
Vim pour les humains

Version 2.0

Vincent Jousse

mai 24, 2024

Table des matières

1 Livre payant, mais prix libre	1
1.1 Prix libre	1
1.2 Licence libre	1
2 Introduction	3
2.1 Et Neovim ?	4
2.2 Pour qui ?	4
2.3 Ce que vous apprendrez (entre autres choses)	5
2.4 Ce que vous n'apprendrez pas (entre autres choses)	5
2.5 Le plus dur, c'est de commencer	5
3 Rendre Vim utilisable	7
3.1 Préambule indispensable : le mode insertion	9
3.2 Les modes : d'où Vim tire sa puissance	11
3.3 La configuration par défaut : indispensable	13
3.4 Que la couleur soit !	15
3.5 L'explorateur de fichiers : notre premier plugin	19
3.5.1 Gestionnaire de plugins : vim-plug	19
3.5.2 Explorateur de fichiers : vim-fern	20
3.6 Nous voilà fin prêts	23
4 L'outil de manipulation de texte rêvé	25
4.1 Se déplacer par l'exemple : Essayer de copier / coller	25
4.1.1 Préambule	25
4.1.2 Se passer de la souris	26
4.2 Se passer des touches directionnelles	28
4.2.1 Se déplacer sans les touches directionnelles	29
4.3 Se passer de la touche Échap	31
4.4 Combiner des touches/déplacements	32
4.5 Rechercher / Se déplacer rapidement	33
4.5.1 Sauts de page	33
4.5.2 Les marqueurs	34
4.5.3 La recherche	34
4.6 Le mode visuel	34
4.7 À vous de jouer	35
5 Les plugins indispensables	37
5.1 Naviguer sur le disque et entre les fichiers : <i>Lusty Explorer</i>	37
5.2 Recherche de fichiers, de chaînes de caractères et d'un peu tout : <i>fzf</i>	40
5.2.1 Installation de <i>fzf</i>	40
5.2.2 Recherche de fichiers par nom	40
5.2.3 Recherche de chaînes de caractères dans les fichiers	41
5.2.4 Recherche dans les noms de buffers	42
5.3 Les plugins avancés	43
6 Pense-bête et exemples	45
6.1 Questions / réponses	45

6.1.1	Comment quitter <i>Vim</i> ?	45
6.1.2	Comment sauvegarder sous ?	45
6.1.3	Comment copier/couper coller ?	46
6.1.4	Comment créer un nouveau fichier ?	46
6.1.5	Annuler / Refaire	46
6.2	Pense-bête	46
6.2.1	Fichiers	46
6.2.2	Déplacements	47
6.2.3	Édition de texte	48
6.2.4	Chercher et/ou remplacer	48

Chapitre 1

Livre payant, mais prix libre

1.1 Prix libre

Qu'est-ce que ça veut dire au juste ? Je suis persuadé que la culture et l'éducation devraient être accessibles à tous. Forcer les personnes à payer un prix que je décide à l'avance placerait forcément une barrière à la lecture du livre. C'est pourquoi je vous laisse choisir votre prix.

Et que se passe-t-il si vous n'avez pas quelques euros à donner contre le travail que j'ai fourni en rédigeant ce livre ? Et bien téléchargez-le gratuitement, partagez-le et promettez-moi d'en faire bon usage, ça me fera plaisir !

Quoique vous donnez, quoique vous fassiez pour partager ce savoir à d'autres personnes sera une motivation de plus pour moi à écrire d'autres livre dans le même style.

Si vous donnez en euros, 20% des gains seront reversés à l'association de promotion du libre FramaSoft.

Vous pouvez donner via le site officiel <https://vimebook.com>.

Merci, et amusez-vous bien !

1.2 Licence libre

En plus de choisir ce que vous donnez, vous pourrez faire ce que vous voulez du livre. Le mettre à télécharger sur votre site, le filer à des potes, le diffuser sur les torrents, bref, ce que vous voulez. J'aime à croire que ce livre ne m'appartient pas : je n'aurais jamais pu le créer sans le travail de tous ceux qui ont contribué à Vim.

Cette œuvre est mise à disposition selon les termes de la [Licence Creative Commons Attribution 4.0 International](#).



Chapitre 2

Introduction

Lorsque le besoin d'écrire ou de coder se fait se sentir, le choix d'un éditeur de texte est primordial. Il en existe énormément sur le « marché », mais peu d'entre eux peuvent se targuer d'environ 40 ans d'existence. C'est le cas d'*Emacs* (<http://www.gnu.org/software/emacs/>), de *Vi* et de son « successeur amélioré » *Vim* (<http://www.vim.org/>). Ils ont été créés dans les années 70 et sont toujours très utilisés actuellement. Comme vous avez sans doute pu le voir, ce n'est pas grâce à la beauté de leur site internet ou à l'efficacité de leur communication (même si je dois avouer que le site d'*Emacs* a fait des efforts depuis la première version de ce livre). Voici quelques **raisons de leur succès** :

Ils s'apprennent pour la vie

Ils s'apprennent une fois et s'utilisent pour toujours. Dans un monde où les technologies/langages changent tout le temps, c'est une aubaine de pouvoir investir sur du long terme.

Ils sont utilisables partout

Ils sont disponibles sur toutes les plateformes possibles et imaginables (et l'ont toujours été).

Ils augmentent votre vitesse d'édition de texte

Grâce à leurs particularités (notamment l'utilisation du clavier), ils permettent d'éditer du texte très rapidement.

Ce sont de vrais couteaux suisses

Ils permettent d'éditer tout et n'importe quoi. Quand vous changerez de langage de programmation, vous n'aurez pas à changer d'éditeur. À noter que ce livre a bien sûr été écrit avec *Vim*.

Et pourtant, ces éditeurs de texte restent difficiles à apprendre. Non pas qu'ils soient plus compliqués qu'autre chose, non pas que vous ne soyez pas à la hauteur, mais plutôt à cause d'un manque de pédagogie des différentes documentations.

Ce livre a pour but de pallier ce manque en vous guidant tout au long de votre découverte de *Vim*. Je laisse *Emacs* à ceux qui savent (pour un bref comparatif c'est ici : https://fr.wikipedia.org/wiki/-Guerre_d%C3%A9diteurs. Les goûts et les couleurs ...). Il ne prétend pas être un guide exhaustif, vous pouvez essayer *A Byte of vim* (en anglais) pour celà : <https://vim.swaroopch.com/>. En revanche, il prétend diminuer la marche à franchir pour s'habituer à *Vim*. À mon sens, le plus compliqué avec *Vim*, c'est de ne pas se décourager de suite et de trouver une méthode qui vous permette de l'apprendre au fur et à mesure. Nous avons tous un travail à effectuer au quotidien, et perdre toute sa productivité à cause d'un changement d'éditeur de texte n'est pas envisageable.

Vous trouverez beaucoup de personnes pour vous dire « mais tu n'as qu'à t'y mettre pour de bon », « tu verras après ça va aller mieux », certes, mais vous devez toujours être productif au jour le jour, ça ne change rien au problème. L'approche de ce livre est la suivante :

- Disposer d'un *Vim* un temps soit peu moderne : coloration syntaxique et jolies couleurs.
- Pouvoir utiliser *Vim* comme n'importe quel autre éditeur de texte : éditer facilement du code et naviguer entre les fichiers en utilisant la souris.
- Apprendre des raccourcis clavier et se passer de la souris au fur et à mesure.
- Installer un *best-of* des plugins pour commencer à tirer partie de la puissance de *Vim*.

À partir du point numéro 2, vous pourrez déjà utiliser *Vim* au quotidien sans perdre énormément de productivité. Et c'est là que la différence se fait : si vous pouvez intégrer *Vim* dans votre quotidien c'est gagné. Vous aurez ensuite toute votre vie pour connaître tous les raccourcis clavier.

Vous aussi vous en avez marre d'attendre la release de TextMate 2 (à noter que depuis l'écriture de ce livre, le code de TextMate 2 a été publié sous licence GPL : <https://github.com/textmate/textmate>) ? D'essayer le n-ième éditeur à la mode et de devoir tout réapprendre et ce jusqu'à la prochaine mode ? De devoir changer d'éditeur quand vous passez de votre Mac, à votre Windows, à votre Linux ? De ne pas pouvoir utiliser VSCode sur votre serveur ou de le trouver trop lourd ? Alors vous aussi, rejoignez la communauté des gens heureux de leur éditeur de texte. **Le changement, c'est maintenant !**

2.1 Et Neovim ?

Petit aparté au sujet de *Neovim* <https://neovim.io/> (si vous ne savez pas ce que c'est, vous pouvez sauter cette partie). J'ai pris le parti de me concentrer uniquement sur *Vim* dans ce livre afin de ne pas frustrer les personnes ne pouvant utiliser que *Vim*.

Si vous êtes un utilisateur de *Neovim*, tout ce qui est dans ce livre reste valable, *Neovim* étant compatible avec *Vim*. Qui plus est, les modes, les manipulations de texte, et tout ce qui ne concerne pas les plugins est commun à *Vim* et à *Neovim* : les apprendre pour *Vim* ou pour *Neovim* ne fait aucune différence. Vous pouvez d'ailleurs suivre ce livre en utilisant *Neovim* à la place de *Vim* sans aucun souci.

La particularité de *Neovim* réside dans le fait qu'il est plus activement maintenu que *Vim* et entre autres qu'il utilise le langage de programmation **Lua** pour la gestion des plugins. Cela a pour conséquence que les plugins écrits pour *Neovim* ne sont pas compatibles avec *Vim* (l'inverse n'étant pas vrai, les plugins écrits pour *Vim* sont compatibles avec *Neovim*). Une partie de la communauté de *Vim* est passée à *Neovim* et les plugins pour *Neovim* fleurissent bien plus vite que ceux pour *Vim*. Pour le but de ce livre, cela ne fera pas de différence et vous pouvez le suivre sans aucun problème avec *Vim* ou *Neovim*. Je ferai sûrement d'autres contenus spécifiques à *Neovim* dans le futur.

2.2 Pour qui ?

Toute personne étant amenée à produire du texte (code, livre, rapports, présentations, ...) de manière régulière. Les développeurs sont bien sûr une cible privilégiée, mais pas uniquement.

Par exemple vous êtes :

Étudiant

Si vous voulez faire bien sur un CV, c'est un must. C'est en effet un gage de sérieux de voir qu'un étudiant a pris sur son temps personnel pour apprendre *Vim*. De plus, vous aurez un outil unique pour écrire tout ce que vous avez à écrire (et que vous pourrez réutiliser tout au long de votre carrière) : vos rapports en LaTeX, vos présentations, votre code (si vous avez besoin d'OpenOffice ou de Word pour écrire vos rapports, il est temps de changer d'outil et d'utiliser LaTeX, Markdown ou reStructuredText).

Petit conseil d'ami, pour vos présentations, n'hésitez pas à utiliser un outil comme *impress.js* : <https://github.com/impress/impress.js>. Basé sur du HTML/JS/CSS, je vous le recommande grandement pour des présentations originales et basées sur des technologies non propriétaires. Il existe aussi *reveal.js* (<https://lab.hakim.se/reveal-js/>) et son éditeur en ligne *slid.es* (<https://slid.es/>).

Enseignant

Il est temps de montrer l'exemple et d'apprendre à vos étudiants à bien utiliser un des outils qui leur servira à vie, bien plus qu'un quelconque langage de programmation.

Codeur

Investir dans votre outil de tous les jours est indispensable. Quitte à apprendre des raccourcis claviers, autant le faire de manière utile. Si cet investissement est encore rentable dans 10 ans, c'est ce que l'on pourrait appeler l'investissement parfait, c'est *Vim*.

Administrateur système Unix

Si vous utilisez *Emacs* vous êtes pardonné. Si vous utilisez nano/pico je ne peux plus rien pour vous, sinon il est grand temps de s'y mettre les gars. Administrer un système Unix à distance est un des cas d'utilisation parfait pour *Vim* (un éditeur de texte surpassant ne nécessitant pas d'interface graphique).

Écrivain

Si vous écrivez en markdown/reStructuredText/WikiMarkup ou en LaTeX, *Vim* vous fera gagner beaucoup de temps. Vous ne pourrez plus repasser à un autre éditeur, ou vous voudrez le « Vimifier » à tout prix.

Faites moi confiance, je suis passé et repassé par ces 5 rôles, mon meilleur investissement a toujours été *Vim*, et de loin.

2.3 Ce que vous apprenez (entre autres choses)

- Comment utiliser *Vim* comme un éditeur « normal » d'abord (vous savez, ceux qui permettent d'ouvrir des fichiers, de cliquer avec la souris, qui ont une coloration syntaxique ...). En somme, la démystification de *Vim* qui vous permettra d'aller plus loin.
- Comment passer de l'édition de texte classique à la puissance de *Vim*, petit à petit (c'est là que l'addiction commence).
- Comment vous passer de la souris et pourquoi c'est la meilleure chose qu'il puisse vous arriver quand vous programmez/tapez du texte.
- Comment vous pouvez facilement déduire les raccourcis claviers avec quelques règles simples.

Si je devais le résumer en une phrase : puisque vous vous considérez comme **un artiste, passez du temps à apprendre** comment utiliser l'outil qui vous permet de vous exprimer, une bonne fois pour toute.

2.4 Ce que vous n'apprenez pas (entre autres choses)

- Vous n'apprenez pas comment installer/configurer *Vim* pour Windows. Pas que ce ne soit pas faisable, mais je n'ai que très peu de connaissances de Windows. Ça viendra peut-être, mais pas tout de suite. On couvrira ici Linux/Unix (et par extension Mac OS X).
- Vous n'apprenez pas comment utiliser *Vi* (notez l'absence du « m »). Je vais vous apprendre à être productif pour coder/produire du texte avec *Vim*, pas à faire le beau devant les copains avec *Vi* (*Vim* est suffisant pour cela de toute façon). Pour ceux qui ne suivent pas, *Vi* est « l'ancêtre de *Vim* (qui veut dire *Vi - IMproved, Vi amélioré*) » et est installé par défaut sur tous les Unix (même sur votre Mac OS X).
- Vous n'apprenez pas à connaitre les entrailles de *Vim* par cœur : ce n'est pas une référence, mais un guide utile et pragmatique.
- Vous n'apprenez pas comment modifier votre *Vim* parce que vous préférez le rouge au bleu : je vous ferai utiliser le thème *Solarized* (<http://ethanschoonover.com/solarized>), il est parfait pour travailler.

2.5 Le plus dur, c'est de commencer

Alors, prêt pour l'aventure ? Prêt à sacrifier une heure pour débuter avec *Vim*, une semaine pour devenir familier avec la bête, et le reste de votre vie pour vous féliciter de votre choix ? Alors c'est parti ! Enfin presque, il faut qu'on parle avant.

Vim fait partie de ces outils avec lesquels vous allez galérer au début. Le but de ce guide est de vous mettre le pied à l'étrier et de diminuer la hauteur de la marche à franchir. Mais soyez conscients que vous mettre à *Vim* va vous demander de la volonté et quelques efforts. Comme on dit souvent, on n'a rien sans rien. Voici la méthode que je vous recommande pour apprivoiser la bête :

- Essayez de faire entrer *Vim* dans vos habitudes. Suivez le premier chapitre de ce guide jusqu'à la partie concernant l'explorateur de fichiers utilisable à la souris *vim-fenn*. Ensuite, vous pourrez utiliser *Vim* comme un Notepad++ ou un TextMate ou un Sublime Text. Vous n'utiliserez que 1% des capacités de *Vim* mais peu importe. Ce qui est important, c'est de le faire entrer dans votre quotidien.

- Gardez une feuille avec les principaux raccourcis imprimée à côté de vous. Le but ce n'est pas de les apprendre par cœur, mais c'est de les avoir à portée de main quand vous vous demanderez « mais il y a certainement une façon plus efficace de faire cela ».
- Gardez la foi. Au début vous resterez un sceptique quant à l'utilité de tout réapprendre avec *Vim*. Et puis un jour vous aurez un déclic et vous vous demanderez pourquoi tous vos logiciels ne peuvent pas se contrôler avec les commandes de *Vim*.
- Gardez à l'esprit que c'est un investissement pour vos 20 prochaines années, et c'est bien connu, un investissement ce n'est pas complètement rentable de suite.

Trêve de bavardage, passons aux choses sérieuses. Go go go !

Chapitre 3

Rendre *Vim* utilisable

Ça peut paraître étonnant comme approche, mais c'est pour moi la première chose à faire : rendre *Vim* utilisable par un humain lambda. Si tout le monde semble s'accorder sur le fait que *Vim* est un **éditeur très puissant**, tout le monde pourra aussi s'accorder sur le fait que de base, il est totalement **imbitable**. Soyons honnête, sans une configuration par défaut minimale, utiliser *Vim* est **contre-productif**.

C'est à mon avis le premier obstacle à surmonter avant toute autre chose. C'est ce que les autres éditeurs « à la mode » comme VSCode, TextMate, Sublime Text, Notepad++ ou NetBeans proposent, c'est à dire un environnement à minima utilisable tel quel, même si l'on n'en exploite pas la totalité.

Voici donc ce qui manque à un *Vim* nu (et ce qui est, de mon point de vue, une **cause d'abandon pour beaucoup** d'entre vous) :

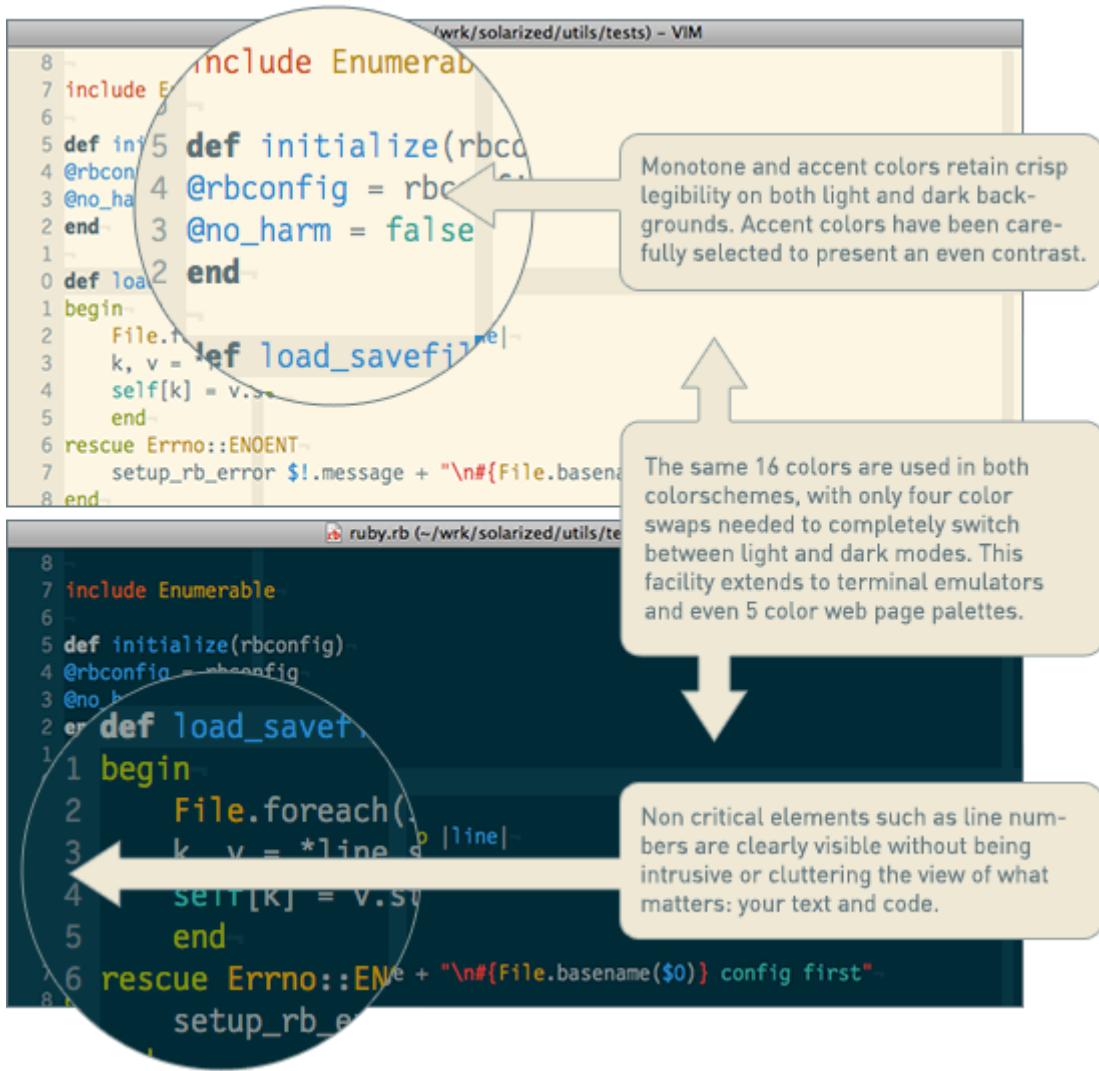
Configuration par défaut

Vim est configurable grâce à un fichier nommé `~/.vimrc`, qui est bien entendu vide par défaut.

La première étape va être d'écrire ou de se procurer un fichier `~/.vimrc` avec une configuration minimale.

Coloration syntaxique

De base, *Vim* est tout blanc et tout moche. On va utiliser le thème *Solarized* (<http://ethanschoonover.com/solarized>). Si votre but est d'être efficace, c'est le meilleur thème disponible actuellement (tout éditeur de texte confondu). La belle image qui suit vous donne une idée des deux looks disponibles (clair ou sombre). Pour ma part j'utilise le thème sombre.



Explorateur de fichiers

Si vous utilisez *Vim* avec une interface graphique (ce qui est le cas de 99% d'entre vous je suppose) vous avez par défaut un menu **Fichier** vous permettant d'ouvrir un fichier. C'est certes un bon début, mais avoir à disposition un explorateur de projet à la NetBeans ou à la TextMate peut s'avérer très pratique. Pour obtenir le même comportement, nous utiliserons *vim-fern* (<https://github.com/lambdalisue/vim-fern>). À savoir qu'à la fin de ce guide, vous n'aurez plus besoin de la souris (et donc des menus et autres boutons).

Ce chapitre est indispensable si vous n'avez que peu d'expérience (voire pas du tout) avec *Vim*. À la fin de ce chapitre, vous aurez un *Vim* dont vous pourrez commencer à vous servir pour vos tâches de tous les jours. Cela devrait être suffisant pour vous permettre d'apprendre le reste petit à petit. Car il n'y a pas de secret, il vous faudra pratiquer pour apprendre *Vim*. Autant commencer de suite et le moins douloureusement possible.

En revanche, si vous êtes déjà familier avec *Vim* et n'utilisez déjà plus la souris, vous pouvez sagement sauter ce chapitre (soyez sûr tout de même de donner sa chance au thème *Solarized*).

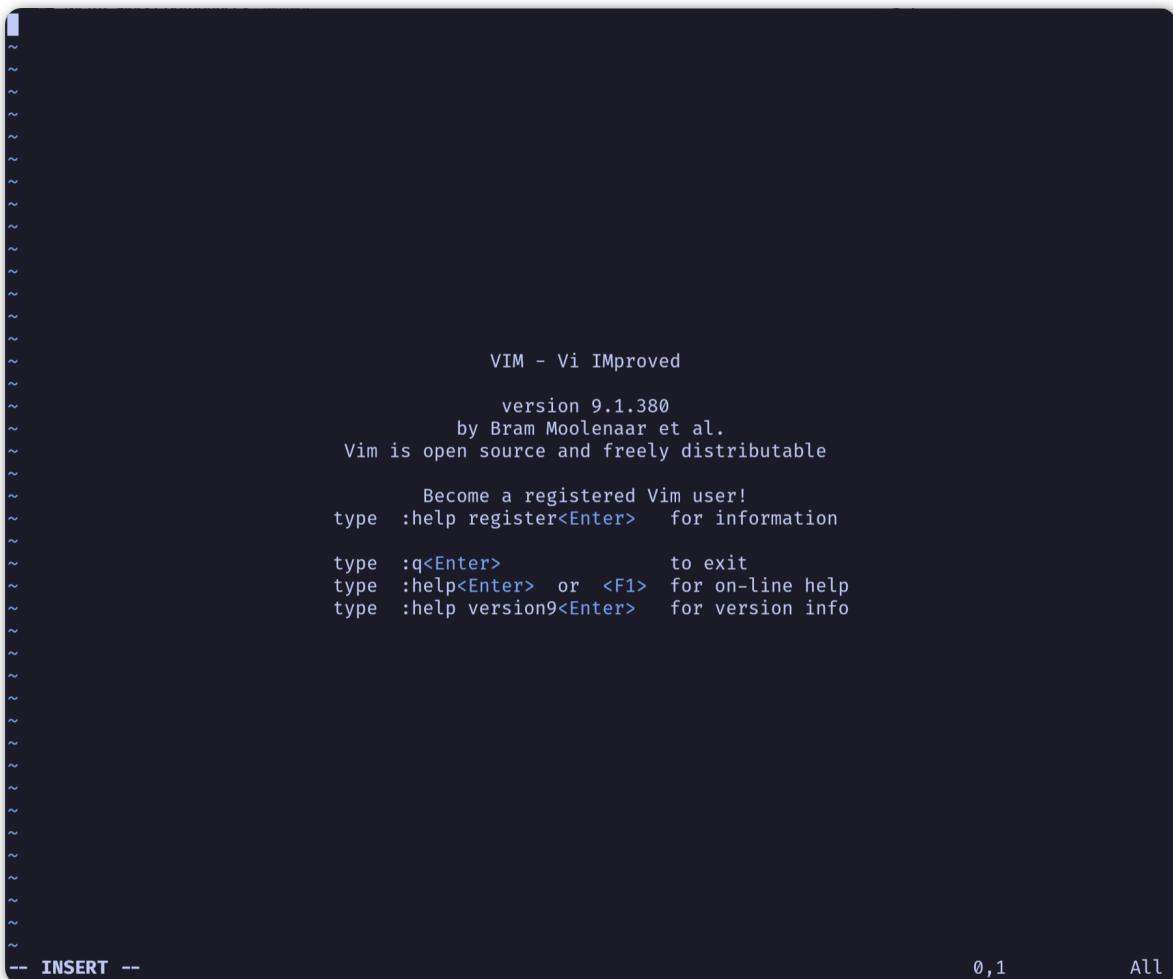
3.1 Préambule indispensable : le mode insertion

Prenons le pari de créer le fichier `~/.vimrc` avec *Vim* lui-même. Comme je vous le disais, lus tôt vous commencerez, mieux ce sera. Vous devrez certainement commencer par installer une version de *Vim*. Si vous utilisez un Mac, essayez MacVim (<https://macvim.org/>) sans aucune hésitation. Si vous utilisez GNU/Linux ou tout autre système « Unix » vous devriez sûrement avoir la commande *vim* à disposition dans votre terminal. Vous pouvez aussi utiliser l'interface graphique *gVim* qui devrait être facilement installable grâce à votre gestionnaire de logiciels, pour Ubuntu le paquet correspondant est *vim-gnome*. Faites attention à bien installer la version **complète** de vim, notamment avec le support de Ruby et de Lua dont nous aurons besoin par la suite. Pour Ubuntu, le paquet sembler s'appeller *vim*. Pour Mac OS X, la question ne se pose pas, MacVim inclut déjà tout ce qu'il faut. Pour Windows, il semblerait y avoir une version disponible sur le site officiel de *Vim* (<http://www.vim.org/download.php>), mais je ne l'ai pas testée.

Pour ma part, j'utilise *vim* directement en ligne de commande, sous Archlinux, dans un terminal *kitty* avec la police Nerd Fonts *FiraCode Nerd Font*. C'est avec cette configuration que les capture d'écran de ce livre sont réalisées.

Au lancement de *Vim*, vous devriez avoir un texte d'accueil vous encourageant à aider les pauvres enfants en Ouganda. Ce texte disparaitra dès que nous allons saisir des caractères dans *Vim*. Nous allons commencer par entrer un commentaire dans l'en-tête du fichier pour y mentionner notre nom. Pour pouvoir entrer du texte appuyez sur la touche `i`. Vous devriez avoir *une page* qui ressemble plus ou moins à la figure ci-dessous, notez bien le `--INSERT--` en bas à gauche qui nous indique que nous sommes en mode insertion (le mode où nous pouvons saisir du texte). Pour information, le thème de mon terminal est un thème sombre, il est donc possible que pour l'instant, les couleurs de votre *Vim* soient différentes.

À l'écriture de ces lignes, la version de *Vim* que j'utilise est la `9.1.380`.



The screenshot shows the Vim editor window. The main area displays the Vim startup message:

```
VIM - Vi IMproved
version 9.1.380
by Bram Moolenaar et al.
Vim is open source and freely distributable

      Become a registered Vim user!
type :help register<Enter>   for information

type :q<Enter>          to exit
type :help<Enter> or <F1> for on-line help
type :help version9<Enter> for version info
```

At the bottom left, there is a status bar with the text "-- INSERT --". At the bottom right, there are status indicators: "0,1" and "All".

À noter : si vous ne savez pas trop ce que vous avez fait et que *Vim* vous affiche des trucs en rouge en bas à gauche ou ne semble pas réagir comme il faut quand vous appuyez sur la touche `i`, appuyez plusieurs fois sur la touche `Esc` (Échap), cela devrait vous remettre au mode par défaut de *Vim*, le mode *Normal*.

Vous devriez maintenant pouvoir entrer *le commentaire ci-dessous* (je vous laisse évidemment changer mon nom par le votre 😊).

" VIM Configuration - Vincent Jousse "

Vous aurez remarqué que les commentaires en *VimL* (le langage de configuration de *Vim*) commencent par un « . Appuyez ensuite sur la touche `Esc` (Échap) pour revenir au mode par défaut (le mode normal) de *Vim*. Et voilà le travail, comme vous pouvez le voir sur *la copie d'écran de Vim avec votre joli commentaire*.

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

The screenshot shows a dark-themed Vim editor window. At the top left, it displays the file name "VIM Configuration - Vincent Jousse". The main area of the window is filled with numerous lines of code, represented by a series of tilde (~) characters. At the bottom right corner of the window, there is a status bar with the number "1,36" and the word "All".

Tout ça pour ça me direz-vous, et vous avez bien raison. Et encore, on n'a même pas encore vu comment le sauvegarder. Mais tout cela a une logique que je vais vous expliquer. L'avantage de *Vim* est qu'il est généralement logique. Quand vous avez compris la logique, tout vous semble limpide et tomber sous le sens.

Par défaut, *Vim* est lancé dans un mode que l'on appelle le mode *Normal*. C'est à dire que ce mode n'est pas fait pour écrire du texte (ça, ça sera le mode *Insertion*) mais juste pour se déplacer et manipuler du texte. C'est la présence de ces deux différents modes (il y en a d'autres mais ce n'est pas le sujet pour l'instant) qui fait toute la puissance de *Vim*. Il vous faudra un certain temps pour vous rendre compte de cette puissance par vous-même, alors faites-moi juste confiance pour l'instant.

Si vous vous demandez pourquoi ces modes, pourquoi on semble se compliquer la vie (on se la simplifie en fait) et en quel honneur, dans le mode par défaut, il n'est même pas possible d'insérer du texte, lisez attentivement la section qui suit.

3.2 Les modes : d'où *Vim* tire sa puissance

Je pense que nous serons tous à peu près d'accord sur le fait que si vous souhaitez apprendre à utiliser *Vim*, c'est pour gagner en efficacité pour la saisie/manipulation de texte/code. Pour gagner en efficacité lorsque l'on tape du code il n'y a pas 36 solutions. Il n'y en a qu'une en fait : il faut bouger le moins possible les mains (voire pas du tout), et ne bouger que les doigts.

Pour ce faire bien sûr, vous oubliez tout d'abord l'utilisation de la souris. En plus d'être lent, le mouvement clavier -> souris puis souris -> clavier est très mauvais pour vos articulations. Il est souvent à l'origine de troubles musculosquelettiques. Vous êtes peut-être jeune et n'avez pas encore eu ce type de soucis. Mais croyez moi, ça vient beaucoup plus vite qu'on ne le croit. Si vous passez votre journée sur un

ordinateur, ne négligez pas ces facteurs, vous le regretterez un jour. D'après *Wikipedia*, c'est le type de maladie professionnelle la plus courante à l'heure actuelle (https://fr.wikipedia.org/wiki/Troubles_musculosquelettiques).

Vous oubliez aussi le mouvement de votre main droite vers les touches directionnelles gauche/droite/-bas/haut. C'est une perte de temps et c'est totalement inutile avec *Vim*.

Qu'est-ce que vous avez le droit de faire dans tout ça ? Pas grand chose, si ce n'est garder vos mains sur la position de repos comme le montre *l'image ci-dessous avec la position idéale des mains*.

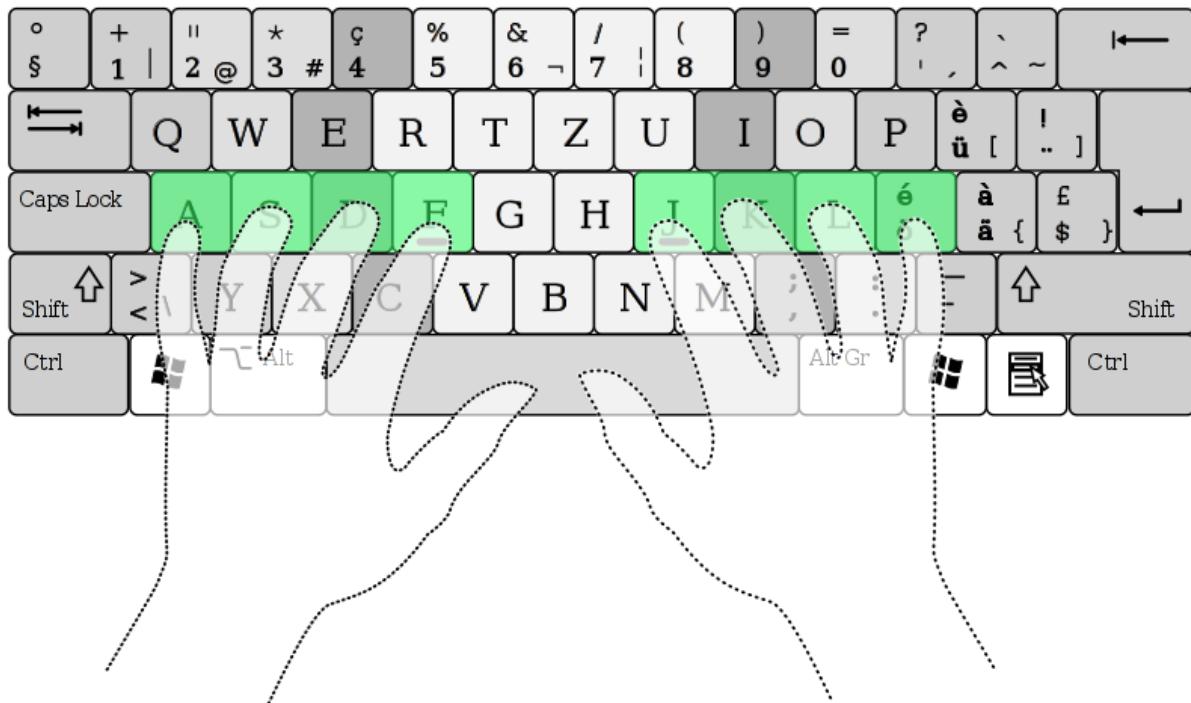


FIG. 1 – Position de repos, clavier QWERTY.

Illustration par Cy21 - CC-BY-SA-3.0 (<http://www.creativecommons.org/licenses/by-sa/3.0>) ou GFDL (<http://www.gnu.org/copyleft/fdl.html>, via Wikimedia Commons <http://commons.wikimedia.org/wiki/File:Typing-home-keys-hand-position.svg>

Vous trouverez d'ailleurs sur la plupart des claviers des marques sur les touches F et J, c'est pour vous donner un repère tactile de la position où doivent se trouver vos index dans la position de repos.

Ce parti pris (bouger le moins possible les mains du clavier) justifie à lui seul la présence d'un mode *normal* et d'un mode *insertion* dans *Vim*. En passant de l'un à l'autre, les touches sous vos doigts serviront tantôt à vous déplacer et à réaliser des opérations sur le texte : copier/coller, macros, ... (c'est le mode *normal*), tantôt à sélectionner (c'est le mode *visuel*) et tantôt à insérer du texte (c'est le mode *insertion*). Tout cela bien sûr en évitant l'utilisation de combinaisons de touches du style *Ctrl + touche* qui ne sont généralement pas bonnes pour vos doigts (*Emacs* si tu nous lis, je te salue).

Par défaut, on passe du mode *insertion* au mode *normal* en appuyant sur la touche **Esc (Échap)**, mais c'est une des premières choses que l'on changera : la touche **Esc (Échap)** est bien trop loin sur les claviers actuels.

Pour passer du mode *normal* au mode *insertion*, on peut par exemple appuyer sur la touche **i**. On apprendra par la suite qu'il existe d'autres moyens de faire. Par exemple pour rentrer en mode *insertion* tout en créant une nouvelle ligne en dessous de la ligne courante (peu importe où se trouve votre curseur sur la ligne), on utilisera la touche **o** en mode *normal*.

J'y reviendrai plus tard dans « *Se déplacer par l'exemple : Essayer de copier / coller* » mais si vous n'êtes pas prêt, à terme, à ne plus utiliser votre souris et les flèches directionnelles pour éditer du texte,

je vous recommanderais presque d'arrêter votre apprentissage maintenant. C'est aussi simple que cela. *Vim* révèle tout sa puissance quand il est utilisé sans souris et en bougeant le moins possible les mains.

Si vous voulez pousser la démarche encore plus loin, vous pouvez aussi vous procurer un clavier orthogonal *TypeMatrix* (<http://www.typematrix.com/>). C'est ce que j'utilise personnellement, et mes doigts m'en remercient tous les jours.

L'ultime changement serait d'utiliser une disposition de clavier encore plus efficace comme le *bépo* pour quasi doubler sa vitesse de frappe au clavier. Pour les plus curieux d'entre vous, j'explique la démarche sur mon blog : <https://vincent.jousse.org/blog/fr/comment-doubler-sa-vitesse-de-frappe-au-clavier/>

3.3 La configuration par défaut : indispensable

Passons aux choses sérieuses, c'est-à-dire comment rendre *Vim* un tant soit peu utilisable. Nous allons donc éditer le fichier de configuration par défaut `~/.vimrc` en y plaçant des valeurs que toute personne normalement constituée souhaiterait y voir figurer.

Ce fichier doit se trouver dans votre répertoire d'accueil. `/home/votre_user/.vimrc` sous Linux, `/Users/-votre_user/.vimrc` sous Mac OS X ou plus généralement `~/.vimrc`. Sous Windows vous pouvez créer un fichier nommé `_vimrc` qui doit se situer dans votre répertoire `%HOME%` qui change en fonction de votre version de Windows. C'est généralement le répertoire jute « au-dessus » de votre répertoire *Mes Documents*. Plus d'infos sur Wikipedia http://en.wikipedia.org/wiki/Home_directory#Default_Home_Directory_per_Operating_System.

J'ai commenté chacune des lignes du fichier directement dans le code. Rien de sorcier ici, on se demande juste pourquoi tout cela n'est pas inclus par défaut.

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

" -- Affichage
set title           " Met à jour le titre de votre fenêtre ou de
                      " votre terminal
set number          " Affiche le numéro 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 sensibilité à la casse
set incsearch         " Surligne les résultats de recherche pendant la
                      " saisie
set hlsearch          " Surligne les résultats de recherche

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

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

" Cache les fichiers lors de l'ouverture d'autres fichiers
set hidden
```

Pour ceux qui ont fait un copier/coller, il ne vous reste plus qu'à sauvegarder votre fichier nouvellement

créé. Nous voulons le placer à la racine de votre compte utilisateur, c'est à dire l'enregistrer sous `~/.vimrc`. Sous Mac OS X et Linux, `~` désigne le répertoire d'accueil de l'utilisateur courant. Attention, les fichiers commençant par un `.` sont des fichiers cachés sous Linux et Mac OS X, ne vous étonnez donc pas de ne pas le voir par défaut dans votre navigateur de fichiers.

Pour le sauvegarder avec *Vim*, il vous suffira, après avoir appuyé sur la touche `Esc` (`Échap`) pour repasser en mode *Normal*, de taper `:w ~/.vimrc`. Pour sauvegarder vos prochaines modifications tapez en mode *Normal* `:w`. Pour sauvegarder et quitter `:wq ~/.vimrc`. Pour quitter `:q` et pour quitter sans sauvegarder (forcer à quitter) `:q!`.

J'ai mis en ligne ce fichier de configuration directement sur *Github*. Vous pouvez le télécharger ou le copier directement à partir d'ici : <http://vimebook.com/link/v2/fr/firstconfig>.

Voici à quoi devrait ressembler *Vim* après votre première configuration.

```

1 " VIM Configuration - Vincent Jousse
2 " Annule la compatibilité avec l'ancêtre Vi : totalement indispensable
3 set nocompatible
4 " -- Affichage
5 set title
6 " Met à jour le titre de votre fenêtre ou de
7 " votre terminal
8 set number" Affiche le numéro des lignes
9 set ruler" Affiche la position actuelle du curseur
10 set wrap" Affiche les lignes trop longues sur plusieurs
11 " lignes
12 set scrolloff=3
13 " Affiche un minimum de 3 lignes autour du curseur
14 " (pour le scroll)
15 " -- Recherche
16 set ignorecase" Ignore la casse lors d'une recherche
17 set smartcase" Si une recherche contient une majuscule,
18 " re-active la sensibilité à la casse
19 set incsearch
20 " Surligne les résultats de recherche pendant la
21 " saisie
22 set hlsearch
23 " Surligne les résultats de recherche
24 " -- Beep
25 set visualbell" Empêche Vim de beeper
26 set noerrorbells" Empêche Vim de beeper
27 " Active le comportement 'habituel' de la touche retour en arrière
28 set backspace=indent,eol,start
29 " Cache les fichiers lors de l'ouverture d'autres fichiers
30 set hidden
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~
```

30,1

All

FIG. 2 – Vim après votre première configuration.

Notez l'ajout des numéros de ligne sur la gauche.

Bon c'est bien beau tout ça mais ça manque un peu de couleurs. Au suivant !

3.4 Que la couleur soit !

Tout d'abord il faut commencer par activer la coloration syntaxique du code dans le fichier de configuration. Ajoutez ces lignes à là fin de votre fichier de configuration `~/.vimrc`.

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

Vous devriez avoir un *Vim* qui ressemble à celui de la figure ci-dessous.

```
1 "# VIM Configuration - Vincent Jousse
2 " Annule la compatibilité avec l'ancêtre Vi : totalement indispensable
3 set nocompatible
4 " -- Affichage
5 set title
6 " Met à jour le titre de votre fenêtre ou de
7 " votre terminal
8 set number" Affiche le numéro des lignes
9 set ruler" Affiche la position actuelle du curseur
10 set wrap" Affiche les lignes trop longues sur plusieurs
11 " lignes
12 set scrolloff=3
13 " Affiche un minimum de 3 lignes autour du curseur
14 " (pour le scroll)
15 " -- Recherche
16 set ignorecase" Ignore la casse lors d'une recherche
17 set smartcase" Si une recherche contient une majuscule,
18 " re-active la sensibilité à la casse
19 set incsearch
20 " Surligne les résultats de recherche pendant la
21 " saisie
22 set hlsearch
23 " Surligne les résultats de recherche
24 " -- Beep
25 set visualbell" Empêche Vim de beeper
26 set noerrorbells" Empêche Vim de beeper
27 " Active le comportement 'habituel' de la touche retour en arrière
28 set backspace=indent,eol,start
29 " Cache les fichiers lors de l'ouverture d'autres fichiers
30 set hidden
31 " Active la coloration syntaxique
32 syntax enable
33 " Active les comportements spécifiques aux types de fichiers comme
34 " la syntaxe et l'indentation
35 filetype on
36 filetype plugin on
37 filetype indent on
~
~
~
~
~

.vimrc" 37L, 1191B
```

1,1

All

FIG. 3 – Coloration syntaxique par défaut.

Pour l'instant, le plus facile pour que les modifications apportées à votre `~/.vimrc` soient prises en compte, c'est de le fermer et de le ré-ouvrir. Si vous voulez vraiment vous la jouer à la *Vim* de suite, en mode normal tapez `:so ~/.vimrc` ou `:so $MYVIMRC`.

`:so` étant un raccourci pour `:source`. C'est une bonne première étape, passons maintenant à l'utilisation d'un thème.

Les thèmes vont vous permettre de rendre votre *Vim* un peu moins austère en changeant généralement la couleur de fond ainsi que les couleurs par défaut pour le code. Comme je l'ai mentionné plus haut,

nous allons utiliser le thème *Solarized*¹ <http://ethanschoonover.com/solarized> (avec fond clair ou foncé, ça dépendra de vous).

<https://raw.githubusercontent.com/ericbn/vim-solarized/master/colors/solarized.vim>

Pour l'installer, commencez tout d'abord par créer un répertoire nommé `.vim` au même endroit que votre `~/.vimrc` (dans votre répertoire utilisateur donc). À noter que ce répertoire s'appelle *vimfiles* sous Windows. À chaque fois que je ferai référence au répertoire `.vim` ça sera en fait *vimfiles* pour les Windowsiens. Dans ce répertoire `.vim`, créez un sous-répertoire nommé *colors*. Téléchargez ensuite le fichier du thème *Solarized* <https://raw.githubusercontent.com/ericbn/vim-solarized/master/colors/solarized.vim> (c'est le même fichier pour les deux versions du thème) et copiez le dans le répertoire `vim/colors/` fraîchement créé.

Sous Linux vous pouvez faire tout ça via les commandes suivantes :

```
mkdir -p ~/.vim/colors  
wget -P ~/.vim/colors https://raw.githubusercontent.com/ericbn/vim-solarized/master/colors/solarized.vim
```

Votre répertoire `.vim` devrait ressembler à cela :

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

Activez ensuite le thème Solarized dans votre `~/.vimrc` comme le montre le code ci-dessous. :

```
" Utilise la version sombre de Solarized  
set background=dark  
" Active les couleurs 24-bits dans le terminal  
set termguicolors  
colorscheme solarized
```

Pour tester le thème clair, remplacez *dark* par *light* (au niveau de la définition de la propriété *background*).

Ci-dessous un aperçu des deux variantes (ma préférence allant à la variante sombre soit dit en re-passant).

Un bonus (si vous n'utilisez pas *Vim* directement dans votre terminal) serait de choisir une police de caractères qui vous convient un peu mieux. C'est bien sûr facultatif, mais je présume que certains d'entre vous sont des esthètes aguerris.

Si vous êtes sous Mac OS X je vous conseille la police *Monaco* qui est assez conviviale. Rajoutez les lignes suivantes à votre `~/.vimrc` pour l'utiliser :

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

Vous pouvez bien sûr changer le *h13* par *h12* si vous voulez une police plus petite (ou par *h14* si vous en voulez une plus grande).

Simon sous Linux j'utilise la police *DejaVu Sans Mono* que je trouve plutôt sympathique :

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

Vous pouvez là aussi bien sûr changer la taille de la police si vous le souhaitez. Pour avoir la liste des polices disponibles tapez en mode normal `:set guifont:*`.

Vous trouverez une version complète du fichier de configuration pour ce chapitre en ligne <http://vimebook.com/link/v2/fr/syntaxhlconfig>. Je ne m'attarderai pas plus sur les polices, c'est assez dépendant de votre système d'exploitation, et un peu moins de *Vim*.

1. À noter que nous utiliserons une version modernisée de *Solarized* pour vim et non l'originale disponible sur le site de l'auteur. Cette version plus récente va notamment lui permettre de fonctionner correctement sur les terminaux modernes. On l'installera à partir de ce fork <https://github.com/ericbn/vim-solarized>.

```

6 set title           " Met a jour le titre de votre fenetre ou de
7                                         " votre terminal
8 set number          " Affiche le numero des lignes
9 set ruler           " Affiche la position actuelle du curseur
10 set wrap            " Affiche les lignes trop longues sur plusieurs
11                                         " lignes
12
13 set scrolloff=3    " Affiche un minimum de 3 lignes autour du curseur
14                                         " (pour le scroll)
15
16 " -- Recherche
17 set ignorecase      " Ignore la casse lors d'une recherche
18 set smartcase        " Si une recherche contient une majuscule,
19                                         " re-active la sensibilite a la casse
20 set incsearch        " Surligne les resultats de recherche pendant la
21                                         " saisie
22 set hlsearch         " Surligne les resultats de recherche
23
24 " -- Beep
25 set visualbell       " Empeche Vim de beeper
26 set noerrorbells     " Empeche Vim de beeper
27
28 " Active le comportement 'habituel' de la touche retour en arriere
29 set backspace=indent,eol,start
30
31 " Cache les fichiers lors de l'ouverture d'autres fichiers
32 set hidden
33
34 " Active la coloration syntaxique
35 syntax enable
36
37 " Active les comportements specifiques aux types de fichiers comme
38 " la syntaxe et l'indentation
39 filetype on
40 filetype plugin on
41 filetype indent on
42
43
44 " Utilise la version sombre de Solarized
45 set background=dark
46 set termguicolors
47 colorscheme solarized

```

47,1

Bot

FIG. 4 – Le thème *Solarized* sombre.

```

6 set title           " Met a jour le titre de votre fenetre ou de
7                                         " votre terminal
8 set number          " Affiche le numero des lignes
9 set ruler           " Affiche la position actuelle du curseur
10 set wrap            " Affiche les lignes trop longues sur plusieurs
11                                         " lignes
12
13 set scrolloff=3    " Affiche un minimum de 3 lignes autour du curseur
14                                         " (pour le scroll)
15
16 " -- Recherche
17 set ignorecase      " Ignore la casse lors d'une recherche
18 set smartcase        " Si une recherche contient une majuscule,
19                                         " re-active la sensibilite a la casse
20 set incsearch        " Surligne les resultats de recherche pendant la
21                                         " saisie
22 set hlsearch         " Surligne les resultats de recherche
23
24 " -- Beep
25 set visualbell       " Empeche Vim de beeper
26 set noerrorbells     " Empeche Vim de beeper
27
28 " Active le comportement 'habituel' de la touche retour en arriere
29 set backspace=indent,eol,start
30
31 " Cache les fichiers lors de l'ouverture d'autres fichiers
32 set hidden
33
34 " Active la coloration syntaxique
35 syntax enable
36
37 " Active les comportements specifiques aux types de fichiers comme
38 " la syntaxe et l'indentation
39 filetype on
40 filetype plugin on
41 filetype indent on
42
43
44 " Utilise la version sombre de Solarized
45 set background=light
46 set termguicolors
47 colorscheme solarized

```

47,1

Bot

FIG. 5 – Le thème *Solarized* clair.

3.5 L'explorateur de fichiers : notre premier plugin

Nous y voilà, nous avons un *Vim* à peu près utilisable avec de jolies couleurs. Maintenant, il faudrait être capable d'ouvrir des fichiers, ça pourrait être pratique ! Ça va être une bonne occasion pour installer notre premier plugin. Nous allons procéder ici en deux étapes, tout d'abord installer un gestionnaire de plugins pour éviter que ça devienne trop le bazar dans vos plugins, puis installer le plugin adéquat pour explorer un répertoire de fichiers.

3.5.1 Gestionnaire de plugins : vim-plug

vim-plug est le genre de plugin typique que vous découvrez après avoir commencé à configurer votre *Vim* et qui génère ce type de réaction : « *Ah si j'avais su j'aurais directement commencé avec* ». Ça tombe bien, c'est ce que nous allons faire.

Tout d'abord, petite explication sur la manière d'installer et de configurer des plugins dans *Vim*. Ils s'installent en copiant les fichiers adéquats (la plupart du temps avec une extension en **.vim*) dans des sous-répertoires de votre répertoire de configuration *.vim*. On a déjà d'ailleurs commencé à y créer un sous-répertoire *colors* qui contient notre « plugin » de coloration *solarized*.

Le problème avec cette approche c'est que les différents plugins ne sont pas isolés (vous allez devoir copier leurs fichiers dans les différents sous-répertoires) et que vous allez donc vous retrouver avec des fichiers un peu partout sans savoir à qui ils appartiennent. Autant vous dire qu'une fois que vous voulez désinstaller ou mettre à jour un plugin, c'est vite l'enfer pour savoir quels sont ses fichiers.

C'est là que *vim-plug* arrive à la rescousse, il va vous permettre d'installer chaque plugin dans un sous-répertoire rien que pour lui. Voici ce que donnerait le répertoire *.vim* d'une installation fictive de *Vim* avant et après l'utilisation de *vim-plug*.

Code source 1 – .vim avant l'utilisation de *vim-plug*

```
.vim-
├── autoload
│   └── phpcomplete.vim
├── colors
│   └── solarized.vim
└── syntax
    ├── php.vim
    └── sql.vim
```

Code source 2 – .vim après l'utilisation de *vim-plug*

```
.vim
├── autoload
│   └── plug.vim
└── plugged
    ├── solarized
    │   └── colors
    │       └── solarized.vim
    ├── php
    │   ├── autoload
    │   │   └── phpcomplete.vim
    │   ├── syntax
    │   │   └── php.vim
    │   └── autoload
    └── sql
        └── syntax
            └── sql.vim
```

Certes la version avec *vim-plug* contient plus de sous-répertoires mais chaque plugin est isolé dans son propre répertoire. Croyez-moi sur parole, ce rangement va vous éviter bien des ennuis par la suite.

Commençons par installer *vim-plug*. Créez un répertoire nommé *autoload* dans votre répertoire *.vim* et copiez y *plug.vim* que vous pouvez télécharger ici : <https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim>. Pour les utilisateurs Unix, la commande qui suit permet de l'installer automatiquement :

```
curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
      https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

Il nous faut maintenant activer *vim-plug* dans notre `~/.vimrc` et le tour est joué. Nous placerons le code listé ci-dessous au début du fichier `~/.vimrc`, directement après la ligne `set nocompatible`. Il est impératif de placer le code **au début** de votre fichier `~/.vimrc` au risque que tout ne fonctionne pas comme souhaité.

```
" Activation de vim-plug
call plug#begin()

" Nous mettrons nos plugins ici

call plug#end()
```

Puisque charité bien ordonnée commence par soi-même, nous allons utiliser *vim-plug* pour gérer **solarized** au lieu de l'installer à la main comme nous l'avons fait précédemment. Commençons par supprimer le répertoire `colors` que nous avons créé précédemment où nous avions placé *solarized* :

```
# Suppression du répertoire colors
rm -rf ~/.vim/colors
```

Modifions ensuite notre fichier `~/.vimrc` pour y ajouter **solarized** comme plugin (*Vim* devrait se plaindre qu'il ne peut pas trouver le thème *solarized*, vous pouvez ignorer l'erreur, nous allons justement l'installer).

```
" Activation de vim-plug
call plug#begin()

" Nous mettrons nos plugins ici

" Installation de solarized
Plug 'ericbn/vim-solarized'

call plug#end()
```

Sauvegardez et quittez en utilisant en mode normal `:wq`. Relancez *Vim* puis, tapez `:PlugInstall` pour installer notre nouveau plugin (appuyez sur la touche `q` pour quitter la fenêtre d'installation). Au prochain chargement de *Vim*, vous devriez avoir retrouvé vos couleurs.

Voilà notre *Vim* est presque prêt pour le grand bain. Il vous reste une petite étape à franchir : disposer d'un moyen pratique pour explorer les fichiers d'un projet. C'est ici que *vim-fern* entre en lice.

3.5.2 Explorateur de fichiers : *vim-fern*

vim-fern est un plugin permettant d'afficher visuellement une arborescence de fichiers directement dans la partie gauche (par défaut) de votre *Vim*, à la *VSCode*, *Sublime Text* ou encore *Eclipse/NetBeans*. Ce plugin n'est pas essentiel si vous souhaitez tout contrôler au clavier (je ne l'utilise plus moi-même), mais est assez pratique lorsque l'on débute avec *Vim*.

@TODO : Vérifier si on va toujours utiliser LustExplorer L'alternative que nous verrons plus tard au chapitre *Les plugins indispensables* est d'utiliser un plugin comme *LeaderF* pour trouver des fichiers et les plugins *LustyExplorer* et *LustyJuggler* pour naviguer entre les fichiers. En effet, devoir visualiser l'arborescence pour trouver un fichier est toujours plus lent que de trouver le fichier à partir de son nom par exemple. *vim-fern* vous permettra donc d'obtenir un *Vim* se comportant comme un éditeur classique avec un explorateur de fichiers sur lequel vous pourrez cliquer.

Nous allons tout d'abord installer *vim-fern* à l'aide de *vim-plug* comme précédemment puis activer l'utilisation de la souris dans le terminal.

```
" Activation de vim-plug
call plug#begin()

" Nous mettrons nos plugins ici

" Installation de solarized
Plug 'ericbn/vim-solarized'

" Installation de vim-fern
Plug 'lambdalisue/fern.vim'

call plug#end()

" -- Activation de la souris
set mouse=a
```

Rechargez votre *vimrc* avec la commande suivante : `:source $MYVIMRC` (ou sauvegardez, quittez et réouvrez *Vim* comme précédemment) puis installez le nouveau plugin grâce à `:PlugInstall` (appuyez sur la touche `q` pour quitter la fenêtre d'installation).

Il va ensuite falloir activer le plugin. Vous pouvez le faire manuellement en tapant `:Fern . -drawer -stay` en mode normal. Si vous préférez activer *vim-fern* à chaque fois que vous ouvrez votre *Vim*, ajoutez ces lignes à la fin de votre `~/.vimrc` :

```
" Activation de vim-fern au lancement de vim
augroup FernGroup
  autocmd! *
  autocmd VimEnter * ++nested Fern . -drawer -stay
augroup END
```

C'est, j'en conviens, une commande un peu barbare qui pourrait se traduire en bon vieux français par : à chaque ouverture de vim (`VimEnter`), peu importe le type de fichier (*), lancer *Fern* dans le répertoire courant . en mode `drawer` sur le côté et en gardant `stay` le focus sur la fenêtre actuelle (`Fern . -drawer -stay`).

Pour activer l'ouverture des répertoires et des fichiers au clic de la souris, remplacez le code ci-dessus par :

```
augroup FernGroup
  autocmd! *
  autocmd FileType fern call s:init_fern()

  autocmd VimEnter * ++nested Fern . -drawer -stay
augroup END

function! s:init_fern() abort
  nmap <buffer> <LeftRelease> <Plug>(fern-action-open-or-expand)
endfunction
```

Rien de particulier ensuite, *vim-fern* vous affiche l'arborescence du répertoire où vous avez lancé *Vim*, comme vous le montre la capture d'écran ci-dessous. Vous pouvez utiliser la souris et/ou le clavier pour vous déplacer. À noter que la touche `j` vous permet de descendre, la touche `k` de remonter, la touche `l` de déplier le contenu d'un répertoire ou d'ouvrir le contenu d'un fichier et la touche `h` de le replier. À noter que si vous avez appuyé sur la touche `Entrée` sur un répertoire, *vim-fern* ne vous affichera plus que le contenu de ce répertoire, il vous suffit d'appuyer sur `la touche Retour` pour retourner au répertoire parent.

Vous pouvez aussi effectuer diverses commandes (créer, copier des fichiers) mais nous ne rentrerons pas

en détail dans ces commandes ici. Vous pouvez toujours appuyer sur la touche `?` dans la fenêtre de *vim-fern* pour avoir un aperçu des commandes ou vous rendre sur le [site officiel de vim-fern](#).

Pour passer de la fenêtre de *vim-fern* à la fenêtre d'édition de votre fichier au clavier, appuyez sur `Ctrl + w` puis `w`. C'est à dire la touche `Control (Ctrl)` et tout en la laissant appuyée la touche `w`. Vous pouvez ensuite tout lâcher pour appuyer une nouvelle fois sur `w`. Ce raccourci clavier sera d'ailleurs toujours valable pour naviguer entre vos différentes fenêtres *Vim* (il n'est pas spécifique à *vim-fern*).

Le fichier complet de votre `~/.vimrc` à ce stade est disponible à cette adresse : <http://vimebook.com/link/v2/fr/vim-plug>

3.6 Nous voilà fin prêts

Voilà, vous avez fait le plus dur. Enfin presque. Nous venons de couvrir ce qui manque cruellement à *Vim* : une configuration par défaut acceptable. Je ne dis pas que c'est la panacée pour l'instant, mais ça devrait vous permettre d'avoir un *Vim* utilisable comme n'importe quel autre éditeur de texte dont vous ne connaissez pas encore toutes les possibilités. Je vous recommande à ce stade de commencer à l'utiliser dans votre vie quotidienne. N'hésitez pas à user et à abuser de la souris pour l'instant. Le but ici étant de réduire l'impact de l'utilisation de *Vim* sur votre travail quotidien. Ce n'est pas encore le temps de briller en société. Vous apprendrez les raccourcis clavier au fur et à mesure, et ça ne fait pas de vous un utilisateur de *Vim* de seconde zone. Il faut bien commencer un jour.

Nous allons maintenant aborder ce qui fait l'unicité de *Vim* : sa gestion des modes et des commandes pour manipuler le texte. La balle est dans votre camp maintenant : ou vous êtes prêt à changer vos habitudes et à passer à un autre niveau d'efficacité, ou alors n'utiliser *Vim* que comme un bloc-notes amélioré vous convient (dans ce cas là, vous pouvez vous arrêter là). C'est vous qui voyez !

Chapitre 4

L'outil de manipulation de texte rêvé

Alors oui, pour ceux qui se demandent, je fais des rêves bizarres, mais bon, chacun a ses petites tares cachées. Et rêver d'un outil qui améliore ma vie quotidienne en tant que codeur (ou écrivain, ou formateur, ou ...) n'est pas si étrange que ça.

Ce qui fait et fera encore le succès de *Vim* est sa capacité à **faciliter les manipulations de texte**. Certes il va vous proposer des fonctionnalités propres à chaque tâche que vous effectuerez (souvent par l'intermédiaire de plugins) comme la validation syntaxique de code, la correction orthographique, ... Mais à la fin, c'est toujours à écrire/corriger/manipuler/se déplacer dans du texte que vous passerez la majeure partie de votre temps.

C'est là que l'approche de *Vim* est différente d'IDE comme VSCode / Eclipse / Netbeans / PhpStorm et consorts. Là où ces IDE vont mettre l'accent sur les particularités de votre langage de programmation tout en vous fournissant des capacités de manipulation de texte basiques, *Vim* adopte l'approche opposée : vous serez **très efficace** à manipuler/écrire du texte quel qu'il soit et vous pourrez enrichir *Vim* avec des fonctionnalités propres à votre langage de programmation via des plugins.

Nous allons donc voir dans ce chapitre comment utiliser *Vim* à bon escient (vous allez commencer à oublier votre souris) et quelle est la logique derrière tous ces enchaînements de commandes qui paraissent barbares au non-initié. Vous devriez pouvoir, à la fin de ce chapitre, **vous passer de votre souris** pour éditer/manipuler le contenu d'un fichier. En tout cas, vous devriez vous forcer à le faire en apprenant *Vim*, ce n'est pas si dur que ça, et c'est ce qui fait la différence entre *Vim* et les autres : le tout clavier.

4.1 Se déplacer par l'exemple : Essayer de copier / coller

Nous avons déjà vu dans la section « *Préambule indispensable : le mode insertion* » comment passer du mode insertion (pour saisir du texte) au mode normal (*a priori* pour l'instant, vous ne savez pas trop à quoi sert ce mode). En appuyant sur la touche `i` votre curseur passe en mode insertion (lorsque vous êtes en mode normal) et en appuyant sur la touche `Esc` (Échap) il repasse dans le mode normal. Bon bah on est bien Tintin. Et maintenant ?

4.1.1 Préambule

Nous allons apprendre notre première manipulation de texte : le copier / coller. J'en vois certains d'entre vous se dire que ça ne sert à rien, car vous savez déjà le faire. Vous passez en mode insertion, vous prenez votre souris (ou vous vous déplacez avec les flèches directionnelles tout en appuyant sur la touche `Shift`) pour sélectionner du texte et vous allez dans le menu Édition de votre terminal puis Copier . Et ensuite menu Édition puis Coller . Bah tiens, essayez pour voir.

Si vous avez suivi la section « *Les modes : d'où Vim tire sa puissance* » traitant de la position idéale pour vos mains, vous savez que vous avez fait une ou plusieurs choses que vous devriez vous interdire :

- Vous avez utilisé votre souris

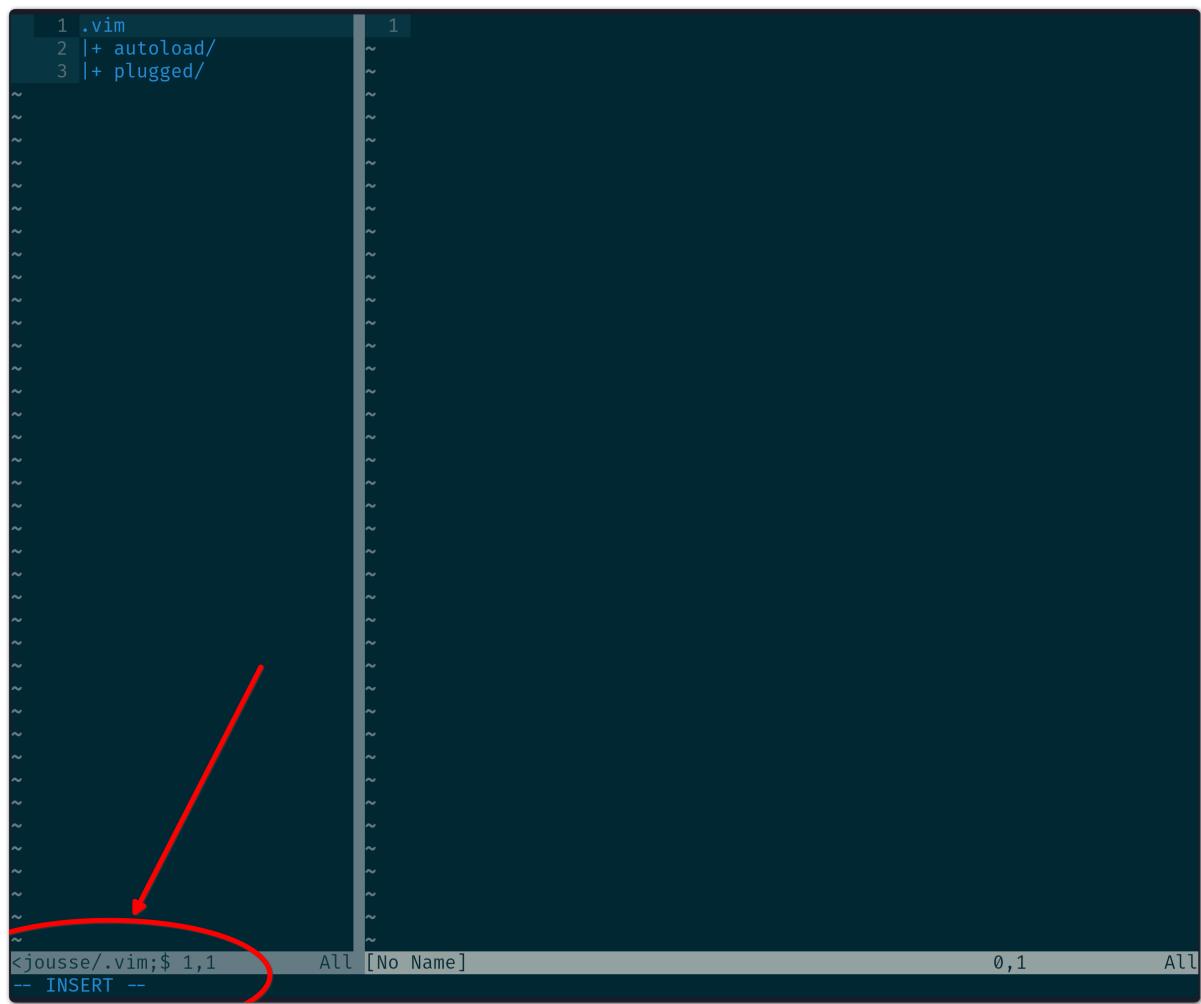
— Vous avez déplacé grandement votre main droite de sa position de repos, pour aller atteindre les flèches directionnelles qui sont très mal placées sur un clavier

Alors certes ce n'est pas grave en soi, mais c'est **inefficace** (se servir de la souris ou déplacer votre main droite vers les touches directionnelles est très lent) et **nuisible** pour vos petites mains. Ceci est votre dernière chance : si vous n'êtes pas prêt à vous forcer à ne pas le faire, **Vim n'est pas fait pour vous**. Vim est parfait pour ne pas utiliser la souris et pour ne pas bouger vos mains (ou presque). Ne pas se forcer à le faire, c'est ne pas tirer partie de tout le potentiel de Vim, et à un moment ou un autre, **vous le quitterez pour un éditeur** qui aura été pensé pour être utilisé à la souris. Alors, on continue ?

4.1.2 Se passer de la souris

Si vous lisez ces lignes c'est que vous avez répondu « oui », allons-y gaiement alors ! Nous allons tout d'abord commencer par nous passer de la souris. La prochaine étape sera de se passer des touches directionnelles, mais chaque chose en son temps.

Pour réaliser un copier/coller avec Vim tout se passe en mode « normal ». Pour savoir dans quel mode vous vous trouvez, vous avez juste à regarder en bas à gauche de votre Vim. La *figure ci-dessous* vous montre Vim en mode « insertion » par exemple.



Lorsque rien n'est marqué en bas à gauche, c'est que vous êtes en mode normal. Pour sortir d'un mode afin de retourner au mode normal, il suffit d'appuyer sur la touche **Esc (Échap)**. À noter que si vous vous demandez pourquoi je vous dis d'arrêter d'utiliser la souris et/ou les touches directionnelles, mais que je ne dis rien sur le fait qu'il faille se torturer la main pour atteindre la touche **Esc (Échap)**, c'est que vous êtes sur la bonne voie. Je vous explique le comment du pourquoi dans « *Se passer de la touche Échap* ».

Admettons donc que vous êtes en mode « normal » et que vous avez un peu de texte de saisi dans votre *Vim* (inséré par vous même grâce à un passage par le mode insertion avec la touche `i` puis retour au mode normal avec la touche `Esc` (Échap)). Par exemple, cette chouette citation de Mark Twain : « Ils ne savaient que c'était impossible, alors ils l'ont fait. ». Votre *Vim* devrait ressembler à celui de la figure ci-dessous. Notez l'absence d'affichage d'un quelconque mode en bas à gauche, c'est donc que l'on est en mode *normal*.

The screenshot shows a terminal window with Vim running. The status bar at the bottom indicates the file is 'book-test' at line 1, column 1, and the buffer is 'All [No Name] [+]' with a total of 1,65-64 lines. The main Vim window displays a single line of text: "Ils ne savaient pas que c'est impossible alors ils l'ont fait.". The left margin of the Vim window shows numerous tilde (~) characters, indicating blank lines or deleted text.

La façon la plus naturelle (mais pas la plus efficace, nous verrons cela un peu plus loin) de copier/coller le mot « impossible » va être de se déplacer sur la première lettre du mot avec les touches directionnelles, d'appuyer sur la touche `v` (pour passer en mode « visuel »), de se déplacer sur la dernière lettre (vous devriez avoir le mot sélectionné, en surbrillance) puis d'appuyer sur la touche `y` (la touche `y` étant utilisée comme raccourci du mot *yank* en anglais qui veut dire *extraire*). Vous avez copié votre premier mot.

Déplacez vous ensuite à la fin de la phrase avec les flèches (toujours en mode « normal ») puis appuyez sur la touche `p` (raccourci du mot *paste* cette fois-ci pour *coller*). Le mot devrait avoir été collé à la fin, et vous devriez avoir le même rendu que la figure qui suit.

A screenshot of the Vim text editor. The left pane shows a file tree with two entries: 'book-test' (containing a single file) and 'README.md'. The right pane displays the contents of 'README.md', which consists of approximately 100 lines of the character '~'. The status bar at the bottom shows the command 'All' and the line numbers '1,75-74'. The title bar says 'Chapitre 4. L'outil de manipulation de texte rêvé'.

On se rend donc compte ici que *Vim* se sert de l'astuce des modes (et notamment du mode « normal » pour les déplacements) afin de ne pas avoir à se servir de la souris. À partir du moment où vous aurez pris l'habitude de passer rapidement d'un mode à l'autre (et pour cela se passer de la touche `Esc` (Échap) va devenir indispensable), utiliser la souris vous apparaîtra comme une perte de temps, mais pour cela il va falloir pratiquer un peu bien sûr.

4.2 Se passer des touches directionnelles

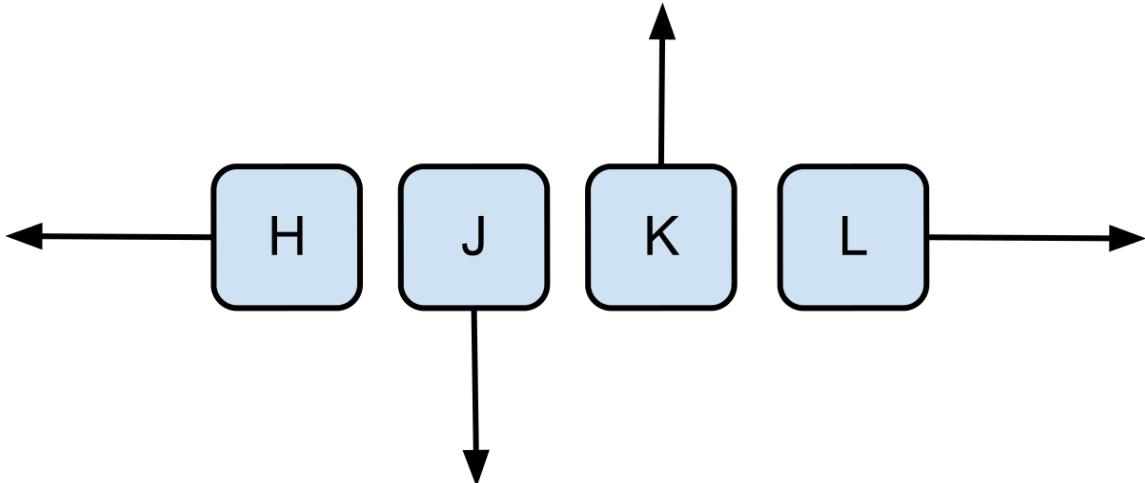
Nous y voilà. Encore plus que de se priver de la souris, se priver des touches directionnelles est la chose à faire si l'on veut utiliser *Vim*, pour de vrai. *Vim* va vous permettre de faire tout plus rapidement et plus intuitivement à la seule condition de le faire sans les touches directionnelles. Cela va vous permettre comme je l'ai déjà dit de ne pas bouger votre main certes, mais ça va aussi vous forcer à passer en mode « normal » pour réaliser vos déplacements et vos mouvements de texte. Il n'y a qu'à ce moment là (un peu douloureux au début il est vrai) que vous commencerez à vraiment tirer parti de *Vim*.

Pour cette section, je vais vous expliquer comment vous déplacer sans utiliser les touches directionnelles. Puis, une fois que vous aurez une vague idée de comment faire, je vous donnerai le code à mettre dans votre `~/.vimrc` pour désactiver les touches directionnelles complètement. Car oui, il n'y a que comme ça que vous y arriverez (en tout cas il n'y a que comme ça que j'y suis arrivé).

4.2.1 Se déplacer sans les touches directionnelles

En mode normal, 4 touches vont vous permettre de déplacer le curseur d'un caractère :

- la touche `h` pour aller **à gauche**
- la touche `j` pour aller **en bas**
- la touche `k` pour aller **en haut**
- la touche `l` pour aller **à droite**



Vous pouvez remarquer que ces touches sont placées sur la rangée de repos de manière à déplacer vos doigts le moins possible. En essayant de placer vos doigts pour atteindre ces lettres vous devriez vous rendre compte que l'index a deux déplacements (gauche et bas) alors que l'auriculaire n'en a pas. Vous verrez qu'on s'y fait assez rapidement et que l'index étant plus fort que l'auriculaire, ça tombe plutôt bien. Vous trouverez le clavier sur lequel *Vi* a été conçu dans la section « [Se passer de la touche Échap](#) », vous comprendrez ainsi le pourquoi du comment.

À noter qu'à force, on se sert de moins en moins des déplacements gauche/droite d'un caractère. On va leur préférer les déplacements de mot en mot, de paragraphe en paragraphe ou les déplacements grâce à des recherches. Quelques exemples de déplacements « rapides » que j'utilise :

Touche	Déplacement
<code>e</code>	à la fin du mot courant
<code>b</code>	au début du mot courant
<code>w</code>	au début du mot suivant
<code>^</code>	au premier caractère non blanc de la ligne
<code>\$</code>	à la fin de la ligne
<code>0</code>	au début de la ligne

Vous avez ici le minimum pour vous déplacer en mode normal. Il existe aussi des commandes vous permettant de vous déplacer puis de rentrer en mode insertion directement, elles sont très pratiques car elles vont vous permettre d'économiser quelques touches. En voici quelques unes que j'utilise à peu près tout le temps :

Touche	Action
<code>i</code>	se place en mode insertion avant l'emplacement du curseur
<code>a</code>	se place en mode insertion après l'emplacement du curseur
<code>I</code>	se place en mode insertion au début de la ligne
<code>A</code>	se place en mode insertion à la fin de la ligne
<code>o</code>	insère une nouvelle ligne en dessous de la ligne courante
<code>O</code>	insère une nouvelle ligne au dessus de la ligne courante
<code>r</code>	remplace les caractères sous le curseur

Arrêtons-nous un peu là dessus. Au risque d'insister lourdement, mais la clé de l'utilisation de *Vim* vient de ce que nous venons de voir dans ce chapitre, ni plus, ni moins. Il y a une chose que vous avez à vous forcer à faire, c'est **d'utiliser les touches hjkl** pour les déplacements. Si vous y arrivez, vous apprendrez tout le reste au fur et à mesure.

Vous trouverez des sites entiers vous détaillant les différentes commandes possibles, les différentes combinaisons, j'en passe et des meilleures. Vous les apprendrez puis les oublierez (ou pas, en fonction de si elles vous sont vraiment utiles). Si vous avez un seul effort à faire c'est celui de se passer des touches directionnelles et donc de vous forcer à utiliser le mode normal. Le reste tombera sous le sens.

Voici l'ultime configuration qu'il vous faudra mettre dans votre `~/.vimrc` pour atteindre le Saint Graal : désactiver les touches directionnelles.

```
" Désactivation des touches directionnelles
map <up> <nop>
map <down> <nop>
map <left> <nop>
map <right> <nop>
imap <up> <nop>
imap <down> <nop>
imap <left> <nop>
imap <right> <nop>
```

Nous y voilà. Croyez-moi, vous allez souffrir un peu au début. Pour moi, ça n'a pas duré plus de deux jours. Ensuite vous aurez oublié. Si vous n'êtes pas prêt à galérer un peu pendant deux jours pour améliorer votre efficacité à vie, que faites-vous ici !

Je ne vous donnerai pas d'autres détails sur toutes les touches possibles pour vous déplacer, d'autres ressources le font déjà bien mieux que moi. Je vais en revanche vous apprendre dans *Combiner des touches/déplacements* comment les utiliser à bon escient.

On peut notamment citer le livre gratuit « A byte of Vim » disponible à l'adresse suivante : <https://vim.swaroopch.com/> (malheureusement sa traduction en français a disparu depuis la première version de ce livre).

Ou encore l'infographie de la figure ci-dessous (téléchargeable sur <https://github.com/vjousse/vim-for-humans-book/blob/master/book-tex/graphics/vi-vim-cheat-sheet.png>) qui donne un aperçu des différents mouvements pour chacune des touches d'un clavier français.

version 1.2
April 13th, 2010

vi / vim graphical cheat sheet

Esc normal mode	1	2	~ toggle case	# prev ident	{ begin parag.	• misc	bol/ goto col	• goto mark	\• not used!	^ "soft" bol	@ play macro]• misc	}` end parag.
2	& repeat s	é	" reg. spec	' . m.k. bol	(begin sentence	- prev line	è	"soft" bol down	ç	0 "hard" bol	o	+ next line	
A append	Z quit	E end WORD	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	..	£		
a append	Z extra cmd	E end word	R replace char	T 'till	Y 'yank	U undo	I insert mode	O open below	P paste after	^ first non-blank	\$ eol		
Q ex mode	S subst line	D delete to eol	F "back" find ch	G eof/ln find	H screen top	J join lines	K man	L screen bottom	M screen mid	% goto match	ù		
Q record macro	S subst char	D delete	F find char	G extra cmd	H h ←	J j ↓	K k ↑	L l →	M m mark	*	next indent		
> indent	W next WORD	X back-space	C change to eol	V visual lines	B prev WORD	N prev (find)	? find (rev.)	. repeat cmd	/ find	§			
< un- ³	W next word	X delete char	C change	V visual mode	B prev word	N next (find)	reverse , t/T/f/F	; repeat ; t/T/f/F	: ex cmd	external filter			

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

N'oubliez pas que le but ici est de gagner en rapidité en ne bougeant quasi plus ses mains de la rangée de repos, et en utilisant le plus possible le « mode normal ». Au boulot !

4.3 Se passer de la touche Échap

Utiliser la touche **Esc** (Échap) pour sortir du mode « insertion » semble être une hérésie tellement elle est difficilement accessible. Il faut déplacer entièrement la main gauche pour y accéder ou alors se torturer le petit doigt.

Pour comprendre pourquoi la touche **Esc** (Échap) est utilisée par défaut, il faut faire un bon de quelques années en arrière, pour se retrouver en face du clavier sur lequel *Vi* a été développé. Vous pouvez voir sur la photo ci-dessous que la touche **Esc** (Échap) était très facilement accessible. Vous pouvez aussi noter l'emplacement des touches directionnelles. Malheureusement depuis, cela a bien changé.



L'étape ultime (après avoir réussi à se passer des touches directionnelles) est donc de rapprocher la touche **Esc** (Échap) de vos petits doigts. Il y a plusieurs solutions pour cela, mais celle que je vous recommande

si vous avez un clavier avec une disposition française est la suivante (dans votre `~/.vimrc`) :

```
" On attribue ;; à la touche <Esc>
" Les ; sont rarement utilisés l'un à la suite de l'autre
:imap ;; <Esc>
:map ;; <Esc>
```

Lorsque vous êtes en mode insertion, il vous suffit d'appuyer deux fois sur la touche `;` pour retourner au mode normal. La touche `;` ne vous demande pas de bouger votre main de la rangée de repos et on l'utilise rarement deux fois de suite (et si c'est le cas, il suffit d'attendre un peu avant de taper le deuxième `;`), c'est donc le parfait candidat.

Voici d'autres solutions possibles (cf http://vim.wikia.com/wiki/Avoid_the_escape_key):

```
:imap jj <Esc>
:imap jk <Esc>
:imap ii <Esc>
:imap ` <Esc>

" Shift-Espace (peut ne pas marcher sur votre système).
:imap <S-Space> <Esc>

" Sous Linux avec gvim Vim en console, vous pouvez utiliser Alt-Space.
:imap <M-Space> <Esc>
```

4.4 Combiner des touches/déplacements

Maintenant que nous savons nous déplacer en mode normal, il est temps de voir comment réaliser d'autres opérations. Nous avons déjà vu le copier/coller au chapitre [Se déplacer par l'exemple : Essayer de copier / coller](#), nous allons maintenant voir comment supprimer/éditer plus facilement.

Dans [Se passer des touches directionnelles](#) nous avons vu qu'il suffisait d'utiliser la touche `w` pour se déplacer au début du mot suivant. Nous allons essayer de combiner cela avec quelques nouvelles touches du mode normal :

- la touche `d` est utilisée pour « supprimer »
- la touche `c` est utilisée pour « supprimer et passer en mode insertion »

À noter que ce qui est supprimé est placé dans le presse-papier en même temps (le « supprimer » se comporte par défaut comme un « couper »).

La particularité de ces touches, c'est qu'elles attendent ensuite un « ordre de déplacement » pour savoir quoi supprimer. Il va donc falloir les combiner avec les déplacements que nous avons déjà vus dans [Se passer des touches directionnelles](#).

Cela donnera par exemple :

Action	Résultat
la touche <code>d</code> puis la touche <code>w</code>	supprime les caractères jusqu'au prochain mot
la touche <code>c</code> puis la touche <code>w</code>	supprime les caractères jusqu'au prochain mot et passera en mode insertion
la touche <code>d</code> puis la touche <code>\$</code>	supprime tout jusqu'à la fin de la ligne
la touche <code>d</code> puis la touche <code>^</code>	supprime tout jusqu'au début de la ligne

Vous pouvez aussi utiliser cela pour copier :

Action	Résultat
la touche <code>y</code> puis la touche <code>w</code>	copie les caractères jusqu'au prochain mot
la touche <code>y</code> puis la touche <code>\$</code>	copie tout jusqu'à la fin de la ligne
la touche <code>y</code> puis la touche <code>^</code>	copie tout jusqu'au premier caractère non blanc de la ligne

Il ne vous restera qu'à appuyer sur la touche `p` pour coller ce que vous voulez où vous voulez. Par défaut la touche `p` colle le texte après la position courante du curseur. Si vous voulez coller avant la position du curseur, utilisez la touche `P`.

Il arrive de temps en temps de vouloir aussi supprimer du texte (non sans blague!), voici quelques commandes utiles pour cela :

Action	Résultat
<code>dd</code>	efface la ligne courante et la place dans le presse-papier
<code>x</code>	efface le caractère sous le curseur
<code>X</code>	efface le caractère avant le curseur

La plupart des mouvements peuvent être préfixés par un nombre multiplicateur. Voici quelques exemples :

Action	Résultat
<code>2 d d</code>	efface deux lignes
<code>3 x</code>	efface 3 caractères vers l'avant du curseur
<code>3 X</code>	efface 3 caractères vers l'arrière du curseur
<code>2 y y</code>	copie 2 lignes dans le presse-papier
<code>5 j</code>	se déplace de 5 lignes vers le bas

4.5 Rechercher / Se déplacer rapidement

Maintenant que nous connaissons les commandes de base pour éditer du texte avec *Vim*, voyons voir comment nous déplacer plus rapidement dans notre document. Nous avons déjà évoqué les touches `w`, `b`, `^` et `$` qui nous permettent respectivement de se déplacer à la fin d'un mot, au début d'un mot, au début d'une ligne et la fin d'une ligne. Tout d'abord, voyons voir comment « scroller » sans la souris. À noter que toutes ces commandes se font en mode « normal ».

4.5.1 Sauts de page

Pour faire défiler les pages, il faut utiliser :

- `Control + f` pour passer à la page suivante (`f` pour forward)
- `Control + b` pour passer à la page précédente (`b` pour backward)

Ces raccourcis vont vous permettre d'avancer rapidement dans votre document.

Vous pouvez aussi :

- Vous rendre au début du fichier en tapant `gg`
- Vous rendre à la fin du fichier en tapant `G`
- Vous rendre à la ligne 23 en tapant `: 23`

4.5.2 Les marqueurs

Lorsque je me déplace dans un fichier, j'aime bien pouvoir revenir à certains endroits. Par exemple lorsque je me rends au début du fichier alors que j'étais en train de travailler au milieu de celui-ci, j'aime bien pouvoir revenir directement où je travaillais. Heureusement, *Vim* a tout prévu pour cela grâce à l'utilisation de **marqueurs**. Les marqueurs sont tout simplement des « marque-pages » qui permettent à votre curseur de se retrouver à la position où vous aviez mis votre marqueur.

Un marqueur se pose en tapant `m a`. Pour déplacer votre curseur à la position du marqueur tapez `' a`. Vous pouvez placer plusieurs marqueurs en changeant `a` par n'importe quelle lettre de l'alphabet (on appelle cela des registres en langage *Vim*). Pour placer un autre marqueur vous pouvez par exemple utiliser la lettre `d`. Grâce à `m d` vous placerez le marqueur et à `' d` vous vous y rendrez.

4.5.3 La recherche

En mode normal, vous pouvez lancer une recherche en utilisant la touche `/` suivie du texte que vous souhaitez rechercher puis de la touche `Entrée`. Grâce à notre configuration de *Vim* vous devriez voir vos occurrences de recherche surlignées en même temps que vous tapez. Par défaut la recherche n'est pas sensible à la casse (pas de différence entre minuscules/majuscules). En revanche, dès que vous taperez une majuscule, la recherche deviendra sensible à la casse. Vous pouvez vous déplacer à la prochaine occurrence de la recherche grâce à la touche `n`. Pour vous déplacer à la précédente utilisez la touche `N`.

Pour rappel, voici les lignes de votre fichier de configuration qui permettent de faire cela :

```
" -- Recherche
set ignorecase          " Ignore la casse lors d'une recherche
set smartcase            " Si une recherche contient une majuscule,
                        " re-active la sensibilité à la casse
set incsearch            " Surligne les résultats de recherche pendant la
                        " saisie
set hlsearch             " Surligne les résultats de recherche
```

Attention par défaut, la recherche utilise les expressions régulières POSIX. Si vous souhaitez rechercher des caractères habituellement utilisés dans les expressions régulières (comme `[] ^{ } $ /`) n'oubliez pas de les préfixer par `\`.

Vous pouvez aussi rechercher directement le mot qui est placé sous votre curseur grâce à la touche `*`. Utiliser la touche `*` fera une recherche vers l'avant. Pour faire une recherche vers l'arrière, utilisez la touche `#`. Pour annuler une recherche, en mode normal, tapez `:noh`.

4.6 Le mode visuel

Je vous en ai déjà parlé lors de l'explication sur le Copier / Coller, mais comme je sais que certains d'entre vous sont tête en l'air, je vous fais un petit rappel ici.

Lorsque vous êtes en mode « normal » appuyez sur la touche `v` pour passer en mode « visuel ». Vous pourrez alors sélectionner des caractères ou des lignes entières grâce aux différentes façons de vous déplacer que vous venez d'apprendre. Vous pourrez ensuite copier le texte sélectionné avec la touche `y` puis le coller avec la touche `p`. Pour le couper il vous faudra utiliser la touche `d`.

En mode normal vous pourrez utiliser la touche `V` pour sélectionner lignes par lignes. Et bien sûr, utiliser la touche `Esc (Échap)` ou `;` pour revenir au mode normal.

4.7 À vous de jouer

Vous trouverez une version complète du fichier de configuration à ce stade en ligne ici <http://vimebook.com/link/v2/fr/text-manip>.

Vous devriez maintenant être capable de n'utiliser que le clavier pour les opérations de manipulation de texte et d'édition. Je n'ai fait que survoler la puissance de *Vim* ici, mais ça devrait être suffisant pour survivre. Je vous ai donné ici le strict nécessaire, mais ce strict nécessaire vous permet déjà de profiter de *Vim* et du plaisir de ne plus utiliser la souris.

À vous maintenant de lire les nombreuses ressources disponibles sur internet vous décrivant tous les mouvements possibles et imaginables.

Voici une liste de ressources qui pourraient vous être utiles, malheureusement les ressources en français sont assez rares :

- Le site de ce livre <https://www.vimebook.com>
- A byte of *Vim* <https://vim.swaroopch.com/>
- Un petit pense bête sympathique de différents raccourcis clavier <https://tuteurs.ens.fr/unix/editeurs/vim.html>
- Un site avec plein de pense-bêtes pour les raccourcis <https://vim.rtorr.com/>
- Les vidéos d'Andrew Stewart en anglais mais vraiment superbement réalisées : <https://www.pluralsight.com/courses/smash-into-vim>
- Le blog de Derek Wyatt's en anglais <http://derekwyatt.org/vim/tutorials/>
- Le livre « Learning to play Vim » en anglais <https://themouseless.dev/vim/>
- Le site « vim-hero » avec un tutorial interactif très bien fait <https://www.vim-hero.com/>

Histoire de réveiller l'enfant qui est en vous, je vous conseille vivement d'aller vous amuser avec <http://vim-adventures.com/>. C'est un jeu de rôle en ligne qui a pour but de vous apprendre à manipuler *Vim* ! Voici un petit aperçu :



Nous allons maintenant passer à la vitesse supérieure : l'utilisation de plugins, ou comment rendre *Vim* incontournable.

Chapitre 5

Les plugins indispensables

Soyons clair, *Vim* sans ses plugins, c'est comme Milan sans Rémo (© François Corbier - Sans ma barbe - <http://www.bide-et-musique.com/song/149.html>) : ça ne rime à rien. C'est grâce aux plugins que *Vim* va pouvoir pleinement exprimer toute sa puissance et vous éléver à un autre niveau de productivité. Vous n'avez pas besoin d'en avoir des mille et des cents, mais quelques uns savamment choisis devraient faire l'affaire.

Qu'on ne se méprenne pas, *Vim* peut bien sûr s'utiliser sans plugins. Il peut d'ailleurs s'avérer utile de savoir faire les manipulations de base sans avoir besoin d'installer de plugin, car c'est souvent le cas sur des serveurs : il n'y a aucun plugin d'installé. Dans ce cas là, savoir ouvrir, sauvegarder sous, passer d'un fichier à l'autre avec les commandes de *Vim* par défaut peut vous sauver la mise. En revanche, dans votre travail quotidien de rédaction ou de code, les plugins sont indispensables pour pleinement tirer partie de *Vim*.

5.1 Naviguer sur le disque et entre les fichiers : *Lusty Explorer*

Nous avons déjà vu *vim-fern* dans *Explorateur de fichiers : vim-fern* qui permettait d'avoir un explorateur de projet dans une fenêtre latérale de *Vim*. Le problème de ce plugin est qu'il n'est pas fait pour être utilisé au clavier. Certes vous pouvez utiliser le clavier, mais il ne sera pas aussi efficace que les plugins pensés uniquement pour une utilisation au clavier.

Personnellement, le premier plugin que j'installe partout où j'ai à utiliser *Vim*, c'est *Lusty Explorer* (http://www.vim.org/scripts/script.php?script_id=1890). Ce plugin va vous permettre de naviguer sur votre disque dur pour ouvrir facilement des fichiers en se passant de la souris. Il va aussi permettre de naviguer rapidement entre vos différents fichiers déjà ouverts (vos buffers en jargon *Vim*). Commençons par l'installer via *vim-plug*. Comme d'habitude, ajoutez la ligne ci-dessous à la suite des plugins déjà listés dans votre `~/.vimrc` :

```
" Installation de Lusty Explorer
Plug 'sjbach/lusty'
```

Reste maintenant à voir comment l'utiliser. Si l'on se réfère à la documentation, voilà ce que l'on trouve (traduit en français) :

```
<Leader>lf - Ouvre l'explorateur de fichiers.
<Leader>lr - Ouvre l'explorateur de fichiers à partir du répertoire du fichier courant.
<Leader>lb - Ouvre l'explorateur de buffers.
<Leader>lg - Ouvre la recherche dans les buffers.
```

On voit qu'il est fait mention d'une touche nommée `<Leader>` qu'il faut ensuite faire suivre d'autres touches comme `lf`, `lr`, `lb` et `lg`. Cette touche `<Leader>` est une touche spéciale que l'on définit dans son fichier `~/.vimrc`. Elle sera énormément utilisée par tous les plugins, beaucoup des commandes de ces

derniers commenceront par la touche `<Leader>`. C'est un moyen d'éviter les collisions avec les raccourcis par défaut de *Vim*.

Il faut donc choisir une touche `<Leader>`. Par défaut, *Vim* utilise `\` comme touche `<Leader>`. Sur nos claviers francophones c'est une très mauvaise idée d'utiliser cette touche car elle n'est pas pratique du tout. La plupart des utilisateurs de *Vim* la remplace par la touche `,`. Elle est directement accessible sous l'index de la main droite ce qui en fait une parfaite candidate. Pour spécifier cela à *Vim* il va falloir rajouter une ligne dans votre fichier `~/.vimrc`, à savoir :

```
" Utilisation de , comme touche <Leader>
let mapleader = ","
```

Une fois la modification effectuée et prise en compte (en redémarrant *Vim* ou en tapant `:so ~/.vimrc` ou `:so $MYVIMRC` en mode normal), vous devriez être en mesure de taper `,lr` et d'avoir le même style de résultat que sur la figure ci-dessous (notez l'affichage du contenu de votre dossier actuel en bas à gauche).

The screenshot shows a terminal window with two panes. The left pane displays the contents of the `~/.vim` file:

```
1 .vim
2 |+ autoload/
3 |+ plugged/
```

The right pane shows a list of files from the `LustyExplorer--Files` directory, with the `autoload/` folder highlighted in yellow:

```
<jousse/.vim; $ 1,1      All [No Name]          0,0-1      All
autoload/    plugged/   LustyExplorer--Files        1,104     All
>> /home/vjousse/.vim/
```

Vous pouvez constater sur [la capture d'écran de lusty](#) qu'il y a deux parties à *Lusty Explorer*. La partie basse vous indique le répertoire que vous êtes en train d'explorer et la partie haute liste le contenu de ce répertoire. En surbrillance se trouve l'élément couramment sélectionné. Dans le cas de [la capture d'écran de lusty](#) c'est le répertoire `autoload/` en jaune (la couleur pourra être différente en fonction de votre thème).

Lusty Explorer utilise une fonctionnalité de *Fuzzy matching* qui va vous permettre de ne taper qu'une partie d'un nom de fichier ou de répertoire pour le sélectionner. Dans mon exemple, si, dans la fenêtre de *Lusty*, je saisi `pl` il va me sélectionner le répertoire `plugged/` sans que j'ai à lui spécifier le nom entier, je n'aurai ensuite plus qu'à appuyer sur la touche `Entrée` pour ouvrir le fichier dans *Vim*. La

figure suivante vous montre l'exemple en question.

The screenshot shows a terminal window with Vim running. The buffer contains the following text:

```
1 .vim
2 |+ autoload/
3 |+ plugged/
~
```

Below the buffer, the status bar shows:

```
<jousse/.vim; $ 1,1      All [No Name]          0,0-1      All
plugged/
LustyExplorer--Files      1,104      All
>> /home/vjousse/.vim/pl
```

To the right of the terminal, the Lusty Explorer sidebar is visible, displaying a list of files and folders.

Lusty Explorer dispose en plus de quelques raccourcis bien pratiques pour utiliser le navigateur de fichiers :

- **Control + n** pour sélectionner le fichier/répertoire suivant
- **Control + p** pour sélectionner le fichier/répertoire précédent
- **Control + w** pour descendre au répertoire parent
- **Control + e** crée un nouveau fichier vide (non sauvegardé sur le disque) avec le nom spécifié actuellement dans *Lusty Explorer*. Vous n'aurez plus qu'à utiliser **:w** pour écrire le contenu du fichier sur le disque.

Lusty Explorer s'utilise donc pour deux choses : naviguer sur votre système de fichiers avec **,lr** et **,lf**, et naviguer entre vos fichiers ouverts (buffers) avec **,lb**. Personnellement j'utilise moins la recherche dans les buffers avec **,lg**, à vous de tester et de vous faire votre propre opinion.

Je vous conseille en guise de test d'ouvrir plusieurs fichiers avec **,lr** ou **,lf**. Ensuite, entraînez-vous à naviguer entre ces différents fichiers ouverts en même temps à l'aide de **,lb**. C'est une des combinaisons que j'utilise le plus au quotidien.

Ce plugin est indispensable et ajoute à lui seul énormément de valeur à *Vim* : se passer de la souris pour ouvrir des fichiers. Prenez donc le temps nécessaire pour l'apprendre correctement, c'est un investissement qui vaut le coup.

5.2 Recherche de fichiers, de chaînes de caractères et d'un peu tout : fzf

Dans le monde informatique il existe un moyen très efficace pour rechercher des choses dont on connaît « à peut près » le nom, on appelle ça le **Fuzzy Matching** (*recherche approximative* ou *recherche floue* en français, cf https://fr.wikipedia.org/wiki/Recherche_approximative). Cette technique va nous permettre de trouver des fichiers dont on connaît une partie du nom, ou alors une partie du répertoire parent et une partie du nom par exemple. Elle va aussi nous permettre de faire la même chose pour le contenu des fichiers : pas besoin d'être très précis (majuscules, minuscules, accents, etc) le **Fuzzy Matching** nous retournera ce qui se rapproche le plus du terme que l'on recherche.

fzf est la référence dans le domaine : il permet de faire du **Fuzzy Matching** un peu partout et notamment avec vim (ça tombe bien hein 😊) !

5.2.1 Installation de fzf

Ajoutez ces deux lignes à vos plugins dans votre `~/.vimrc` pour installer `fzf` et le plugin *Vim* correspondant :

```
" Installation de fzf
Plug 'junegunn/fzf', { 'do': { -> fzf#install() } }
Plug 'junegunn/fzf.vim'
```

Puis ajoutez ces mappings plus bas dans votre fichier (n'importe où après le `call plug#end()`) :

```
" -- Mappings FZF
" On recherche dans les fichiers du répertoire courant
nmap <silent> <Leader>ff :Files<CR>
" On recherche dans les buffers ouverts
nmap <silent> <Leader>fb :Buffers<CR>
" On recherche dans le contenu des fichiers
nmap <silent> <Leader>fr :Rg<CR>
```

Comme d'habitude, pour prendre en compte ces modifications, tapez `:so ~/.vimrc` ou `:so $MYVIMRC` en mode normal puis `:PlugInstall` pour installer les deux plugins.

5.2.2 Recherche de fichiers par nom

Tapez `,ff` en mode normal (ou `:Files`) et vous devriez voir une fenêtre similaire à la capture d'écran suivante s'ouvrir : `fzf-files`

The screenshot shows a Vim session with two windows. The left window displays the contents of the file `.vim`, which includes lines like `1 .vim`, `2 |+ autoload/`, and `3 |+ plugged/`. The right window shows a search results window with a title bar "1 / 903". Inside, there's a preview of a file named `plugged/fzf/README.md`. The preview content includes HTML code such as `<div align="center">`, `^{Special thanks to:}`, and `<a href="https://warp.dev/?utm_sour`. Below the preview, the status bar shows "5/378 (0)" and "5/378". The bottom status bar indicates the current file is `All [No Name]` and shows coordinates "0,0-1" for both the left and right panes.

En fonction d'où vous avez ouvert votre *Vim*, les résultats seront bien sûr différents. Je l'ai pour ma part ouvert dans le répertoire `.vim`. Vous pouvez noter que j'ai juste tapé *REAm* et qu'il a automatiquement trouvé tous les fichiers nommés *README.md*. Il a même mis en surbrillance dans les noms de fichiers ce qui a permis de faire la correspondance, dans notre cas le *REA* au début du nom du fichier puis le *md* dans l'extension du fichier.

Vous pouvez naviguer dans les résultats de recherche avec les raccourcis *Vim* par défaut, à savoir `Ctrl-k` pour bouger la sélection d'une ligne au dessus et `Ctrl-j` pour bouger la sélection d'une ligne en dessous. Il suffira ensuite d'appuyer sur la touche `Entrée` pour ouvrir le fichier sélectionné. Vous noterez l'aperçu du fichier à droite de la fenêtre qui s'est ouverte. Vous pouvez naviguer dans cet aperçu grâce à `Shift-haut` et `Shift-bas` (eh oui, pas de raccourci *Vim* pour cette fonction !)

5.2.3 Recherche de chaînes de caractères dans les fichiers

Pour rechercher dans les fichiers nous allons utiliser un outil nommé *rg* (pour *ripgrep*). Assurez-vous donc de l'avoir installé, les instructions sont disponibles sur le [github de rg](#). Si vous ne connaissez pas *ripgrep*, il est grand temps de remplacer votre traditionnel *grep* par *rg* : il est beaucoup plus performant et globalement bien mieux fichu.

Une fois *rg* installé, tapez `,fr` en mode normal (ou `:Rg`) et vous devriez voir une fenêtre similaire à *la capture d'écran suivante* s'ouvrir :

The screenshot shows a Vim session with several buffers open. The main buffer contains the file `vim`, showing configuration snippets for autoloading and plugging. A search window is open over the buffer, with the query `Rg> config`. The results show matches in the `README-VIM.md` file, specifically in the `Configuration` section of the `zf/ZFZ!` plugin documentation. The search results window displays lines 92 to 112, detailing fzf configuration options like `g:fzf_action`, `g:fzf_layout`, `g:fzf_colors`, and `g:fzf_history_dir`. The status bar at the bottom shows the current buffer is `All [No Name]`.

Dans mon exemple, *fzf* a trouvé le texte *config* au sein du fichier `README-VIM.md` dans le titre `### Configuration`.

5.2.4 Recherche dans les noms de buffers

Tapez `,fb` en mode normal (ou `:Buffers`) et vous devriez voir une fenêtre similaire à *la capture d'écran suivante* s'ouvrir :

The screenshot shows a terminal window running Vim. In the top buffer, there is configuration code for the .vimrc file. In the bottom buffer, three files are listed: README.md, LICENSE, and README-VIM.md. A tooltip for the LICENSE file displays its contents, which is the MIT License. The bottom part of the screen shows the configuration for vim's key bindings.

```
1 .vim  
2 |+ autoload/  
3 |+ plugged/  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
[1]     plugged/fzf/README.md  
> [9] #  plugged/fzf/LICENSE  
[11] %  plugged/fzf/README-VIM.md  
2/2 ━━  
Buf> ━━
```

```
85 " Look for files under your home directory  
86 :FZF ~  
87  
88 " With fzf command-line options  
89 :FZF --reverse --info=inline /tmp  
90  
91 " Bang version starts fzf in fullscreen mode  
92 :FZF!  
93
```

```
1 The MIT License (MIT)  
2  
3 Copyright (c) 2013-2024 Junegunn Ch  
4  
5 Permission is hereby granted, free  
6 of this software and associated doc  
7 in the Software without restriction  
8 to use, copy, modify, merge, publis  
9 copies of the Software, and to perm  
10 furnished to do so, subject to the  
11  
12 The above copyright notice and this  
13 all copies or substantial portions  
14  
15 THE SOFTWARE IS PROVIDED "AS IS", W  
16 IMPLIED, INCLUDING BUT NOT LIMITED  
17 FITNESS FOR A PARTICULAR PURPOSE AN  
18 AUTHORS OR COPYRIGHT HOLDERS BE LIA  
19 LIABILITY, WHETHER IN AN ACTION OF  
20 OUT OF OR IN CONNECTION WITH THE SO  
21 THE SOFTWARE.
```

```
115 ```vim  
116 " This is the default extra key bindings  
117 let g:fzf_action = {  
118   \ 'ctrl-t': 'tab split',  
119   \ 'ctrl-x': 'split',  
120   \ 'ctrl-v': 'vsplit' }  
121
```

<jousse/.vim; \$ 1,1 All plugged/fzf/README-VIM.md 102,1
 0,0-1 All

Vous noterez que j'avais pour ma part 3 fichiers (buffers) ouvertes et vous aurez remarqué que cette fonctionnalité est similaire à celle déjà présente dans *LustyExplorer*. À vous de choisir celle que vous préférez !

Vous trouverez une version complète du fichier de configuration en ligne ici <http://vimebook.com/link/v2/fr/full>.

5.3 Les plugins avancés

J'aurais pu faire un livre entier qui recense les plugins *Vim*, mais je pense que l'intérêt aurait été assez limité. Je ne vais donc pas vous décrire plus en détails d'autres plugins, ceux que je vous ai présentés jusqu'ici devraient vous suffrir pour utiliser *Vim* de manière efficace ! En revanche je vous donne ci-dessous une liste de plugins qui pourraient vous intéresser. Cette liste est issue d'un sondage que j'avais effectué sur Twitter demandant à mes followers quels étaient les plugins *Vim* indispensables selon eux. La voici :

- **coc.vim**. C'est un plugin qui va transformer votre *Vim* en IDE complet à la VSCODE : auto-complétion des fonctions, des classes, « allez à la définition », etc. Même si la tendance est à utiliser des *plugins plus simples qui s'intègrent avec des LSP (Language Server Protocol)* directement, *coc.vim* a l'avantage d'être complet, testé et de s'appuyer sur les configurations similaires à VSCODE. Il a aussi le désavantage d'utiliser javascript. Le repo Github : <https://github.com/neoclade/coc.nvim>.
- **surround**. Ce plugin permet de gérer (changer, ajouter, supprimer) tout ce qui « entoure » : les parenthèses, les crochets, les guillemets ... Par exemple vous pourrez en une combinaison de touches changer « Hello world! » en “Hello world!” ou <q>Hello world!</q>. Le repo Github : <https://github.com/tptope/vim-surround>.

- **fugitive**. Si vous travaillez sur du code source vous utilisez forcément un gestionnaire de version de code source. Si ce n'est pas le cas vous pouvez aller vous cacher. Sinon si vous utilisez Git, Le plugin fugitive est pour vous. Il permet de gérer git directement dans *Vim*. Le repo Github : <https://github.com/tptope/vim-fugitive>
- **ALE**. ALE vérifie pour vous la syntaxe de votre code source. Il va, comme peut le faire VSCode par exemple, vous afficher vos erreurs de syntaxe directement dans *Vim*. Peut vous faire gagner un temps certain si vous éditez souvent du code. Si vous voulez l'utiliser avec *coc.vim*, assurez vous de mettre `"diagnostic.displayByAle": true` dans votre `:CocConfig` comme mentionné dans le dépôt Github de ALE. Le repo Github est par ici : <https://github.com/dense-analysis/ale>

Chapitre 6

Pense-bête et exemples

Nous venons de faire un tour d'horizon de tout ce qui est nécessaire pour bien commencer dans la vie avec *Vim*. Tout cela devrait être suffisant pour pouvoir l'utiliser au quotidien. C'est le secret de la réussite avec *Vim* : réussir à l'enclencher dans nos habitudes journalières. Une fois que cela est fait, le reste devrait couler de source.

Cette dernière partie est là pour vous donner un endroit de référence où vous pourrez revenir comme bon vous semble lorsque vous serez un peu perdu sur comment faire telle ou telle chose avec *Vim*. Ce chapitre est composé de deux parties. La première est un ensemble de questions réponses qui couvre les principaux problèmes que les débutants rencontrent lorsqu'ils commencent. Le but est de répondre aux questions du type : « rha mais comment on fait ça, c'était pourtant si simple avec mon ancien éditeur ». La seconde partie est une liste (non exhaustive) des commandes *Vim* les plus utiles dont vous pourrez vous servir comme pense-bête. Allez hop, au boulot.

6.1 Questions / réponses

6.1.1 Comment quitter Vim ?

La première chose à faire est de se mettre en mode normal. Grossièrement, tapez la touche `Esc` (Échap) ou la touche `;` en fonction de votre configuration et vous devriez vous retrouver en mode normal. Ensuite tapez `:q` pour quitter. Il y a de grandes chances que *Vim* ne vous laisse pas faire. Si vous avez des modifications non enregistrées par exemple, il ne voudra pas quitter. Vous pouvez annuler les modifications en le forçant à quitter grâce à l'utilisation de `!` comme ceci : `:q!`. Vous pouvez aussi enregistrer vos modifications puis quitter comme ceci : `:wq`.

6.1.2 Comment sauvegarder sous ?

En mode normal, si vous tapez `:w`, *Vim* par défaut sauvegarde vos modifications dans le fichier courant. Si vous souhaitez utiliser un autre nom de fichier pour « sauvegarder sous », vous avez juste à lui spécifier le nom du fichier après `w` comme ceci : `:w monfichier.txt`. *Vim* sauvegardera alors votre fichier sous le nom *monfichier.txt*. En revanche *Vim* n'ouvrira pas *monfichier.txt*, il restera sur votre précédent fichier.

Si vous souhaitez que *Vim* sauvegarde sous *monfichier.txt* et ouvre ensuite ce fichier dans le tampon courant, vous devrez utiliser `:sav monfichier.txt`.

6.1.3 Comment copier/couper coller ?

Celle là est facile, j'y ai déjà consacré un chapitre, cf. [Se déplacer par l'exemple : Essayer de copier / coller.](#)

En résumé :

- Passez en mode visuel avec la touche `v`,
- Sélectionnez ce que vous voulez copier en vous déplaçant,
- Copiez avec la touche `y` ou couper avec la touche `x` ou la touche `d`,
- Collez après l'emplacement du curseur avec la touche `p` ou avant l'emplacement du curseur avec la touche `P`.

6.1.4 Comment créer un nouveau fichier ?

La façon traditionnelle de faire est de taper, en mode normal, `:e monfichier.txt` pour ouvrir un tampon (buffer) vide. Ensuite, sauvegardez votre tampon grâce à `:w`. Il sera sauvegardé sous le nom `monfichier.txt` dans le répertoire courant.

Vous pouvez aussi utiliser Lusty Explorer (cf. [Naviguer sur le disque et entre les fichiers : Lusty Explorer](#)) pour ce faire. Lancez le grâce à `,lr` ou `,lf`, tapez le nom du fichier que vous souhaitez créer puis appuyez sur la touche `Control` puis en même temps la touche `e`. Vous pouvez ensuite le sauvegarder de la même manière que ci-dessus.

6.1.5 Annuler / Refaire

Pour annuler il suffit d'utiliser la touche `u` en mode normal. Pour annuler le annuler (donc refaire) maintenez la touche `Control` appuyée puis la touche `r`.

6.2 Pense-bête

6.2.1 Fichiers

Résultat attendu	Action	Commentaire
Sauvegarder	<code>:w</code>	w pour write (écrire)
Sauvegarder sous	<code>:w nomdefichier.</code>	Sauvegarde sous nomdefichier.txt mais n'ouvre pas nomdefichier.txt
Sauvegarder sous / ouvre	<code>:sav nomdefichier</code>	Sauvegarde sous et ouvre nomdefichier.txt
Quitter sans sauvegarder (forcer à quitter)	<code>:q!</code>	
Sauvegarder et quitter	<code>:wq</code>	wq pour write (écrire) and quit (quitter)

6.2.2 Déplacements

Résultat attendu	Action
Se déplacer d'un caractère à gauche	<code>h</code>
Se déplacer d'un caractère en bas	<code>j</code>
Se déplacer d'un caractère en haut	<code>k</code>
Se déplacer d'un caractère à droite	<code>l</code>
Se déplacer à la fin d'un mot	<code>e</code>
Se déplacer au début d'un mot	<code>b</code>
Se déplacer au début du mot suivant	<code>w</code>
Se déplacer à la ligne 42	<code>:42</code>
Se déplacer au début du fichier	<code>gg</code> ou <code>:0</code>
Se déplacer à la fin du fichier	<code>GG</code> ou <code>:\$</code>
Se déplacer à la fin de la ligne	<code>\$</code>
Se déplacer au premier caractère non vide de la ligne	<code>^</code>
Se déplacer au début de la ligne	<code>0</code>
Descendre d'une page	<code>Ctrl+f</code>
Monter d'une page	<code>Ctrl+b</code>
Se déplacer à la première ligne de l'écran	<code>H</code>
Se déplacer au milieu de l'écran	<code>M</code>
Se déplacer à la dernière ligne de l'écran	<code>L</code>

6.2.3 Édition de texte

Résultat attendu	Action	Commentaire
Insérer avant le curseur	i	
Insérer avant le premier caractère non vide de la ligne	I	
Insérer après le curseur	a	
Insérer à la fin de la ligne	A	
Insérer une nouvelle ligne en dessous	o	
Insérer une nouvelle ligne au dessus	O	
Remplace le reste de la ligne	C	
Remplace un seul caractère (et reste en mode normal)	r	
Supprime le caractère après le curseur (comme la touche suppr.)	x	
Supprime le caractère avant le curseur (comme la touche backspace)	X	
Supprime la ligne courante	dd	
Copie la ligne courante	yy	
Colle après le curseur. Si c'est une ligne, colle la ligne en dessous.	p	
Colle avant le curseur. Si c'est une ligne, colle la ligne au dessus.	P	
Intervertit la casse des caractères (majuscules / minuscules)	~	Marche en mode visuel
Déplace le texte vers la droite (indentation)	>	Marche en mode visuel
Déplace le texte vers la gauche	<	Marche en mode visuel
En mode visuel, supprime la sélection	d	Mode visuel
En mode visuel, remplace la sélection	c	Mode visuel
En mode visuel, copie la sélection	y	Mode visuel
Annuler (Undo)	u	
Refaire (Redo)	Ctrl+r	

6.2.4 Chercher et/ou remplacer

Résultat attendu	Action	Commentaire
Rechercher	/*toto	Cherche la chaîne de caractères <i>toto</i> à partir de l'emplacement courant du curseur
Suivant	n	Affiche le prochain résultat de recherche
Précédent	N	Affiche le précédent résultat de recherche
Remplacer sur la ligne courante	:s/toto/t	Remplace <i>toto</i> par <i>titi</i> sur la ligne courante (une fois)
Remplacer tout sur la ligne courante	:s/toto/t	Remplace <i>toto</i> par <i>titi</i> sur la ligne courante (pour toutes les occurrences de <i>toto</i>)
Remplacer dans toutes les lignes	:%s/toto/	Remplace <i>toto</i> par <i>titi</i> sur toutes les lignes du fichier (une fois par ligne)
Remplacer tout dans toutes les lignes	:%s/toto/	Remplace <i>toto</i> par <i>titi</i> sur toutes les lignes du fichier (pour toutes les occurrences de <i>toto</i> par ligne)
Remplacer sur la ligne courante ignorant la casse	:s/toto/t	Remplace <i>toto</i> par <i>titi</i> sur la ligne courante (une fois)
Remplacer tout sur la ligne courante ignorant la casse	:s/toto/t	Remplace <i>toto</i> par <i>titi</i> sur la ligne courante (pour toutes les occurrences de <i>toto</i>)