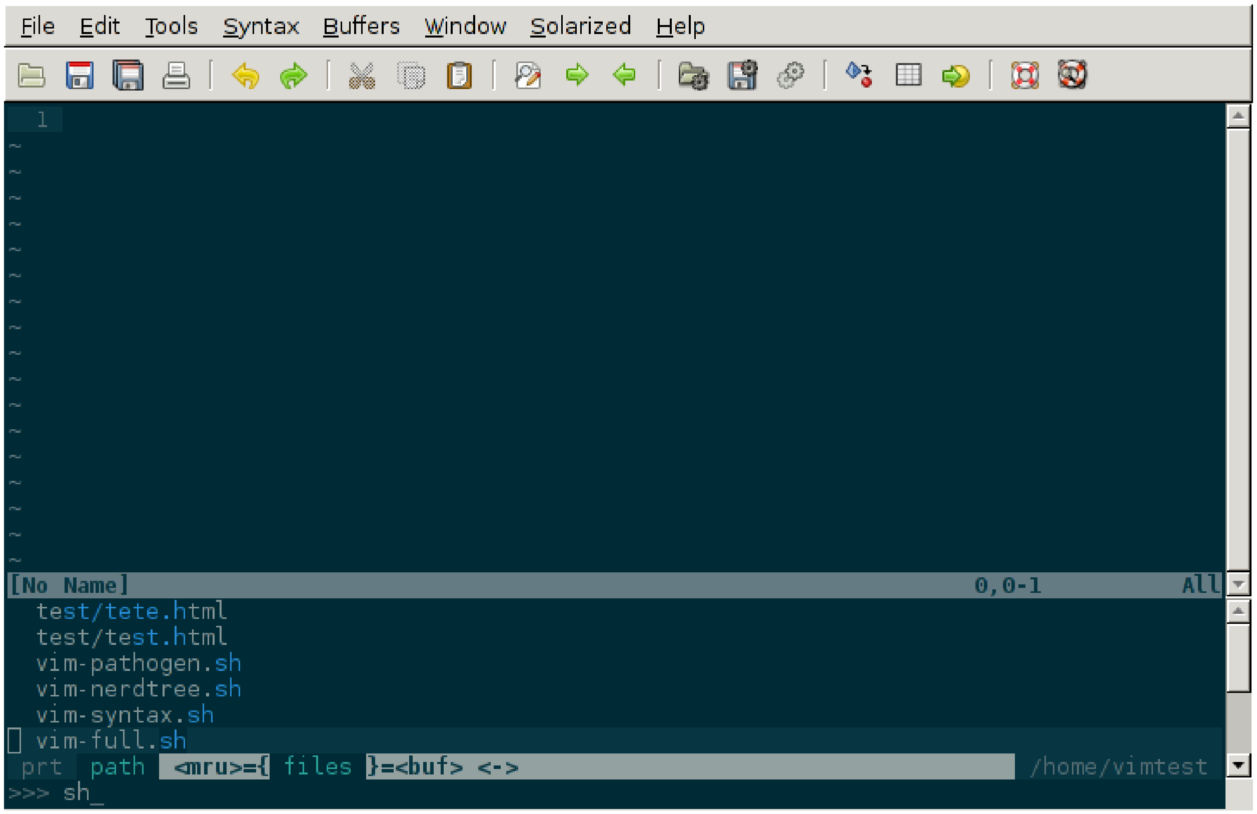
```
bundle
|
...
|-- ctrlp
|   |-- autoload
|   |   |-- ctrlp
|   |   |   |-- bookmarkdir.vim
|   |   |   |-- buffertag.vim
|   |   |   |-- changes.vim
|   |   |   |-- dir.vim
|   |   |   |-- line.vim
|   |   |   |-- mixed.vim
|   |   |   |-- mrufiles.vim
|   |   |   |-- quickfix.vim
|   |   |   |-- rtscript.vim
|   |   |   |-- tag.vim
|   |   |   |-- undo.vim
|   |   |   `-- utils.vim
|   |   `-- ctrlp.vim
|   |-- doc
|   |   `-- ctrlp.txt
|   |-- plugin
|   |   `-- ctrlp.vim
|   `-- readme.md
...
```

As always, be sure that your modifications are taken into account by restarting *Vim* or by entering :source ~/.vimrc while in normal mode.

Now we need to add the shortcut to our `~/.vimrc` to invoke CtrlP like in the listing below. Personnaly, I've chosen `,c`, but you can choose whatever you want.:

```
let g:ctrlp_map = '<leader>c'
```

Here is CtrlP in action.



```
[No Name]                                                0,0-1          All
  test/tete.html
  test/test.html
  vim-pathogen.sh
  vim-nerdtree.sh
  vim-syntax.sh
  vim-full.sh
 prt  path  <mru>={ files }=<buf> <->                          /home/vimtest
>>> sh_
```

Launch it with `,c` and then type the name of the file you want to search. When the searched file will be selected first, you will just have to press the `Enter` key to open it.

CtrlP can be used for navigating through the opened files (like Lusty), but I find Lusty handier for that. You can also use to navigate directly through your code by "following" your functions thanks to something called the tags (like you can do when using Eclipse). It's a too big topic for the scope of this book, but if you're interested you can read this blog article on the topic: http://andrew-stewart.ca/2012/10/31/vim-ctags.

5.4 Advanced plugins

Writing an entire book about the *Vim* plugin is definitely something doable, but I have to admit that I don't have enough courage. So, I will stop here with the plugins thing. However, below is a list of some plugins that may interest you. This list comes from a poll I did on Twitter asking my followers what were the most useful *Vim* plugins to them. Here it is:

- **neocomplcache**. An automatic completion plugin. It can autocomplete file names, language attributes, snippets and a lot more. The Github repo: https://github.com/Shougo/neocomplcache
- **surround**. With this plugin, you can manage (change, add, delete) everything that "surrounds": parenthesis, brackets, quotes, etc. For example, you will be able to change "Hello world!" with 'Hello world!' or <q>Hello world!</q> with a simple key combination. The Github repo: https://github.com/tpope/vim-surround
- **fugitive**. If you work with source code, you have to use a version control system. If it's not the case, you can go flog yourself. Otherwise, if you're using Git, fugitive was made for you. It allows you to manage your git command directly inside *Vim*. The Github repo can be found here: https://github.com/tpope/vim-fugitive
- **syntastic**. Syntastic will check the syntax of your source code. It will directly display your syntax errors in your *Vim*, much like, for example, Eclipse. It can be a time saver if you edit a lot of code. The Github repo can be found here: https://github.com/scrooloose/syntastic

- **ctags + ctrlp**. Ctags is a small external program that will parse your source code and allow you to “follow” your functions calls through your source code. Very useful to navigate into your source code. Used with **ctrlp** described above, it will soon become a must. Everything is explained here: http://andrew-stewart.ca/2012/10/31/vim-ctags.

CHAPTER

SIX

CHEATSHEET & EXAMPLES

So here we go, we now have everything needed to make a good start with *Vim*. It should be enough to use it on a daily basis. It's the most important thing to do if you want to successfully use *Vim*: developing the habit of using it everyday. Once you'll have this habit, everything should seem natural to you.

The aim of this chapter is to provide you with a reference for the most common things you'll have to do with *Vim* so that, when you are lost, you know where to search for help. This chapter contains two parts. The first one is a set of Q/A covering the most common problems that beginners face. The goal is to answer the question of "f**k, how can I do this, it was so simple with my old editor". The second is a non-exhaustive list of the most useful *Vim* commands.

6.1 Questions / Answers

6.1.1 How do I quit *Vim*?

First you need to be in normal mode. If you don't know if you are in normal mode, just press the `Esc` key or the `;` key (depending on your configuration) multiple times. This should bring you to normal mode. Then, type :q to exit *Vim*. The problem is that most of the time, *Vim* will not let you quit immediately. For example, if you have unsaved changes, *Vim* will not let you quit. You can cancel your modifications by using ! like this: :q!. You can also save your modifications and save like this: :wq.

6.1.2 How do I 'save as'?

In normal mode, if you type :w, *Vim* will by default save your modifications into the current file. If you want to use another filename to "save as", you just need to specify it after w like this: :w myfile.txt. *Vim* will save your file under the name *myfile.txt*. However, be aware that *Vim* will not open *myfile.txt*, it will stay on the previous file.

If you want *Vim* to save under the filename *myfile.txt* and then open the file in the current buffer, you will have to use :sav myfile.txt.

6.1.3 How do I copy/cut and paste?

This one is easy, and there is already a full chapter on it: *Learning how to move: the copy/paste use case*.

In short:

- Switch to visual mode with the `v` key,
- Select the text you want to copy by moving the cursor,
- Copy using the `y` key or cut using the `x` key or the `d` key,
- Paste after the cursor using the `p` key or before using the `P` key.

6.1.4 How do I create a new file?

The traditional way to create a file is to type, in normal mode, :e myfile.txt to open an empty buffer. Then, save your buffer using :w. Il will be saved as `myfile.txt` in the current directory.

You can also use Lusty Explorer (cf. *Managing and switching between files : Lusty Explorer*). To do so, launch it using `,lr` or `,lf` (supposing that your leader key is `,`), type the name of the file you want to create and then press the `Control` key and the `e` key at the same time. You can then save as above.

6.1.5 How do I undo/redo?

To undo, just press the `u` key while in normal mode. To undo your undo (and so, to redo) press the `Control` key and the `r` key at the same time.

6.2 Reminders

6.2.1 Files

Expected result	Action	Comments
Save	:w	
Save as	:w filename.txt	Save as filename.txt but don't open filename.txt
Save as / open	:sav filename.txt	Save as and open filename.txt
Quit without saving (force quit)	:q!	
Save and quit	:wq	
Save as root	:w !sudo tee %	

6.2.2 Movement

Expected result	Action
Move one character left	`h`
Move one line down	`j`
Move one line up	`k`
Move one character right	`l`
Move to the end of the word	`e`
Move to the beginning of the word	`b`
Move to the beginning of the next word	`w`
Move to line 42	`:42`
Move to the beginning of the file	`gg` or `:0`
Move to the end of the file	`GG` or `:$`
Move to the end of the line	`$`
Move to the first non empty character of the line	`^`
Move to the beginning of the line	`0`
Move one page down	`Ctrl+f`
Move one page up	`Ctrl+b`
Move to the first line of the screen	`H`
Move to the middle of the screen	`M`
Move to the last line of the screen	`L`

6.2.3 Text editing

Expected result	Action	Comments
Insert before the cursor	`i`	Normal mode
Insert before the first non empty character of the line	`I`	Normal mode
Insert after the cursor	`a`	Normal mode
Insert at the end of the line	`A`	Normal mode
Insert a new line below	`o`	Normal mode
Insert a new line above	`O`	Normal mode
Replace everything after the cursor	`C`	Normal mode
Replace one character (and stay in normal mode)	`r`	Normal mode
Delete the character after the cursor (like the del. key)	`x`	Normal mode
Delete the character before the cursor (like the backspace key)	`X`	Normal mode
Delete the current line	`dd`	Normal mode
Copy the current line	`yy`	Normal mode
Paste after the cursor. If it's line, paste the line below.	`p`	Normal mode
Paste before the cursor. If it's line, paste the line above	`P`	Normal mode
Switch the case (upper/lower)	`~`	Visual mode
Move the text to the right (indent)	`>`	Visual mode
Move the text to the left	`<`	Visual mode
In visual mode, delete the selected text	`d`	Visual mode
In visual mode, replace the selected text	`c`	Visual mode
In visual mode, copy the selected text	`y`	Visual mode
Undo	`u`	Normal mode
Redo	`Ctrl+r`	Normal mode

6.2.4 Search and/or replace

Expected result	Action	Comments
Search	`/*toto`	Search the *toto* string starting at the current cursor position
Next	`n`	Go to the next search result
Previous	`N`	Go to the previous search result
Replace on the current line (once)	`:s/toto/titi`	Replace toto by titi on the current line (once)
Replace on the current line (multiple)	`:s/toto/titi/g`	Replace toto by titi on the current line (for all occurences of toto)
Replace on all the lines (once)	`:%s/toto/titi`	Replace toto by titi on all the lines of the file (once per line)
Replace on all the lines (multiple)	`:%s/toto/titi/g`	Replace toto by titi on all the lines of the file (for all occurences of toto)
Replace on the current line, case insensitive (once)	`:s/toto/titi/i`	Replace toto by titi on the current line, case insensitive (once)
Replace on the current line, case insensitive (multiple)	`:s/toto/titi/gi`	Replace toto by titi on the current line, case insensitive (for all occurences of toto)

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search