Cours *vi* et *vim*

Lorens Kockum *vi@lorens.org* 2013-02-02

J'ai rencontré beaucoup de personnes qui utilisent l'éditeur *vi* au quotidien, par obligation, parfois depuis des années, mais sans avoir eu d'autre formation que l'apprentissage sur le tas de quelques commandes indiquées par le voisin. Ce document-ci est écrit pour eux.

Contents

1 Licence, versions HTML et PDF 2

2 Introduction (c'est ici qu'il faut commencer) 2

3 Généralités 2

4 Interprétation des touches 3

5 Les commandes de déplacement 4

6 Les commandes pour passer en modes frappe 4

7 Les commandes ex 5

8 Options 8

9 Commandes de recherche 9

10 Les commandes de copier-couper-coller 10

11 D'autres commandes 11

12 Tampons 12

13 Macros 13

14 Un tout petit peu de système autour 14

15 Petites astuces 14

16 Pour aller plus loin 14

A Commandes d'affichage sans bouger le curseur dans le fichier 15

B Commandes de positionnement du curseur 16

1 Licence, versions HTML et PDF

- *HTML une page* http://vi.lorens.org/vi_15_ans_apres.htm
- *HTML multi-pages* http://vi.lorens.org/vi_15_ans_apres.html
- *PDF* http://vi.lorens.org/vi_15_ans_apres.pdf
- *Texte UTF8 avec effets* http://vi.lorens.org/vi_15_ans_apres.rich.utf8.txt
- *Texte UTF8 sans effets* http://vi.lorens.org/vi_15_ans_apres.plain.utf8.txt
- *Texte ISO8859-1 avec effets* http://vi.lorens.org/vi_15_ans_apres.rich.iso.txt
- *Texte ISO8859-1 sans effets* http://vi.lorens.org/vi_15_ans_apres.plain.iso.txt

Cette œuvre a pour objet principal la déscription d'une autre œuvre appelée *vim*, dont la licence (commande *vim* **:help license**) incite à faire des dons à une cause pour laquelle l'auteur de *vim* est engagé. Si vous utilisez *vim*, visitez *http://www.vim.org/iccf/* .

2 Introduction (c'est ici qu'il faut commencer)

J'ai cherché, mais je n'ai pas trouvé (et surtout pas en français) une introduction à *vi* qui soit ni basique, ni long et interactif, ni un simple listing de commandes. Les commandes se retiennent mieux quand on comprend la logique qui est derrière ! Quand on les comprend, il y a d'ailleurs beaucoup moins à retenir...

J'ai donc voulu permettre l'exploitation efficace de l'éditeur UNIX omniprésent et aux capacités souvent peu soupçonnées.

En écrivant à destination d'un public déjà utilisateur, professionnels qui chercheront un maximum de retour sur leur investissement de temps, j'ai voulu privilégier l'explication mnémotechnique et la présentation d'exemples réels.

J'ai donc écrit ce document pour être lu d'une traite, rapidement et séquentiellement, sans sauter de sections, et pour qu'il serve ensuite d'aide-mémoire (si besoin !). Je ne couvre pas tout, mais j'ai essayé d'inclure tout ce qui me sert plus ou moins régulièrement. Je laisse à la fin un aperçu pour ceux qui voudront approfondir.

3 Généralités

vi date de la fin des années 1970. Il fait partie de la spécification UNIX (Single UNIX Specification), et doit donc être présent sur tout système se voulant UNIX ou compatible UNIX.

vim (vi improved) est un remplaçant étendu du *vi* standardisé. Il en existe d'autres : *elvis* le (presque) premier, *Nvi* le défaut sous BSD, *vile* comme emacs...

Les Solaris, AIX, et autres HP-UX fournissent des versions identiques au standard, ou en tout cas plus proches que ne l'est *vim*. Dans la suite, j'appelle *vi* le standard, je travaille cependant avec *vim*. J'essaie de noter quand un comportement n'est pas disponible dans le *vi* standardisé.

La commande ex **:version** permet de savoir dans quel éditeur on se situe.

vim a trois aspects selon le nom utilisé pour l'invoquer : *vi* pour un comportement au maximum compatible *vi*, *vim* pour le mode texte, et *gvim* pour un mode graphique. Les deux derniers ne sont pas forcément installés. Le comportement compatible garde tout de même certaines extensions (notamment touches flèche en modes frappe, mais pas en ligne de commande ex). Même si le vim complet est bien installé, on peut cependant se retrouver en comportement compatibilité sans le vouloir, par exemple avec *visudo*, *crontab -e*, ou d'autres commandes qui lancent eux-mêmes un éditeur. Dans ce cas-là il suffit généralement de définir d'abord la variable d'environnement EDITOR (sous *bash* : *export EDITOR=vim*).

4 Interprétation des touches

Chose souvent déroutante pour les néophytes, l'appui sur une touche ne produit pas forcément un effet visible. En effet, *vi* est ce qu'on appelle un éditeur "modal", avec un héritage d'avant les touches curseur et même d'avant les écrans...

Il y a de nombreux modes (**:help vim-modes** en compte six principaux et six secondaires), et leurs noms peuvent prêter à confusion. Pour simplifier un peu, je compte trois façons principaux d'interpréter les touches :

- commande (mode normal, aussi appelé mode commande) : les touches ne s'affichent pas. Chaque touche effectue une commande ou contribue à effectuer des commandes qui font au plus quelques touches.
- frappe (mode insertion, mode remplacement...) : les caractères s'affichent à l'écran à l'endroit du curseur pour composer le texte.
- ex (mode command-line, mode ex) : le curseur se met sur la dernière ligne, les touches s'affichent sur cette ligne uniquement, et une fois la commande composée on valide par Entrée.

Les habitués se baseront souvent sur le mode normal, y revenant à chaque fois qu'ils s'arrêtent de taper. Cela a des avantages notamment pour la répétition des commandes et pour revenir sur ses modifications.

Quand vous lancez l'éditeur, vous êtes en mode normal, on commencera par là. (Bon, pour lancer *vi* on met en paramètre un ou plusieurs noms de fichier, on n'ira pas plus loin que ça !)

Les commandes sont majoritairement représentées par une touche. On peut les classer en quelques catégories :

- les déplacements
- les commandes simples
- les commandes suivis d'un déplacement

La plupart peuvent être utilement préfixées d'un nombre qui indique un nombre de répétitions à effectuer.

5 Les commandes de déplacement

Des commandes permettent de déplacer le curseur. Eh oui, dans les années 70, les touches curseur n'étaient pas courantes ! Comme vous le savez, les flèches devraient marcher la plupart du temps, en tout cas sous Linux, mais le détail aura son importance plus tard.

Il y a de très nombreuses commandes de déplacement, je n'en citerai que quatre parce que je les utilise comme exemples. Une liste complète est en B (annexe).

$

aller en fin de ligne (le $ signifie fin en regexp)

w

aller au mot suivant (word)

j

descendre d'une ligne

G

fin du fichier

Vous pouvez préfixer une commande par un nombre, qui indique un nombre de fois à répéter la commande (que ce soit un déplacement ou une autre commande). Par exemple, **5w** signifie d'avancer de cinq mots vers la droite, et **50j** signifie de descendre cinquante lignes. Préfixer **G** par un nombre conduira au numéro de ligne correspondant.

Préfixer n'est pas parfaitement équivalent à la répétition, notamment quand il s'agit de couper des caractères, mais on verra ça plus tard avec le copier-coller.

Une commande de recherche (section 9 (Commandes de recherche)) peut sans aucun problème être utilisée comme commande de déplacement, et également être préfixée d'un nombre pour aller à la n-ième occurrence.

6 Les commandes pour passer en modes frappe

Depuis le mode normal, appuyer sur une des touches suivantes :

i

insérer du texte à l'endroit du curseur

I

insérer du texte avant le premier caractère imprimable de la ligne

a

ajouter du texte après le curseur

A

ajoute du texte en fin de ligne

o

ouvre une nouvelle ligne au-dessous de la ligne courante

O

ouvre une nouvelle ligne au-dessus de la ligne courante

s

remplace le caractère sous le curseur par le nouveau texte. Pas très différent de la suite de commandes **xi**

S

remplace la ligne entière — équivalent de **cc** (par exemple).

R

remplace au fur et à mesure de la frappe, jusqu'au bout de la ligne. Il faut noter que ce remplacement est fait sans avoir coupé le texte supprimé vers le tampon comme l'aurait fait la commande **C**.

Après un de ces caractères, vous êtes en mode insertion (ou remplacement). Vous pouvez taper le texte que vous voulez. Pour revenir en mode normal, utilisez la touche Échap. Pour insérer dans le texte un caractère de contrôle tel que Échap, précédez-le de Ctrl-V.

Pour passer en mode insertion il y a aussi la commande **c** (change), très utilisée, qui coupe une partie du texte et entre en mode frappe pour le remplacer. Elle est décrite avec les autres commandes de 10 (copier-couper-coller).

Les lignes continuent jusqu'à ce que vous tapez Entrée, c'est-à-dire qu'il n'y a pas de retour à la ligne automatique, à moins que vous ne l'ayez programmé !

Sous *vim*, une mention en bas de l'écran vous indique quand vous êtes en mode insertion ou remplacement ("–INSERT–" ou "–INSERTION–" par exemple).

Sous *vim* également, vous pouvez (sur la plupart de terminaux) utiliser les flèches pendant que vous êtes en mode insertion ou remplacement. Ce n'est pas le cas pour *vi*.

Notez que même sous *vim*, utiliser une touche flèche lors de la frappe met fin à la commande en cours et en commence une nouvelle. C'est transparent, sauf si vous voulez rejouer la commande avec la commande **.** (dans quel cas vous ne rejouerez que ce qui s'est passé depuis le dernier mouvement de curseur), ou si vous avez tapé un nombre devant votre commande (dans quel cas rien ne sera répété).

7 Les commandes ex

ex est un autre aspect de l'éditeur *vi*, on peut même dire que *vi* n'est "que" le mode "visuel" de l'éditeur qui s'appelle en fait *ex*. Les commandes ex peuvent être beaucoup plus élaborées que les commandes du mode normal.

On y accède en tapant un **:** (un deux-points) depuis le mode normal. Ce mode s'appelle "command line" dans la documentation, parfois en anglais "colon mode" du fait du deux-points initial. Je l'appelerai (lourdement) "ligne de commande ex" pour le distinger du mode commande qui est un autre nom pour le mode normal... Ce mode ligne de commande ex permet de taper une commande ex. Après avoir exécuté la commande, on revient en mode normal.

Il y a aussi le mode "ex", contraste du mode "visual". Dans ce mode, après avoir exécuté sa commande ex, on reste en attente d'une nouvelle commande ex. Ce mode devait être utile quand dans les années 1970 on n'avait qu'une imprimante matricielle pour servir d'écran, mais j'ai du mal à lui voir une utilité hors de ce contexte.

Les caractères s'affichent en bas de l'écran. Un appui sur la touche Entrée exécute la commande ex et vous fait habituellement revenir en mode commande. Un appui sur Échap ou Ctrl-C abandonne.

Dans ces modes ex l'interprétation des touches est plus proche du shell *sh*. Le déplacement du curseur se fait avec les flèches, les touches curseur Home et End sont habituellement mappés en début et fin de ligne, et Ctrl-W supprime le mot précédent, par exemple.

Certaines commandes ex (notamment **:version**) affichent plusieurs lignes et ne reviennent pas tout de suite en mode normal. Pour cela il faut appuyer une nouvelle fois sur Entrée.

Pour distinguer les commandes mode normal des commandes ex, je les note en considèrant que le **:** fait partie de la commande. Pour exécuter la commande **:version**, on part donc comme toujours du mode normal, on tape la touche **:** qui au lieu de s'afficher là où est le curseur dans le texte va s'afficher en bas à gauche de l'écran. Le curseur se positionne juste après, on tape **version** puis la touche Entrée. Le texte informatif sur la version s'affiche, suivi d'une invitation à appuyer une nouvelle fois sur Entrée. Après avoir appuyé, le texte informatif disparaît et on revient en mode normal, avec le curseur au point d'origine puisque la commande exécutée n'avait pas pour but de déplacer le curseur. Les commandes ex qui modifient le contenu du fichier laisseront le curseur au dernier endroit où le fichier a été modifié.

Les principales commandes sont :

:q

quitter, lorsque le fichier n'a pas été modifié

:q!

quitter en abandonnant les modifications

:w

sauvegarder les modifications (write) Ajouter un **!** pour forcer, par exemple quand le fichier sur disque a été modifié hors *vim*, ou quand il n'est en principe pas écrivable.

:wq

quitter en sauvegardant les modifications (write quit)

:x

idem que **:wq** (eXit)

:n

(Next) passe au fichier suivant quand on a donné plusieurs fichiers sur la ligne de commande au lancement et que l'actuel n'a pas été modifié. **:wn** pour sauver et passer à la suite, **:n!** pour passer au suivant sans sauver, **:p :wp :p!** pour le fichier précédent. Rajouter un **!** pour forcer l'abandon d'un fichier modifié ou forcer l'écriture d'un fichier.

:w nouveaunomdefichier

sauvegarder le fichier sous un autre nom (mais sans changer le nom du fichier en cours). Rajouter un **!** (donc **:w! nouveaunomdefichier**) pour forcer.

:f

affiche des informations en bas de l'écran (le nom du fichier, le nombre de lignes, la position dans le fichier) en plus de ce que *vim* met habituellement.

:history

Affiche l'historique des commandes ex. Dans la plupart des versions de *vi* vous pouvez aussi parcourir l'historique des commandes ex avec les flèches verticales. Il y a d'autres façons de les obtenir aussi...

:g/RECHERCHE/#

exécute une commande ex (ici #) sur toutes les lignes correspondant à la recherche. # étant une commande ex qui affiche la ligne préfixée de son numéro, cette commande-ci revient donc à un **:%!grep -n 'RECHERCHE'** suivi d'un **u** pour annuler).

:!ls

lancer une commande Unix (ici *ls*). La sortie s'affiche et après l'appui sur une touche vous revenez au fichier.

:r!*ls* | *grep -i data*

(read) insérer après la ligne du curseur la sortie d'une commande Unix (ici, insérer la sortie de la commande *ls* filtrée par un *grep -i data*)

:r*autrefichier.txt*

(read) insérer après la ligne du curseur le contenu d'un fichier. Sous *vim* l'autocomplétion fonctionne avec la touche Tabulation.

:%!sort X

passer tout le fichier par une commande filtre Unix, ici **sort**. Les lignes passées sur l'entrée standard de la commande disparaissent et sont remplacées par la sortie standard de la commande.

:%s/RECHERCHER/REMPLACER/OPTION

passer tout le fichier par le filtre interne sous-ensemble du filtre UNIX *sed*. L'OPTION la plus courante est **g**, sans laquelle seul le premier remplacement de chaque ligne sera fait. Une autre option est **c** qui demande confirmation avant le remplacement. Comme avec *sed*, on peut remplacer le / par autre chose pour travailler sur des noms de fichier sans être obligé de mettre un backslash devant chaque slash, mais on a un peu moins de choix qu'avec *sed* (pas de pipe, pas de lettre). Un exemple avec une virgule : **:%s,/bin/,/sbin/,g**

:%d

couper tout le fichier (**d** comme en mode normal, comme dans *sed*, etc.).

:%y

copier tout le fichier dans le tampon par défaut (yank, comme en mode normal)

:%w fichier

écrire tout le fichier dans un nouveau fichier (comme **:w fichier**)

Ces trois dernières ne servent pas tels quels, mais le **%** signifiant tout le fichier peut être remplacé par un spécification de lignes. Dans cette spécification, un nombre représente un numéro de ligne, un point signifie la ligne où est le curseur, un apostrophe suivi d'une lettre ou de < ou > signifiant une marque (voir la commande **m**) et même **/REGEXP/** une ligne correspondant à l'expression régulière. On peut y ajouter un plus ou un moins suivi d'un nombre. On a ainsi

:s/RECHERCHER/REMPLACER/g

substitution de toutes les occurrences, sur la ligne courante uniquement

:/^#/,/^#/+10s/RECHERCHER/REMPLACER/

aller à la prochaine ligne commençant par un # et effectuer le remplacement sur cette ligne et les 10 suivantes.

:5s/RECHERCHER/REMPLACER/

substitution uniquement sur la ligne 5

:5,50s/RECHERCHER/REMPLACER/

substitution de la ligne 5 à 50 inclus

:.,$w nouveaufichier

écriture dans un nouveau fichier des lignes à partir de la ligne courante jusqu'à la fin

:.,.+20s/RECHERCHER/REMPLACER/

substitution entre la ligne courante et la ligne 20 lignes plus bas

Une autre façon est d'utiliser depuis le mode normal la commande **!** suivi d'un déplacement. **!20j** affichera **:.,.+20!** et attendra en mode ligne de commande ex qu'on entre la commande à exécuter.

L'intérêt de ces spécifications un peu compliquées est plutôt quand on voudra copier-coller, répéter, ou scripter des commandes. En interactif avec *vim* on préférera la plupart du temps utiliser la commande **v** pour fixer le début et la fin. Cette commande est décrite dans la section 10 (copier-couper-coller).

Une dernière combinaison de touches : **@:** signifie "exécuter le tampon dans lequel est stocké la dernière commande ex". Cela relance donc la dernière commande ex, avec quelques restrictions cependant.

8 Options

Les options se configurent soit dans un fichier de configuration, soit avec une commande ex. Il y a trois formes :

:set option

affecte vrai à une option booléenne

:set nooption

affecte faux à une option booléenne

:set option=valeur

affecte une valeur à une option non booléenne

Il y a beaucoup d'options. Beaucoup ont une forme abbréviée (ts pour tabstop par exemple). En *vim* il y a normalement l'autocomplétion avec la touche Tabulation ; si on appuie plusieurs fois on parcourt les différentes possibilités. Les options que j'utilise le plus souvent sont

:set nowrap

pour que les lignes "logiques" dont la longeur dépasse la largeur de l'écran ne soient pas affichées sur plusieurs lignes "écran".

:set tw=65

(textwidth) pour définir la colonne à laquelle *vi* insère un retour à la ligne automatique

:set ts=30

(tabstop) pour définir la largeur de colonnes séparées par des tabulations

:set lazyredraw

pour que l'écran soit redessiné moins souvent (utile si on exécute une longue macro)

:set hlsearch

pour mettre en surbrillance toutes les occurrences de la recherche en cours. La commande pour l'enlever est **:noh**, mais si la surbrillance me gêne parce que je viens de rechercher quelque chose de très courant, je fais simplement une recherche bidon.

:set expandtab

pour convertir automatiquement toutes les tabulations qu'on tape en un nombre approprié d'espaces. Utile pour faire du python par exemple.

:set ignorecase

ignorer la casse lors des recherches.

:set paste

dans un terminal, où *vim* ne sait pas si les caractères proviennent du clavier ou d'un collage, cette option permet de désactiver la mise en forme automatique (autoindentation, continuation de commentaires...).

9 Commandes de recherche

La recherche revient à déplacer le curseur à la prochaine occurrence de la chaîne de recherche. On peut sans aucun problème l'utiliser comme commande de déplacement, et également la préfixer d'un nombre pour aller à la n-ième occurrence.

Pour entrer en recherche, appuyer sur / ou ?

/ correspond à une recherche en avant, ? à une recherche en arrière. Notez en passant que ce syntaxe est le même que dans la commande UNIX less.

Le mode d'interprétation des touches est celle des commandes ex (plus précisément, la recherche est une commande ex comme les autres). Il faut entrer une chaîne de recherche (expression régulière standard) et terminer par la touche entrée (ou Ctrl-C ou Échap pour abandonner). On peut ne rien entrer ; cela exécutera de nouveau la dernière recherche dans le sens indiqué.

La touche **n** réexécute la dernière recherche. La touche **N** effectue la recherche dans le sens contraire.

La touche ***** recherche en avant le mot sous le curseur, # recherche en arrière le mot sous le curseur.

Exemple : dans un fichier délimité par des points-virgule, les trois touches **d/;** suivis de la touche Entrée supprimeront la colonne pour peu qu'on soit sur un point-virgule, et ensuite des appuis sur le point referont la suppression.

10 Les commandes de copier-couper-coller

Tout d'abord, avant de parler de *vi* ou *vim*, la souris permet de copier et de coller. Le tampon de la souris est celui du terminal et n'a rien à voir avec les commandes qui suivent.

La commande la plus connue est **x** qui coupe le caractère sous le curseur. On peut comme d'habitude préfixer par un nombre. **5x** coupera une chaîne de cinq caractères.

Les commandes **d** (delete - couper), **y** (yank - copier), et **c** (change - couper et remplacer en mode frappe) attendent en paramètre un déplacement. La commande agit sur les caractères entre le curseur et la destination du déplacement. Ainsi, la commande **dw** signifie de couper le mot à la droite du curseur, **c$** signifie couper et remplacer jusqu'à la fin de la ligne, et **dG** de couper jusqu'à la fin du fichier. Notez simplement qu'un déplacement vertical embarquera toute la ligne courante, et pas simplement à partir du curseur.

Les commandes **D C Y** correspondent à **d$ c$ y$** ; sur un clavier français le raccourci n'est pas flagrant.

On peut toujours préfixer par des nombres : **y5w** ou **5yw** signifient copier les cinq mots à partir du curseur.

Pour agir sur toute la ligne, ou sur plusieurs lignes, on dédouble la lettre, donc **cc** pour remplacer la ligne, **y5y** ou **5yy** pour copier 5 lignes, **d10d** ou **10dd** pour couper dix lignes.

Un peu plus avancé : en *vim*, on peut au lieu d'un déplacement indiquer un objet texte, sur deux caractères. Le premier caractère est **i** (indiquant d'inclure les délimiteurs de l'objet texte) ou **a** (pour ne pas les inclure). Le deuxième caractère indique le type d'objet sur lequel agir : **w** pour word/mot, **s** pour sentence/phrase, **p** pour paragraphe, **" ' `** pour des chaînes délimitées par ces caractères, **)] } >** pour des chaînes délimitées par les couples de caractères **()** [] {} <>. Ainsi **diw** coupe le mot sous le curseur, **2dap** coupe le paragraphe sous le curseur et le suivant, **da>** coupe le tag HTML ou XML contenant le curseur et **ci>** le coupe en laissant les <> et mettant le curseur en place pour remplacer le tag.

Toujours avec *vim* : pour rendre le couper/copier plus facile, *vim* apporte la commande **v**. Un appui sur **v** pose une marque de départ (pensez "enfoncer la touche de la souris"). Vous pouvez ensuite déplacer le curseur comme vous voulez ("bouger la souris"). N'importe quel déplacement est possible, même une recherche. Le texte entre le curseur et la marque de départ sera mis en surbrillance, et une commande de manipulation de texte agira sur ce texte-là. Ça peut être une commande comme **d**, **c**, ou **y**, mais aussi une commande ex, dans quel cas l'appui sur **:** affichera **:'<,'>** ce qui correspond à la spécification de lignes avec les marques de début et fin posées par **v** (voir prochaine section). On peut enchaîner par exemple par **!** ou **s** (voir les commandes ex). On peut aussi appuyer sur **!** directement, mais pas sur **s** car c'est interprété comme la commande **s** (supprimer et passer en mode frappe).

La commande **V** fait pareil que **v** mais sur des lignes entières.

Après avoir coupé ou copié, on peut coller. C'est fait avec les commandes **p** (put after) et **P** (put before). Selon que le tampon utilisé (voir section sur les tampons) contient des lignes entières ou pas, ces commandes colleront avant ou après la ligne du curseur ou bien avant ou après le curseur. Comme d'habitude, on peut

préfixer par un nombre. Pour faire suivre la ligne actuelle de cinq répétitions, on a donc le choix entre **dd6P** ou **yy5p**

Petits exemples rapides : **xp** pour inverser deux caractères, **ddp** pour inverser deux lignes.

11 D'autres commandes

Certaines commandes attendent un déplacement, et ce déplacement peut être remplacé par une sélection préalable avec **v**, mais beaucoup d'autres commandes acceptent une sélection préalable.

r

suivi d'un caractère remplace le caractère sous le curseur. Si précédé d'une sélection, tous les caractères de la sélection (sauf les retours à la ligne) seront remplacés. Par contre ça ne sert à rien de le précéder par un nombre de répétitions !

J

va joindre la ligne suivante à la fin de la ligne courante, avec un espace séparateur. On peut précéder d'une sélection ou d'un nombre de répétitions.

.

Le simple point répète la dernière commande exécutée, hors déplacements et hors commandes ex. Par exemple, si vous voulez rajouter un point-virgule à la fin de quelques lignes, sur la première ligne faites **A;** suivi de Échap, puis positionnez-vous sur chaque autre ligne, sans vous occuper de l'endroit où vous êtes sur la ligne, et appuyez sur point. Comme noté ci-dessus, un mouvement de curseur à l'intérieur de la mode frappe va interrompre "l'enregistrement" de la commande. On peut précéder d'un nombre de répétitions. Pareil pour indenter quelques lignes : mettre le curseur sur la première ligne à indenter, appuyer sur **I** suivi des caractères à mettre, puis sur Échap. On a indenté la première ligne. On met le curseur sur la prochaine ligne, peu importe où sur la ligne, on appuie sur **.** et ainsi de suite. Il y a bien sûr d'autres moyens, dont une commande exprès :

>

indenter (suivi de déplacement ou précédé d'une sélection avec v, par exemple). < enlève l'indentation. La taille de l'indentation est contrôlée par l'option **sw** (shiftwidth).

n N * #

voir section 9 (Commandes de recherche)

u

(undo, annuler). Annule la dernière commande. Sous *vi*, il n'y a normalement pas d'undo multiple, et un deuxième appui sur **u** fait un redo. J'ai cependant connu des versions (*nvi* je crois) où un **u** suivi de points remontait l'historique, et un deuxième **u** suivi de points repartait en avant. Sous *vim*, de multiples appuis sur **u** remontent l'historique, Ctrl-R repart en avant. Quand je me rends compte que j'ai supprimé quelque chose que je voulais remettre ailleurs, je remonte facilement très loin pour retrouver ce que je veux, je copie avec la souris, et j'enfonce Ctrl-R pour revenir où j'étais avec le texte voulu prêt à être collé avec un clic de souris.

ZZ

équivalent de **:wq**

ga

affiche le code hexa du caractère sous le curseur. Si vous faites beaucoup de travail sur les fichiers binaires, renseignez-vous sur la commande UNIX *xxd* et l'option -b de *vim*.

~

(le tilde) va inverser la casse de la lettre sous le curseur (ou de la sélection préalable). C'est un moyen facile de faire une majuscule accentuée...

gu

suivi de déplacement (ou **guu** pour toute la ligne, ou précédé d'une sélection) convertit en minuscules. On retrouve là le même principe de dédoublement que pour les commandes **cc dd yy**

gU

suivi de déplacement (ou **gUU** pour toute la ligne, ou précédé d'une sélection) convertit en majuscules.

m

suivi d'une lettre pose une marque à l'endroit du curseur. **'** (l'apostrophe) suivi de cette lettre indiquera l'endroit de la marque ; en mode normal cela y déplace le curseur, en mode ligne de commande ex cela permet de définir une spécification de lignes telle que dans la commande **:'a,$s/X/Y/g** (remplacement de X par Y sur toutes les lignes entre la marque "a" et la fin du fichier).

12 Tampons

Le fichier sur lequel on travaille est dans un tampon mémoire que j'appelle le fichier, mais il y a d'autres tampons. Il y a 26 tampons nommés par des lettres. Au moins en *vim*, il y a aussi 10 tampons nommés par des chiffres, et même quelques uns nommés par un signe de ponctuation. Les tampons alphabétiques servent quand on y fait appel explicitement (je les utilise surtout pour enregistrer des macros, voir la section 13 (Macros)). Les tampons numériques sont les tampons par défaut du copier-coller de lignes entières : si on coupe deux lignes elles arrivent en tampon 1. Si ensuite on coupe cinq lignes, elles arrivent en tampon 1 et les deux lignes qui y étaient sont déplacées en tampon 2, etc. Si on ne coupe pas des lignes entières, c'est perdu (il faudra utiliser undo). La destination par défaut d'une copie (yank) est le tampon 0, et pour autant que je puisse déterminer les commandes coller **p** et **P** s'alimentent par défaut dans le dernier utilisé entre le tampon 0 et 1.

On désigne les tampons nommés par des lettres ou des chiffres en appuyant sur **"** suivi de la touche correspondante. Pour la copie ou la coupe vers les tampons alphabétiques, utiliser la majuscule fera un ajout au contenu actuel du tampon, sinon le contenu d'avant est écrasé.

Ainsi, **"dP** va insérer le contenu du tampon **d** avant le curseur, et **"px** va mettre le caractère sous le curseur dans la tampon **p**. L'intérêt de mettre un seul caractère dans le tampon peut paraître minime, mais ça peut avoir son intérêt si le caractère est un caractère bizarre voire non imprimable, provenant par exemple d'un mauvais encodage d'accent. En mode ex, appuyer sur Ctrl-R suivi de la touche du tampon va insérer le contenu du tampon. On pourra ainsi faire du rechercher ou du rechercher-remplacer sur un caractère qu'on n'aurait pas su taper au clavier.

On peut ouvrir plusieurs fichiers à la fois, et dans ce cas chaque fichier a son tampon. On peut visualiser plusieurs fichiers à la fois, voire avoir plusieurs fenêtres de visualisation sur un même tampon. Je n'en parlerai pas plus car je n'utilise guère ; je préfère ouvrir plusieurs fenêtres.

13 Macros

Les macros n'existent pas en *vi* standard.

q suivi d'une lettre de tampon enregistrera la suite dans le tampon, aussi bien des commandes normales que des commandes ex. On termine par un deuxième appui sur **q**. On réexécute avec la commande **@** suivie de la lettre du tampon. Un recherche qui n'aboutit pas arrête l'exécution. Comme d'autres commandes, la commande **@** peut être préfixée d'un nombre... 5, 10, 100000. Habituellement je teste ma macro une fois ou deux, et avec d'exécuter un grand nombre de fois je mets l'option lazyredraw (**:set lazyredraw**), car la macro s'exécute plus vite. Cependant, même ainsi c'est plus lent qu'un *sed* ou un *tr* simple sans passer par *vi*, et à partir de quelques centaines de milliers de lignes il vaut habituellement mieux de sortir de *vi* et de prendre le temps de faire un *sed* ou un *tr* voire un *perl*.

Pour peu que *vim* ait le droit d'écrire dans le fichier *~/.viminfo*, les macros sont persistants.

Les macros *vim* sont nettement plus faciles à faire que les scripts sed dès que l'on agisse sur plusieurs lignes. Un exemple : pour passer d'un format

```
Entete1
   ligne1-1
   ligne1-2
   ligne1-3
Entete2
   ligne2-1
   ligne2-2
   ligne2-3
```

à un format

```
Entete1:ligne1-1:ligne1-2:ligne1-3
Entete2:ligne2-1:ligne2-2:ligne2-3
```

je mets l'option lazyredraw, j'enregistre une macro (p par exemple) consistant à mettre le curseur sur une ligne commençant par espace, monter d'une ligne, joindre avec la ligne suivante (celle qui commençait avec espace, donc), puis remplacer l'espace entre les lignes par un : (le dernier q termine la macro). J'exécute la macro une fois ou deux pour vérifier que ça marche bien, et ensuite je la lance par exemple 10000 fois. Ça donne :

```
:set lazyredraw
qp/^ /
kJr:q@p@p10000@p
```

Le nombre de répétitions dépend bien sûr du nombre de lignes dans le fichier. Trop c'est pas grave, une macro s'arrête quand une recherche n'aboutit pas. Pas assez ce n'est pas grave non plus, car pour un très gros fichier, ça permet de surveiller l'exécution : si l'exécution 10000 fois prend 1 minute mais n'a atteint que 10% du fichier, on peut envisager de faire 100000 fois et d'aller prendre un café, mais si ça n'a pas dépassé 0% du fichier, c'est le moment de faire un script sed ou perl !

Pour interrompre l'exécution, taper Ctrl-C.

14 Un tout petit peu de système autour

vim crée des fichiers temporaires (swap files). Ce fichier contient l'état de l'édition dans un format spécifique à *vim*. Si le fichier n'a pas de nom (si on a juste lancé la commande *vim*), alors ils sont dans le répertoire $HOME et le premier fichier s'appelle *.swp*. Si le nom est déjà pris, les suivants s'appellent *.swo .swn* etc. Si le fichier a un nom, alors ils sont dans le répertoire du fichier édité et le nom du fichier temporaire est ce nom précédé d'un point et suivi de *.swp*. Si on essaye d'éditer un fichier alors que le fichier temporaire associé existe déjà, vim émettra un message d'erreur et proposera un menu avec des actions. Si la raison est qu'une session précédente s'est terminée anormalement, on peut récupérer l'état de l'édition (option Recover), dans quel cas il faudra ensuite supprimer le fichier temporaire manuellement. Ce n'est pas parfait ; j'aurais bien aimé une gestion plus facile du cas où le fichier n'a pas été modifié.

15 Petites astuces

J'aime bien faire des mini-macros dans une fenêtre à côté, que je copie avec la souris. Ainsi un appui sur le bouton "coller" de ma souris exécute ma macro, et je peux l'éditer facilement, ou prendre un morceau seulement. Pour cela, les touches **hjkl** de déplacement du curseur sont très utiles. Je ne connais pas de moyen de simuler la touche Échap en passant par le copier-coller de la souris, donc soit je ne fais qu'une seule commande que je symbolise par le point, soit je copie depuis des tampons, soit je fais une vraie macro.

Une macro étant enregistrée dans un tampon, rien n'empêche de la copier dans le tampon principal pour la visualiser, ou de copier du texte dans un tampon pour l'exécuter.

On peut rechercher deux mots séparés par un saut de ligne en utilisant \n. Les objets texte, les petites macros, et les spécifications de lignes ex permettent de faire sans grande difficulté des remplacements impliquant plusieurs lignes.

16 Pour aller plus loin

La première chose à faire pour aller plus loin est d'apprendre *sed* et les expressions régulières, qui serviront pour effectuer recherches/remplacements avancés, que ce soit dans *vi* ou pas.

Ceux qui cherchent à utiliser *vi* efficacement sur des centaines d'ordinateurs différents ne voudront pas s'encombrer de fichiers de configuration ou de macros enregistrés, et s'arrêteront là.

Pour les autres, il y a je ne sais combien d'options. On peut les définir dans le fichier de configuration de *vim* (système ou utilisateur). On peut définir des options selon l'extension du fichier en cours.

On peut définir sa propre coloration syntaxique (par défaut sous linux beaucoup sont déjà prédéfinies ; elles se déclenchent selon l'extension du fichier).

On peut redéfinir les touches et même des suites de touches, en mode normal, insertion, ou ex, et en assigner à des macros. Juste un exemple : sur tous les ordinateurs où j'utilise habituellement *vi*, j'ai installé l'utilitaire "par", et redéfini la touche F ainsi :

```
:map F {!}par -w64 -gtq
```

Décryptage : à chaque appui sur F en mode normal exécuter ce qui suit : { monte le curseur au début du

paragraphe, **!}** exécute la commande qui suit sur le texte du curseur jusqu'à la fin du paragraphe. *par* est un filtre qui reformate le texte, ici pour faire de jolis paragraphes d'au maximum 64 caractères.

Comme j'ai dit plus haut, on peut éditer plusieurs fichiers à la fois, mais aussi ouvrir plusieurs fenêtres sur le même fichier.

On peut "plier" le texte (folding) pour par exemple n'afficher que la première ligne de chaque fonction d'un langage de programmation, et ainsi naviguer plus facilement dans son code.

On peut s'intégrer avec un compilateur pour l'édition de code, par exemple en affichant code, compilateur, et exécution dans des fenêtres séparées, en allant directement à la ligne ayant généré une erreur de compilation, etc. Bon, en 2013 Eclipse est plus facile !

On peut installer beaucoup de scripts plus ou moins gros provenant d'autres utilisateurs.

Pour tout cela, il vous faudra une documentation plus complète. L'aide interne (**:help**) pour commencer (on en sort par **:q**). Ensuite, O'Reilly a publié un livre (bien sûr). Vos recherches web vous mèneront sans doute vers `http://vimdoc.sourceforge.net` , `http://vim.wikia.com` , et plus généralement vers `http://www.vim.org` qui renvoie sur ces ressources-là et sur beaucoup d'autres.

Pour l'histoire de *vi* (et *ex*), par exemple pour voir le clavier utilisé par Bill Joy qui lui a conduit au choix des touches hjkl et Échap, ou pour comprendre les multitudes de versions, consultez d'abord Wikipedia.

J'espère que cette introduction un peu poussée vous aidera à mieux exploiter l'outil !

Je remercie ceux qui m'ont relu et m'ont apporté leurs commentaires et corrections, notamment Jérôme Abela et Ludo. Toutes les erreurs qui restent sont de moi... n'hésitez pas à me les signaler pour que je les corrige !

Pour ceux qui se le demandent, oui tout cet article a été écrit exclusivement avec *vim*, et mis en forme avec LinuxDoc ; le résultat n'est certes guère joli comme ça mais cela permet d'avoir facilement des versions HTML, texte, PDF, et ça n'a pas changé depuis que je l'avais choisi en 1997 pour un support de cours *vi*.

A Commandes d'affichage sans bouger le curseur dans le fichier

Ctrl-L

Re-affiche l'écran

z suivi de Entrée

Re-affiche l'écran avec la ligne courante en haut

z. ou zz

Re-affiche l'écran avec la ligne courante au centre. C'est très agréable pour voir le contexte, par exemple après avoir fait une recherche et être arrivé sur la dernière ligne, ou pendant qu'on exécute des suites de commandes par macro ou par le coller de la souris et qu'à chaque exécution on se retrouve une ou quelques lignes plus bas.

z-

Re-affiche l'écran avec la ligne courante en bas

Ctrl-E

Re-affiche l'écran avec le texte une ligne plus haut. Le curseur ne bouge pas dans le texte (donc il monte d'une ligne sur l'écran) sauf s'il est déjà à la première ligne de l'écran.

Ctrl-Y

Re-affiche l'écran avec le texte une ligne plus bas. Comme pour Ctrl-E, le curseur ne bouge pas dans le texte (donc il descend d'une ligne sur l'écran) sauf s'il est déjà à la dernière ligne de l'écran.

B Commandes de positionnement du curseur

h

curseur gauche (cf. Ctrl-H). Le curseur n'ira pas au-delà du début de la ligne logique actuelle.

j

curseur bas, donc aller à la ligne logique suivante. Si la partie restante de la ligne logique actuelle dépasse sur les lignes écran suivantes, le curseur sautera des lignes écran.

k

curseur haut, donc aller à la ligne logique précédente (même remarque que pour le curseur bas).

l

curseur droit. Le curseur n'ira pas au-delà de la fin de la ligne logique actuelle.

^

premier caractère imprimable de la ligne (cf. regex)

0

début de la ligne

:0

début du fichier

$

fin de la ligne (cf. regex)

:$

fin du fichier (cf. regex)

gg

début du ficher (*vim* uniquement, sous *vi* **:0** fera l'affaire)

G

fin du fichier

un nombre suivi de G ou gg

aller à la ligne de ce numéro, en la positionnant au centre de l'écran

un nombre suivi de z et Entrée

aller à la ligne de ce numéro, en la positionnant en haut de l'écran

un nombre suivi de %

Aller à ce pourcentage du fichier. Très utile pour faire de la dichotomie dans un fichier volumineux (par exemple, si on a un fichier de log gigantesque pour un mois entier et qu'on veut trouver le 20 du mois, ce sera beaucoup plus rapide de commencer par disons **60%** plutôt que de faire une recherche du timestamp depuis le début du fichier).

Ctrl-F

(Forward) Avancer d'un écran. Préfixable par un nombre. Habituellement accessible par la touche PageDown.

Ctrl-D

(Down) Avancer d'un demi-écran. Préfixable par un nombre.

Ctrl-B

(Backward) Recule d'un écran. Préfixable par un nombre. Habituellement accessible par la touche PageUp.

Ctrl-U

(Up) Reculer d'un demi-écran. Préfixable par un nombre.

H

(High) Aller à la première ligne de l'écran. Préfixer par un nombre n correspond à **H**n**jk**, donc aller à la n-ième ligne de l'écran.

L

(Low) Aller à la dernière ligne de l'écran. Préfixer par un nombre n correspond à **L**n**kj**, donc aller à la n-ième ligne de l'écran en comptant de la fin.

M

(Middle) Aller au milieu de l'écran

w

(word) Mot suivant. En majuscule, délimité par blanc.

b

(back) Mot précédent. En majuscule, délimité par blanc.

e

(end) Fin du mot. En majuscule, délimité par blanc.

( )

Parenthèse précédente **(** et suivante **)**

{ }

Paragraphe (c'est-à-dire ligne blanche) précédente { et suivante }