

Vim, le Vi amélioré : votre seul éditeur de texte, tout simplement

Denis Bodor



NOTE POLÉMIQUE

Vi ou Emacs. De longue date, mais de nos jours peut-être un peu moins en raison de l'utilisation triste et massive d'éditeurs de texte graphiques (remarquable antinomie), Vi et Emacs ont toujours été les deux églises rivales de la sainte édition de texte et de code.

D'un côté comme de l'autre, une partie des utilisateurs adore détester le logiciel d'en face. Ainsi, les utilisateurs de Vi n'ont pas besoin d'Emacs, car ils disposent déjà d'un système d'exploitation. Les utilisateurs d'Emacs, quant à eux, connaissent Vi, puisqu'ils savent qu'il s'agit d'un éditeur modal, un mode où il est impossible de faire quoi que ce soit et où l'éditeur fait bip.

Emacs ou Vi ? C'est une affaire de goût et de préférences personnelles. Moi, j'utilise Vi (Vim) et si vous ne trouverez pas d'article sur Emacs dans le présent magazine, c'est en partie pour cette raison. Personne n'ayant apporté de contribution sur Emacs, je ne pouvais pas, raisonnablement, apporter un avis objectif sur ce syst[^]Wlogiciel.

S'il est un classique dans le monde Unix, c'est bien l'éditeur Vi. Ce n'est pas un hasard si une version de Vi est fournie de base avec tous les systèmes Unix qu'il s'agisse de GNU/Linux, *BSD ou encore Mac OS X. L'éditeur de texte est l'outil indispensable de tout utilisateur de la ligne de commandes. Comme son nom l'indique, il permet l'édition de texte pure. Il n'est pas question ici de mise en forme de quelque manière que ce soit. Un éditeur de texte n'est pas un traitement de texte. Son travail consiste à fournir à l'utilisateur un certain nombre de fonctionnalités facilitant l'édition de fichiers de configuration, de scripts (*shell*, Perl, etc.), de fichiers sources, de fichiers HTML...

Pour la petite histoire, la quasi-totalité des articles du présent magazine ont été écrits avec l'éditeur de texte Vim au format HTML/CSS, puis mis en page avec une application spécialisée.

Malheureusement, la version de Vi fournie par défaut avec le système GNU/Linux lors de son installation n'offre pas ces fonctionnalités bien avancées. Il s'agit, le plus souvent d'Elvis, nvi ou encore Vim-Lite. On choisira donc d'installer au plus vite une version plus performante en utilisant les outils de gestion de paquets de sa distribution. Il existe plusieurs versions modernes de Vi (prononcez Vi Aïe), parmi lesquelles on trouve le logiciel qui nous intéresse aujourd'hui : Vim.

Vim est la compression des termes *Vi IMproved* (Vi amélioré). C'est une version de Vi écrite et maintenue par Bram Moolenaar. Vim est disponible sous les termes de la licence libre GNU GPL. La dernière version en date est la 7.1. Celle-ci comprend un grand nombre d'améliorations par rapport au Vi traditionnel datant originellement de 1978. Sachez également que Vim est disponible pour des dizaines de systèmes.

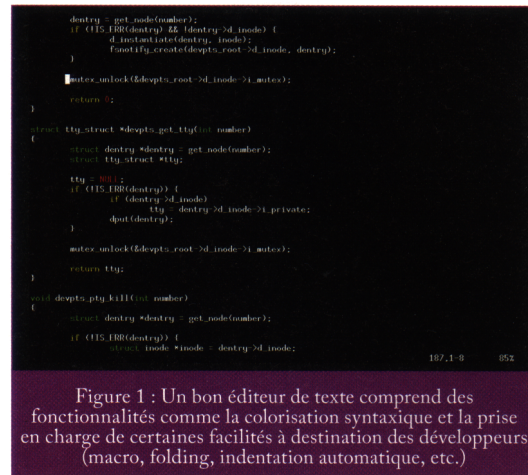


Figure 1 : Un bon éditeur de texte comprend des fonctionnalités comme la colorisation syntaxique et la prise en charge de certaines facilités à destination des développeurs (macro, folding, indentation automatique, etc.)

Il s'agit pour la plupart de systèmes Unix, mais on trouve également des versions Windows, par exemple. La disponibilité d'un éditeur de texte sur un grand nombre de plateformes est une chose importante. Un véritable éditeur performant nécessite un certain temps d'apprentissage. Si l'éditeur existe sur plusieurs systèmes et qu'il est disponible sous une licence libre, il vous suffira d'installer la version correspondante pour immédiatement retrouver vos marques.

Avant de nous lancer dans nos premières manipulations de base avec Vim, résumons en quoi un éditeur de texte peut être avantageux :

- Il doit permettre de rapidement écrire ou modifier un texte. Il doit donc démarrer rapidement et être immédiatement utilisable.
- Il doit permettre une modification rapide du texte ce qui implique des capacités à se placer à un endroit donné d'un texte via un minimum de commandes.
- Il doit faciliter la lecture du texte durant la rédaction. Les exemples ne manquent pas. Il s'agit de garder en vue une partie de texte en éditant une autre ou encore de lire un fichier en en modifiant un autre.
- Il doit faciliter la réorganisation d'un texte en gérant rapidement et facilement les copier/coller, les déplacements de paragraphes, etc.

Un bon éditeur de texte, comme c'est le cas pour Vim, excellera dans chacun de ces domaines. Enfin, un bon éditeur de texte est... en mode texte et se lance depuis une ligne de commandes. Il existe une version graphique de Vim (support GTK+), mais tout comme le trop utilisé GEdit, je trouve qu'il y a quelque chose de malsain au simple fait de taper `gedit /etc/inittab`, par exemple. Pourquoi donc lancer un éditeur dans



une fenêtre alors que le terminal (Xterm ou autre) est là pour ça ? Le véritable utilisateur Unix comme le développeur est fainéant, car il n'a pas d'énergie à dépenser dans des choses inutiles.

Notions de base avec Vi/Vim

Avant de manipuler pour la première fois un Vi, vous devez comprendre des principes de base communs à toutes les versions, même les plus anciennes. La première notion est celle de « mode ». Quel que soit le Vi que vous utilisez, une chose est absolument invariable : les deux modes d'utilisation. Vi est un éditeur modal qui peut se trouver dans deux « états ». Contrairement à un traitement de texte ou un éditeur comme GEdit, Vi sépare l'édition pure du texte des fonctionnalités concernant tout le logiciel.

Le premier « état » est appelé le « mode commande ». Il sert à donner des ordres à l'éditeur (ouverture de fichier, sauvegarde, recherche, etc.). Lorsque vous désirez modifier le texte, vous devez passer dans un autre mode : le mode insertion. Il est important de bien comprendre ce principe de base. Car, en fonction du mode dans lequel vous vous trouvez, l'éditeur réagira de manière différente aux touches et aux séquences de touches composées au clavier. Un très bon exemple est **dd**. En mode insertion (édition du contenu), deux pressions successives sur la touche [d] provoqueront l'insertion de **dd** dans le texte édité. En mode commande, par contre, **dd** permet de supprimer une ligne complète.

En mode commande, les ordres donnés prennent la forme d'un ou deux caractères. Tantôt, ces ordres (ou commandes) sont accompagnés d'arguments comme un nom de fichier ou une valeur spécifique à utiliser. Notez que le mode commande est sensible à la casse des caractères. Ainsi, la commande **x** (effacer le caractère sous le curseur) est différente de **X** (effacer le caractère AVANT le curseur). En mode commande, ce que vous tapez au clavier n'apparaît pas à l'écran (comme le **dd** de notre exemple précédent). Il n'est pas nécessaire non plus de valider les commandes avec la touche [Entrée].

Dernier point important concernant ce mode, il est possible de lancer des ordres spécifiques destinés à l'éditeur sous-jacent à l'interface de Vi qui n'est que la partie visible du logiciel. Ces commandes sont appelées « commandes **ex** ». Toutes les commandes **ex** commencent par un caractère double-point (:). Dans ce cas précis, les commandes apparaissent à l'écran et il est utile de les valider. On pourrait penser qu'il s'agit d'un troisième mode, mais non. Il s'agit bien du mode commande. Inutile de vous tracasser avec **ex**, sachez simplement que lorsqu'une commande commence par un double-point, il s'agit d'une commande **ex**.

Par défaut, après son lancement, Vim est en mode commande. Ce que vous tapez n'est donc pas ce qui arrivera dans le texte édité, également appelé « tampon » ou « *buffer* ». Pour passer en mode édition, vous utiliserez **i** (comme *insert*). Selon la configuration de votre Vi,

vous verrez ou non apparaître le mot **INSERT** au bas de l'écran. Dès lors, les caractères que vous taperez seront insérés dans le texte édité. Pour sortir du mode d'édition (et donc retourner en mode commande), appuyez simplement sur la touche [Echap] du clavier.

L'autre notion importante avec Vi concerne les unités de mesure. Alors qu'avec un traitement de texte cette notion est purement subjective, avec Vi, une ligne, un paragraphe ou un mot sont des éléments aux caractéristiques bien précises. Un mot est une chaîne de caractères continue sans espace, ni blanc. Une ligne est une suite de mots comprise entre un début de ligne et un retour à la ligne. Comprenez bien ceci : une phrase occupant plusieurs lignes de l'écran ne forme qu'une seule ligne avec Vi. Dans la pratique, en vous plaçant en début de ligne, vous arriverez en début de phrase. De la même manière, en vous plaçant en fin de ligne, vous arriverez sur ou après le dernier caractère de la phrase. Ceci est très différent du comportement d'un traitement de texte où les lignes sont simplement des morceaux de texte entre la marge de gauche et de droite. Enfin, un paragraphe est une masse de texte (plusieurs lignes) séparée des autres paragraphes par une ligne vide.

Première manipulations

Si les explications qui viennent d'être données vous paraissent obscures, nous allons éclaircir tout cela avec quelques manipulations. Vim se lance avec la commande... **vim**. Vous pouvez passer en paramètre le nom d'un fichier. Si celui-ci existe, il sera ouvert. Dans le cas contraire, un nouveau fichier sera peut-être créé (si vous l'enregistrez par la suite). Lorsqu'on ouvre un fichier, Vim le charge dans un tampon (buffer) et l'affiche à l'écran pour vous permettre d'y faire des modifications. Lorsque vous sauvegardez le fichier, Vim enregistre le tampon sur le disque et écrase l'ancien fichier.

Commençons donc par éditer notre premier fichier :

```
% vim premier
```

Vous voici dans **Vim** et en mode commande. Vous devez voir apparaître en bas de l'écran un message **premier [Nouveau fichier]**, vous indiquant qu'il s'agit d'un nouveau fichier. À présent, passez en mode insertion avec la commande **i**, puis tapez un texte quelconque. Enfin, repassez en mode commande avec la touche [Echap].

Vous pouvez, à présent, enregistrer les changements que vous venez d'apporter au tampon dans le fichier **premier** et quitter avec la commande **ZZ**. Notez qu'il s'agit de majuscules et que les caractères n'apparaissent effectivement pas. Vous voici revenu à l'invite du shell.

La commande **ZZ** n'est pas la seule manière de procéder. Voici quelques synonymes :

► **:x** sauvegarde et quitte ;

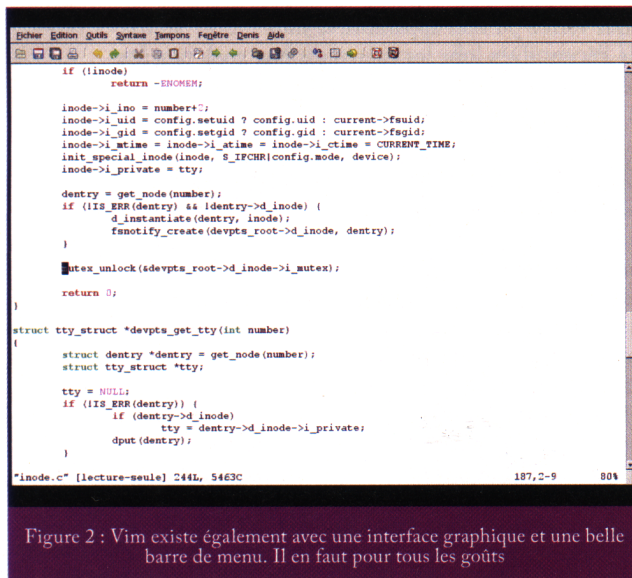


Figure 2 : Vim existe également avec une interface graphique et une belle barre de menu. Il en faut pour tous les goûts

- **:wq** sauvegarde et quitte ;
- **:W** sauvegarde sans quitter.

Cette dernière commande peut prendre en argument le chemin et le nom d'un éventuel fichier dans lequel enregistrer le tampon en cours. Ainsi, la commande suivante enregistrera les changements dans **nouveau_fichier** et non dans **premier** :

```
:w /chemin/nouveau_fichier
```

Personnalisation de Vim

Avant de poursuivre dans la découverte de ce fantastique éditeur, nous allons paramétrer Vim de manière à le rendre plus facilement utilisable. Historiquement, Vi ne connaît pas les touches de direction du clavier. En effet, à l'époque des premières versions de l'éditeur (1978), les ordinateurs ne possédaient tout simplement pas ces touches. On utilisait **h**, **j**, **k** et **l** (respectivement gauche, bas, haut et droite) en mode commande.

En démarrant Vim avec sa configuration standard, il respectera le mode de fonctionnement traditionnel de Vi (c'est dépendant de la configuration par défaut stockée dans **/etc/vim/vimrc**). Ceci devient vite lassant. D'autant plus que Vim permet une gestion du clavier et de l'écran bien plus ergonomique que le traditionnel (préhistorique ?) Vi.

Nous allons donc optimiser quelque peu Vim en créant un fichier de configuration. Rassurez-vous, ce n'est rien de bien complexe. Placez-vous dans votre répertoire personnel et utilisez la commande suivante pour créer le fichier de configuration (configurer Vim avec Vim a quelque chose de magique, non ?) :

```
% vim .vimrc
```

Le fichier **.vimrc** sera lu à chaque démarrage de Vim. Les instructions que nous allons y placer seront traitées à ce moment tout comme si nous les avions tapées en mode commande à destination d'**ex** (précédées d'un double-point). Passez en mode insertion en appuyant sur **i**, puis saisissez :

```
set nocompatible
set backup
set history=50
set ruler
```

Une fois cette dernière ligne saisie, faites **[Echap]** : **wq** pour quitter en enregistrant les changements. Vous pouvez ensuite relancer Vim et constater les améliorations. La première ligne du fichier de configuration spécifie à Vim de ne pas être compatible historiquement avec Vi et de permettre ainsi l'utilisation de fonctionnalités nouvelles (flèche du clavier, affichage du mode en bas de l'écran, etc.). La seconde directive nous permet de systématiquement créer un fichier de sauvegarde. Lorsque vous enregistrerez des changements dans un fichier, Vim gardera une version précédente sous le même nom suivie d'un tilde (~). La troisième directive permet de garder un historique des 50 dernières commandes. Enfin, **set ruler** permet d'afficher la position du curseur en bas de l'écran dans les deux modes de Vi (ligne/colonne).

Poursuivons les manipulations

Notre fichier **premier** contient maintenant du texte. Nous pouvons donc le rééditer pour y apporter quelques modifications. Une fois dans Vim, passez en mode insertion. Mais, cette fois, avec la commande **R**. Ce que vous taperez maintenant à l'endroit où se trouve le curseur écrasera le texte. Vous remplacez les caractères (c'est le mode reffappe des traitements de texte). Vous pouvez repasser en insertion en faisant **[Echap]**, puis **i**. Certains Vi, dont Vim, permettent de basculer de l'insertion à l'écrasement avec la touche **[Insert]**.

Vous venez de modifier le tampon, mais, comme ces modifications ne sont pas enregistrées, plusieurs solutions s'offrent à vous. Vous pouvez, bien sûr, les enregistrer avec **ZZ** ou **:wq**. Dans le cas où ces changements s'avèrent inopportuns, vous pouvez utiliser la commande **:e!** qui vous permettra d'annuler tout changement et de revenir à la dernière version enregistrée. Vim rechargera le fichier d'origine dans le tampon. Si vous désirez annuler ces changements et, par la même occasion, quitter Vim, vous utiliserez alors **:q!**. Le point d'exclamation permet de forcer Vim à exécuter un ordre. En effet, si le tampon est modifié et non enregistré, l'appel de la commande **:q** provoquera un message d'avertissement signalant que vous êtes sur le point de perdre les modifications apportées au tampon.

Autre caractéristique importante concernant l'ouverture ou l'enregistrement des fichiers, le caractère **%** possède



une signification particulière. En effet, % représente le nom courant du tampon, c'est-à-dire le nom du fichier que vous avez originellement ouvert. Ainsi, en utilisant la commande :

```
:w %2
```

vous enregistrez vos modifications sous le nom **premier2** dans le répertoire courant. % étant **premier**, %2 nous donne **premier2**.

Utilisation plus avancée

Commençons nos manipulations plus élaborées avec quelque chose de simple (simplement élaboré donc) : les déplacements du curseur. Déplacer le curseur dans le texte est quelque chose de relativement simple avec Vim à partir du moment où l'on a assimilé certaines règles.

De base, **h**, **j**, **k** et **l** permettent respectivement un déplacement du curseur d'un caractère à gauche, en bas, en haut et à droite. Notez la position de ces touches sur le clavier... elles se suivent. Bien sûr, vous pouvez maintenant utiliser les habituelles touches de direction. Peu importe le mode dans lequel vous vous trouvez, un déplacement vertical dans le texte se fera ligne par ligne. Une ligne dans Vi est tout le texte contenu entre le premier caractère et le retour chariot final. Ainsi, une phrase faisant visuellement 5 lignes de texte à l'écran n'en fera réellement qu'une seule. Un déplacement vertical d'une ligne (avec **j** ou **k**) déplacera donc visuellement le curseur de 5 lignes. Faites l'essai sur un gros fichier, vous comprendrez rapidement.

En dehors des déplacements simples « au caractère », il existe tout un jeu de touches permettant un déplacement bien plus efficace. En voici quelques-unes :

- ▶ **w** déplace le curseur au début du mot suivant ;
- ▶ **b** déplace le curseur au début du mot courant ou précédent ;
- ▶ **e** déplace le curseur à la fin du mot courant.

Plus intéressant, lors de la rédaction d'un texte, les touches **)** et **(** vous permettent respectivement de déplacer le curseur au début de la phrase courante/précédente ou de la phrase suivante. Une phrase est une suite de caractères se terminant par un point suivi d'une espace.

Quelles que soient les touches de déplacement que vous utilisez, vous pouvez les faire précéder d'une valeur numérique. Celle-ci indiquera alors le nombre d'occurrences de la commande. De ce fait, dans le mode commande, si vous tapez **12l**, vous déplacerez 12 fois le curseur d'un caractère vers la droite. Essayez de saisir un texte de taille conséquente (même sans signification) et amusez-vous à faire des déplacements de la sorte. Vous acquerez sans doute rapidement des

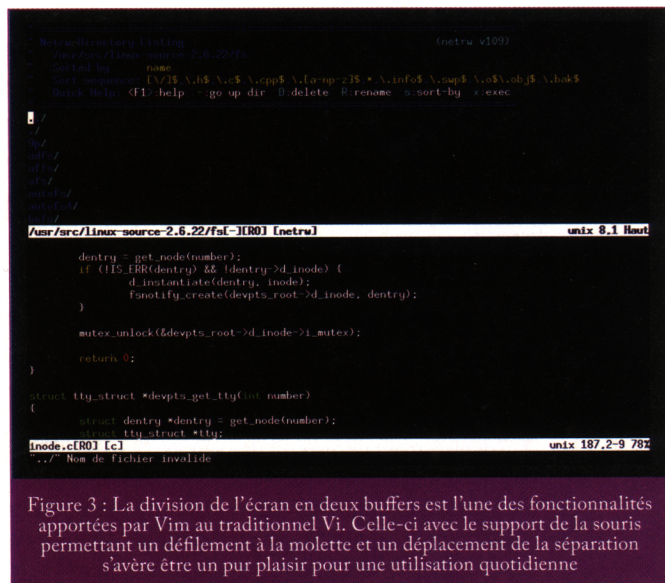


Figure 3 : La division de l'écran en deux buffers est l'une des fonctionnalités apportées par Vim au traditionnel Vi. Celle-ci avec le support de la souris permettant un défilement à la molette et un déplacement de la séparation s'avère être un pur plaisir pour une utilisation quotidienne

automatismes et, ainsi, vos déplacements seront déjà plus efficaces qu'avec un traitement de texte ou un éditeur comme Gedit.

Modification du texte

Qu'il s'agisse d'un fichier de configuration, de sources ou encore, plus simplement, d'un texte quelconque, il existe un grand nombre de modifications différentes que l'on peut apporter à un fichier.

Les suppressions

En dehors de l'habituelle touche « Effacement arrière » (*backspace*) qui s'utilise en mode insertion, Vim dispose d'un large panel de commandes d'effacement bien plus efficaces. Toutes les commandes de suppression débutent par **d** comme *delete*. Le second caractère de la commande renseigne Vi sur l'objet auquel s'applique la suppression :

- ▶ **dw** : efface les caractères d'un mot se trouvant après le curseur (notez que l'espace immédiatement après ce mot est également supprimée). Si votre curseur est positionné en début de mot, il sera supprimé dans son intégralité. En revanche, si le curseur se trouve à un endroit précis du mot, ce n'est que la fin du mot qui sera éliminé.

Dans le même esprit :

- ▶ **dE** : supprime le reste du mot avec la ponctuation ;
- ▶ **db** : supprime le début du mot (les caractères placés avant le curseur dans le mot) ;
- ▶ **d0** : supprime tous les caractères de la ligne placés avant le curseur ;
- ▶ **d\$** : supprime tous les caractères de la ligne placés après le curseur. La commande **D** est un raccourci pour **d\$**.

► **dd** : supprime la ligne complète (attention, il s'agit d'une ligne au sens Vi du terme).

Comme toutes les commandes, celles-ci peuvent être précédées d'une valeur numérique indiquant le nombre d'occurrences. Là encore, ces manipulations permettent de travailler bien plus vite qu'avec un éditeur de texte simpliste ou un traitement de texte. Imaginez simplement que vous vous trouviez au milieu d'une phrase et que vous souhaitiez en éliminer la fin. Avec un traitement de texte, vous serez obligé de prendre votre souris, de faire une sélection et de supprimer le texte sélectionné. Avec Vim, un simple **d\$** fera le travail.

Les déplacements

Un déplacement de texte est une suppression suivie d'un ajout (comme un couper/coller). Toutes les commandes de suppression précédemment citées permettent des déplacements de texte. Ainsi, une ligne supprimée avec **dd** pourra être placée après le curseur n'importe où dans le texte avec la commande **p**. Si vous préférez placer ce texte avant le curseur, utilisez **P**.

Attention : Seul le dernier texte supprimé ne pourra être placé avec **p** ou **P**. Si vous utilisez plusieurs fois une commande de suppression, ce n'est que le dernier objet texte qui sera collé. Les précédentes suppressions sont perdues.

Une fois encore, vous pouvez préciser le nombre d'occurrences de votre placement de texte. **3p** placera trois fois de suite le texte précédemment supprimé.

La copie

Rien de spécial ici, la copie de texte (*yanking*) se fait par les commandes **y** de la même manière que la suppression. Notez cependant que **Y** n'est pas un raccourci pour **y\$**, mais pour **yy** (copie de la ligne complète). Le texte copié peut ensuite être placé avec les commandes **p/P**.

Les annulations et répétitions

Les erreurs de saisie sont fréquentes chez les humains (si, si !). Il est donc normal que l'éditeur de texte permette l'annulation de la dernière modification. Dans ce cas, la commande **u** vous permettra d'annuler votre dernier changement. Avec le Vi classique, seule la dernière modification peut être annulée. Une seconde utilisation d'**u** annulera l'annulation :)

Les différents Vi modernes comme Vim ou nvi permettent de choisir un fonctionnement différent et de conserver un grand nombre de niveaux d'annulation.

Dans certains cas, il est également pratique de pouvoir répéter la dernière manipulation. Ce sera donc la commande **.** qui sera utilisée ou encore **:redo**.

Les répétitions avec argument numérique

Nous en avons déjà parlé, les commandes peuvent être précédées d'une valeur de répétition. Bien que cela soit

fort pratique pour une suppression ou un placement de texte, il est un domaine où cette fonctionnalité est encore plus utile.

Imaginez-vous en train d'éditer un fichier de configuration. Lorsqu'il s'agit d'un gros fichier, il est courant de placer judicieusement des commentaires afin de conserver une trace des modifications. Parfois, il est également utile de bien séparer chaque partie du fichier par une ligne composée, par exemple, de **#**. Quoi de plus lassant que de rester appuyé sur la touche en attendant d'obtenir le résultat escompté...

Avec Vim, rien de plus facile : Si vous désirez placer 25 occurrences d'un ou de plusieurs caractères, il vous suffit d'utiliser :

```
25i#[ECHAP]
```

Et vous obtiendrez automatiquement vos 25 dièses (**#**) à la position courante du curseur. **25i#** signifie 25 fois **i** (mode insert), puis dièse (comme caractère à saisir) et, enfin, **[ECHAP]** pour revenir au mode commande et appliquer la commande. Ceci peut être à double tranchant. En ne prenant pas garde au mode dans lequel vous vous trouvez en saisissant une valeur numérique, vous risquez d'obtenir des résultats très impressionnants, mais peu amusants.

Recherches simples

Fonctionnalité indispensable de tout éditeur de texte qui se respecte, la recherche est indispensable dans bien des cas. Imaginez-vous en train de parcourir les quelques 400 ou 500 lignes d'un fichier de configuration à la main pour y trouver l'option que vous désirez changer... insupportable ! Inacceptable ! Impensable !

La recherche simple de motif avec Vim se fait en utilisant la commande **/** (*slash* ou barre de fraction) pour une recherche vers le bas ou avec **?** pour une recherche vers le haut. Ainsi, en tapant **/coup[ENTREE]**, vous rechercherez le groupe de caractères « coup ». Faites bien attention aux espaces ! Avec une telle recherche, vous trouverez aussi bien le mot « coup », que « couper », « coupant », etc.

Vous pouvez répéter la recherche précédente vers le bas avec **/[ENTREE]** ou vers le haut avec **?[ENTREE]**. Sachez également que la commande **N** vous permet de répéter la recherche dans le sens opposé. Ainsi, si vous êtes en plein milieu d'un tampon et que la recherche vers la bas n'aboutit pas. Il vous suffit de taper **N** pour rechercher la même chose vers le haut.

Vous pouvez également combiner les opérations de recherche et de suppression avec, par exemple, une commande de suppression. Attention, aucune confirmation n'est demandée ! Ainsi, la commande :

```
d/coup
```



permet de supprimer la prochaine occurrence du groupe de caractères « coup » dans la recherche vers le bas. N'oubliez pas que **u** vous permet d'annuler une éventuelle erreur de manipulation.

Il existe également une version minimaliste de la recherche. Celle-ci correspond à la commande **f** et permet de rechercher un caractère dans la ligne courante. **fè** placera le curseur sur la prochaine occurrence du caractère **è**. Inversement **Fè** placera le curseur sur la précédente occurrence du caractère **è**.

Recherches et remplacements avancés

Nous venons de voir quelques fonctionnalités courantes de recherche avec un éditeur de texte. N'importe quel éditeur « bas de gamme » en fait de même, mais, ce qui fait la puissance d'un logiciel comme Vim dans ce domaine est la capacité d'utiliser beaucoup plus de critères dans la recherche. Prenons un exemple simple...

Supposons qu'après être arrivé au terme de la rédaction d'un texte, vous vous rendez compte d'une regrettable erreur : vous avez utilisé le terme « boîte » en lieu et place de « récipient » dans tout le texte. Pire encore, vous avez tantôt écrit ce mot « boîte » et tantôt « boite ». Comment donc rechercher ce terme dans ces différentes versions, et ce, au singulier comme au pluriel ? Pas d'inquiétude, c'est un jeu d'enfant avec Vim :

```
/bo.te*
```

Qu'avons-nous fait ? Nous venons de rechercher vers le bas un groupe de caractères dont les deux premières lettres sont « bo », le troisième n'importe quel caractère, suivi de « te », puis de n'importe quel nombre de caractères. **bo.te*** est un motif de recherche ou *pattern* en anglais. En résumé, nous cherchons **bo**(un caractère quelconque)**te**(plusieurs caractères quelconques).

Clarifions un peu tout cela. Le **.** (point) représente n'importe quel caractère unique n'étant pas une nouvelle ligne. Les espaces sont considérées comme des caractères. Le ***** représente n'importe quel nombre de caractères.

Mais cela ne convient pas vraiment. En effet, vous pouvez parfaitement avoir des mots dans votre texte répondant exactement aux mêmes critères de recherche (« *booter* » ou « *botte* ») par exemple. Dans le cas d'une recherche, ce n'est pas vraiment grave, mais, avec un remplacement, le résultat peut être catastrophique (surtout sans confirmation). Il nous faut donc être plus précis dans notre recherche en spécifiant des différentes alternatives pour ce mot. Réfléchissons, nous pouvons avoir un « i » ou un « î ». Nous pouvons donc spécifier cela dans notre recherche. Celle-ci sera alors :

```
/bo[iî]te*
```

Les crochets permettent de définir plusieurs alternatives pour un caractère. Ici **[iî]** signifie **i** ou **î**. Le résultat est bien meilleur cette fois.

Passons donc à l'étape suivante en procédant à nos remplacements :

```
:s/bo[iî]te*/récipient/gc
```

Pas de panique, cela est plus clair qu'il n'y paraît. Ce fonctionnement est très peu courant avec les éditeurs de textes classiques que vous connaissez sans doute déjà. Mais, il s'agit d'une simple habitude à prendre.

Cette fois-ci, nous utilisons une commande **ex** pour remplacer toutes les occurrences du mot. Voyons cela dans le détail :

► **%** est un caractère très utile pour ce genre de manipulation. Il s'agit en fait d'un raccourci pour **1,\$** qui signifie « depuis la ligne 1 jusqu'à la fin du fichier ». Plutôt que de spécifier le nombre d'occurrences de la commande, nous précisons sa portée. L'unité de mesure est la ligne. Ainsi, si nous avions voulu que notre commande de remplacement ne porte que sur les 10 premières lignes, nous aurions pu faire **:1,10**. De la même manière, il est possible de n'apporter des changements qu'à une ligne précise, par exemple **:2** qui ne concerne que la seconde ligne.

► **s** est la commande de substitution/remplacement. Sa syntaxe est relativement simple : **s/ancien/nouveau**

► enfin, **g** et **c**, sont optionnelles. **g** permet d'appliquer le remplacement de toutes les occurrences dans une ligne. En omettant cette option, seule la première occurrence de **bo[iî]te*** serait remplacée. **c** permet de confirmer manuellement les remplacements. Dans le bas de l'écran de Vim, une question sera posée avec tout remplacement permettant de confirmer ou non, d'annuler l'opération, mais également de déclencher un remplacement sans confirmation.

Une autre manière de procéder est d'utiliser la commande **ex g**. Attention, ne confondez pas la commande **g** et l'option **g** citée plus haut. La ligne complète utilisant la commande **g** serait alors :

```
:g/bo[iî]te*/s//récipient/gc
```

Le résultat est le même, mais le sens de la commande est différent. Ici, la commande **g** nous permet de sélectionner les lignes contenant **bo[iî]te***. C'est alors sur ces lignes que nous utilisons la commande de remplacement **s** en substituant le motif recherché par **récipient**. **g** et **c** ont, bien sûr, la même utilité. Notez que vous n'avez pas besoin de répéter le motif de recherche pour la commande **s** si celui-ci est le même que pour **g**.



Figure 4 : Des efforts de traduction existent pour la documentation et l'aide en ligne de Vim. Et pour aider, rendez-vous sur <http://vim.dindinx.net>

Pourquoi donc existe-t-il deux manières de procéder ? Nous arrivons au même résultat avec les deux méthodes, car notre remplacement est simpliste. Imaginez-vous dans le cas où vous posséderiez une liste très longue de produits alimentaires. Dans cette liste, figurent des tomates vertes et rouges et des haricots rouges. Malheureusement pour vous, les haricots rouges ne sont plus disponibles et remplacés par des haricots verts (je sais, c'est léger comme exemple, mais au moins c'est printanier, ce qui n'est pas désagréable en plein hiver). Vous ne pouvez donc pas remplacer toutes les occurrences de « rouge » en « vert ». Vous ne pouvez pas non plus rechercher « Haricots verts », car ces deux mots ne sont pas toujours côte à côte (« Haricots secs verts » par exemple). Dans ce cas, vous ne pouvez pas utiliser la commande `s`. En revanche, `g` vous sortira une belle épine du pied :

```
:g/haricot*/s/rouge/vert/g
```

Et voilà ! Nous venons de rechercher les lignes où l'on fait mention de haricots et ce n'est que dans ces lignes que nous remplaçons « rouge » par « vert ». Essayez d'en faire autant avec un traitement de texte, même le plus réputé d'entre eux.

Dernier exemple

Essayons de faire plus utile cette fois. Il est courant d'utiliser certains acronymes comme s'il s'agissait de mots courants. Mais, en bon français, les acronymes s'écrivent en majuscule. Ainsi, parler de « chaîne scsi » n'est pas tout à fait français. Il convient d'écrire « chaîne SCSI ». Encore une fois, Vim possède une solution. Commençons par la commande adéquate :

```
:%s/\([cC][sS][iI]\)/\U\1/g
```

Impressionnant, n'est-ce pas ?

- `%s` que vous connaissez déjà est la commande de remplacement appliquée jusqu'à la fin du fichier.
- La partie entre crochets est également acquise, nous cherchons « s » ou « S » suivi de « c » ou « C », etc.
- Autour de ce motif, nous avons placé des parenthèses, car nous comptons nous servir du résultat de la recherche dans la suite de la commande. Comme ces deux parenthèses ne font pas partie du motif de recherche, nous avons dû les faire précéder d'un `\`. C'est ce qu'on appelle « échapper un caractère ». Vous pouvez utiliser jusqu'à 9 motifs entre parenthèses. L'avantage des parenthèses est que le résultat de la recherche est conservé pour que nous puissions nous en resservir dans la suite de la commande.
- Nous remplaçons ensuite le résultat de la recherche par `\U\1`. Il s'agit davantage d'une transformation que d'un remplacement, puisque nous réutilisons le résultat de la recherche. `\U` (comme *uppercase*) permet de passer ce qui suit en majuscule. `\1` représente le résultat de la recherche.

- Nous appliquons cela à toutes les occurrences d'une ligne, `/g`.

Je dois avouer que cette commande est un peu complexe, mais fort utile. Résumons-la en expliquant ce qui va se passer. Vim va rechercher le motif. Il trouvera par exemple « ScSi ». Nous lui demandons alors d'utiliser ce qu'il trouve et de passer les caractères du résultat en majuscule. Ici, trois des caractères sont déjà en majuscule, mais le `c` est en minuscule. `\U` le transformera en « C », puis Vim continuera la recherche...

Partant des explications données dans cet article, amusez-vous avec Vim. Apprenez à naviguer dans le texte, à maîtriser les changements de mode, à remplacer et rechercher du texte... Nous avons vu ici que Vim nécessite un apprentissage, mais le jeu en vaut la chandelle. Les quelques exemples donnés dans cet article suffisent (je l'espère) à convaincre de la puissance d'un bon éditeur. Nous n'avons cependant vu qu'une toute petite partie de ce que Vim peut offrir. Ses fonctionnalités sont incroyablement nombreuses et originales : division de l'écran, pliage du texte (*folding*), fonctionnement distant, écriture de scripts... Plein de choses à découvrir si vous vous en donnez la peine... ■

Denis Bodor,

db@ed-diamond.com

lefinnois@lefinnois.net

