Soyons honnête, en mode "découverte", VIM n'est pas l'outil le plus sexy qu'il soit. On a même l'impression qu'il est conçu pour être moche et ne pas faire grand chose de plus qu'un notepad compliqué par pur plaisir.

La raison fondamentale de cet état de fait est que VIM est un outil conçu pour s'adapter au plus large éventail de besoin possible (sans pour autant faire le café, et encore...). Au départ ce n'est donc qu'une toile vierge, un peu comme un FireFox brut de coffrage avant que vous y installiez vos extensions favorites. Comme FireFox, VIM est conçu pour être programmé et/ou étendu en piochant dans sa très dense collection de plugins. Mais à la différence de FireFox, VIM ne s'écroulera que difficilement sous le poids de ses propres plugins ;-)

Premiers paramétrage

En réalité le terme de paramétrage est quelque peu erroné. En effet, VIM ne se configure pas par le biais d'un quelconque fichier xml ou ini, mais se programme à travers un langage, le VIMScript. Ce langage de script comporte tous les éléments d'un langage classique : variables, assignation, boucles, conditions, fonctions, modules, etc. Maintenant rassurez-vous, il n'est pas nécessaire de maîtriser ce langage pour pouvoir utiliser VIM car dans un premier temps il s'agira essentiellement de modifier les valeurs des variables système pour faire coller l'éditeur à vos besoins.

Pour charger ces réglages, VIM exécute à chaque lancement le code contenu dans le fichier `~/.vimrc`. Prenons un exemple simple et disons que nous désirons que VIM active la numérotation des lignes, ce qu'il ne fait pas par défaut. Nous allons pour cela créer notre premier fichier `.vimrc` et y placer le contenu suivant en tapant `vim ~/.vimrc` :

```
set nocompatible
set number
```

mon premier vimrc

Sans sortir de VIM, vous pouvez tester ce code en tapant `:w|source %` (`w` pour sauvegarder `source` pour exécuter le code du fichier courant `%`).

Dans ce code nous avons utilisé la fonction système `set` pour affecter des valeurs. `set number` affecte la valeur `vrai` à `number`. Si nous voulions faire l'inverse, nous aurions écrit `set nonumber`.

La variable `compatible` indique un compatibilité avec l'ancêtre de VIM, le célèbre VI. Aujourd'hui on préfèrera mettre cette valeur à faux via set nocompatible pour activer un ensemble de réglages plus actuels.

La commande `:help` documente absolument toutes les variables. Essayez `:help number` ou `:help compatible` pour plus d'information.

Une fois le fichier sauvegardé, il suffit de quitter VIM, puis que votre premier paramétrage soit définitif.

Nous n'allons pas étudier ici tous les réglages possibles pour VIM, ce serait bien trop long. Sachez donc trouver les solutions à vos besoins grâce à votre automate de recherche préféré. Vous découvrirez alors rapidement que nous sommes très nombreux à utiliser cet outil. Par exemple si vous désirez activer la coloration syntaxique, chercher "vim activate syntax highlighting" et vous aurez moulte réponses qui vous mèneront à ajouter à votre vimrc les lignes suivantes :

```
filetype plugin indent on
syntax on
```

coloration syntaxique

Sauvez et éditez à nouveau, miracle, votre vimrc est maintenant colorisé.

Le langage VIMScript permet de faire de nombreuse choses et pour qui a à cœur de personnaliser son environnement à fond, sa maîtrise n'est pas un luxe. Une ressource très précieuse sur ce point est l'excellent tutoriel d'IBM.

Structure

Le paramétrage de VIM ne se limite pas au simple fichier `.vimrc`. Il est aussi possible d'exploiter toute une arborescence de dossiers nichés sous `~/.vim` (non créé par défaut). Les principaux sont :

autoload On trouve dans ce dossier des scripts qui vont être chargés au démarrage, avant le fichier `~/.vimrc`. colors Ici sont stockés des scripts appelés par la commande `colorscheme` permettant de charger un jeu de couleur. syntax Là se trouvent des scripts gérant la colorisation syntaxique pour un type de fichier donné sous la forme `type_de_fichier.vim`. plugin Dans ce dossier sont stockés les plugins de VIM. doc Les fichiers documentation des plugins utilisables par la commande `:help`. spell Pour les dictionnaires du correcteur orthographique. Pour une description complète et exhaustive de tous les autres dossiers, utilisez la commande `help runtimepath`.

Nous le verrons plus loin, cette liste n'est pas exhaustive car certains plugins vont ajouter ici leur propres sous-dossiers. Alors dans la mesure où `~/.vim` va vite devenir le centre névralgique de votre configuration, je vous conseille la manipulation suivante :

```
gastoncd
gastonmkdir -p .vim/{autoload,colors,syntax,plugin,spell,config}
gastonmv .vimrc .vim/vimrc
gastonln -s .vim/vimrc .vimrc
```

Initialisation de sa configuration VIM

Ceci a pour but de pré-créer `.vim` et un ensemble de sous-dossiers. D'y déplacer notre `.vimrc` en un fichier visible (sans point devant) et de faire un lien symbolique pour permettre à vim de le démarrer en ne se rendant compte de rien. Ainsi toute votre configuration sera à un seul et même endroit.

Pour parfaire le dispositif, vous noterez la création du sous-dossier `config`. Il ne s'agit pas d'un dossier de VIM mais d'un endroit où nous allons stocker des bouts de vimrc. En effet, vous vous en rendrez vite compte, vimrc peut rapidement devenir titanesque. A titre d'exemple le mien fait plus de 1000 lignes de code... Du coup, il est bien pratique de pouvoir ventiler cette configuration dans plusieurs fichiers thématiques (ex. settings.vim, mappings.vim, etc.).

Pour que cela fonctionne, il faut que nous rajoutions à vimrc le code permettant de lire ce dossier config :

```
runtime! config/**/*.vim
```

Lecture des scritps contenus dans .vim/config

Ceci fait, vous pouvez déplacer dans un `.vim/config/settings.vim` tout ce qui se trouve après set nocompatible (il est préférable de laisser cette ligne en en-tête de vimrc).

A plugin to rules them all

Comme vous le savez sûrement, VIM dispose d'un stock impressionnant de plugins. Il sera en effet bien rare de ne pas trouver le plugin correspondant à l'un de vos besoin.

Techniquement, un plugin est fournie sur un site comme vim.orgsous la forme d'une archive zip à décompresser à la racine de `.vim` provoquant l'ajout de fichiers dans un ou plusieurs sous dossiers (plugin, doc, autoload, etc.). A terme je vous laisse imaginer le foutoir... Fort heureusement il y a une solution à ce problème. Il y en a même plusieurs en réalité, mais celle que j'ai retenu s'appelle pathogen.

Le principe de pathogen est très simple. Il s'agit de créer un nouveau dossier `.vim/bundle` dans lequel chaque plugin aura son propre dossier dans lequel nous retrouverons `plugin`, `doc`, `autoload`, etc. Pathogen se chargera alors, au démarrage de VIM, d'explorer ce dossier et d'instruire VIM sur tous les fichiers à charger. Ainsi tout reste bien rangé.

Pour mettre en place pathogen, il faut d'abord le télécharger. Comme pour absolument tous les plugins pour VIM, le plus simple est clairement de passer par GIT via GitHub. Pour ceux qui découvrent, GIT est un gestionnaire de version et github une plateforme hébergeant de très nombreux projets libres et bâti autours de GIT.

Même pour si le développeur d'un plugin n'utilise pas github, le site vim.org effectue un mirroring automatisé de tous ses plugins sur github. Ainsi si vous cherchez la version "github" d'un plugin, il suffit généralement de taper dans votre moteur de recherche "github vim le_plugin_que_je_cherche" pour le trouver. Ainsi toute installation de plugin passera systématiquement par github, ce qui est extrêmement pratique.

La première chose à faire est donc d'installer git sur votre système, cela se fait sur debian par `sudo apt get git`, tout simplement. Ceci fait, allez sur la page du projet pathogen. Dans la partie du haut, vous trouverez une URL qu'il suffit d'utiliser comme ceci

```
gastoncd ~/.vim
gastongit clone https://github.com/tpope/vim-pathogen.git pathogen
Cloning into pathogen...
remote: Counting objects: 225, done.
remote: Compressing objects: 100% (120/120), done.
remote: Total 225 (delta 61), reused 209 (delta 48)
Receiving objects: 100% (225/225), 27.14 KiB, done.
Resolving deltas: 100% (61/61), done.
gastoncd autoload
gastonln -s ../pathogen/autoload/pathogen.vim .
```

Récupération du code source de pathogen

Nous sommes allé dans notre dossier `.vim` pour y cloner (comprendre récupérer) le code source de pathogen dans un dossier `pathogen`. Ensuite nous avons créé le dossier `autoload`, y sommes rentré, puis avons créé un lien symbolique à partir du fichier `../pathogen/autoload/pathogen.vim`.

Nous aurions aussi pu cloner le dépôt n'importe où et ensuite juste recopier `pathogen.vim` dans le dossier `autoload` mais en procédant par lien symbolique, nous pourrons facilement bénéficier des éventuelles mises à jour de pathogen de la manière suivante :

```
gastoncd ~/.vim/pathogen
gastongit pull
```

Mise à jour de pathogen

Maintenant il ne reste plus qu'à ajouter quelque ligne de code à vimrc, **juste après** `set nocompatible` pour que la magie pathogen opère :

```
" Initialisation de pathogen
call pathogen#infect()
call pathogen#helptags()
```

Ajout de la gestion des plugins par pathogen

Et c'est tout.

Pour tester tout cela, installons notre premier plugin que je classe dans la catégorie incontournable, NerdTree. Sont rôle est d'ajouter à VIM un volet latérale permettant d'explorer les fichiers et sous-dossiers du dossiers courant. Comme pour pathogen, nous allons passer simplement par la page du projet GitHub de NerdTree.

```
gastoncd ~/.vim
gastonmkdir -p bundle
gastoncd bundle
gastongit clone https://github.com/scrooloose/nerdtree.git nerdtree
Cloning into nerdtree...
remote: Counting objects: 2396, done.
remote: Compressing objects: 100% (805/805), done.
remote: Total 2396 (delta 1076), reused 2314 (delta 999)
Receiving objects: 100% (2396/2396), 979.05 KiB | 485 KiB/s, done.
Resolving deltas: 100% (1076/1076), done.
```

Installation de NerdTree

Et... C'est tout :-) Il ne nous reste qu'à redémarrer VIM et à taper la

commande `:NERDTree` pour disposer d'un navigateur du plus bel effet.

Conclusion

Comme vous le voyez, VIM peut rapidement devenir très personnalisé tant par ajout de paramétrage, que par celui de plugins (personnellement je tourne avec déjà 26 plugins...). Alors évidement, cette approche ne conviendra pas à tous ceux qui veulent "que ça marche". Mais pour les autres, ceux qui aiment construire leur environnement, VIM est une véritable bouffée d'oxygène.