vi

(vi a 50 ans)

vi.lorens.org

Lorens

Développement logiciel de sécurité réseau
Sécurité informatique grand FAI
Directeur technique petit registrar et FAI B2B
Ingénieur système banque
Sécurité informatique groupe petites annonces

Et ensuite ?

Lorens et vi

32+ ans de vi

Un cours vi, quelle idée…

En 1997 une première mouture de cours, pour étudiants

En 2013 le “15 ans après” sur vi.lorens.org, pour les collègues

En 2026… pour FinistDevs !

Pourquoi vi ?

Écrit par Bill Joy en 1976
Avant on utilisait ed… un éditeur ligne par ligne !

Avec le Single UNIX Specification, vi est partout !

Emacs ? Une philosophie différente, plus complète… mais incompatible, moi je ne peux pas concilier les deux !

Il y a beaucoup d'évolutions de vi, et dans cette présentation on va dériver vers vim. Il y en a d'autres, souvent à partir de vim : neovim, helix…

Un éditeur… modal ?

Je tape mais rien ne s'affiche ?

:help vim-modes compte 6 modes principaux et 6 secondaires, mais on va faire plus simple:

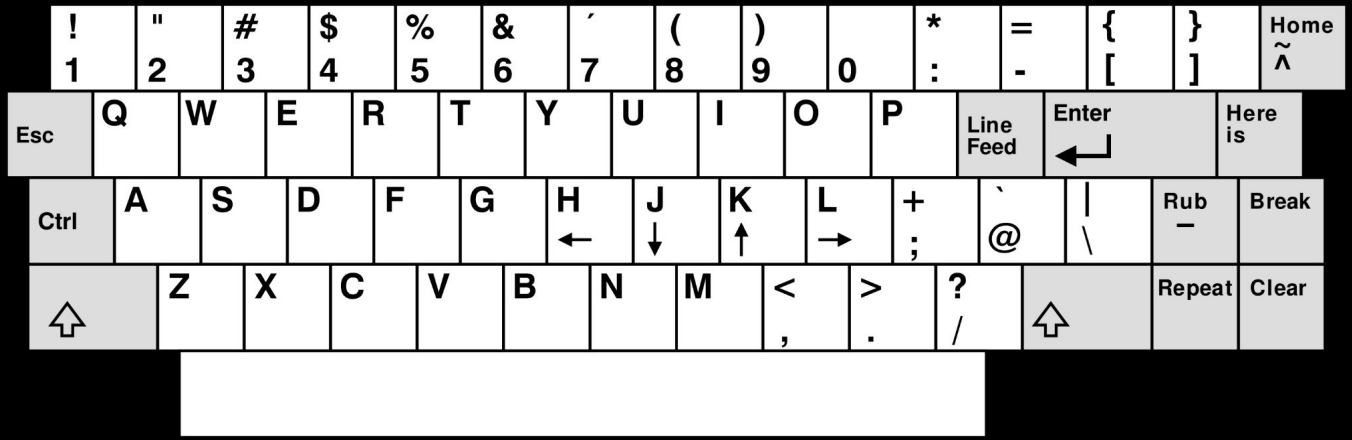
- Normal (ou commande) :
  - Les frappes ne s'affichent pas, ce sont des commandes
- Frappe (insertion ou remplacement) :
  - Les frappes s'ajoutent au texte, jusqu'à Esc
- Ex (ou commande…) :
  - Les frappes s'affichent en bas de l'écran, et se terminent avec la touche **Entrée** (↵)

Et… pourquoi vi c'est comme ça ???

Voici le ADM-3A utilisé par Bill Joy.
Notez :

- La facilité d'accès à . : @ et Esc
- Les flèches sous **HJKL**
- Le ~ sous "Home"

Source https://en.wikipedia.org/wiki/ADM-3A



Mais surtout, Bill Joy travaillait sur un modem à 300 bps ! Et sans souris !

Quelques commandes pour passer en mode frappe

i — insérer du texte à l'endroit du curseur
I — insérer avant le 1er caractère imprimable de la ligne

a — ajouter du texte après le curseur
A — ajouter du texte en fin de ligne

o — ouvre une nouvelle ligne en-dessous
O — ouvre une nouvelle ligne au-dessus

Il y en a quelques autres qui remplacent du texte (**R s S**)

On en sort avec ESC

Le concept essentiel

On revient toujours en mode normal… tôt ou tard !

Il ne faut pas penser qu'on utilise **i** pour passer en édition et Esc pour passer dans un mode où on peut faire des commandes.

Au lieu de cela, il faut penser à un ensemble commençant (par exemple) avec la commande **i** et qui prend en argument tout ce qui suit jusque Esc.

Premier inattendu

Donc oHello World^{Esc} va ajouter le texte Hello World.

La commande . va répéter la dernière commande.

On peut aussi préfixer une répétition: 3oHello World^{Esc}

Ça aussi on peut répéter: n.

P.S. Vous noterez l'usage des couleurs : l'invariant pour les commandes et le variable pour ce qui dépend de vous, N pour un chiffre ou nombre, n quand c'est un compteur optionnel (défaut 1), :next pour une fin non obligatoire.

Principales commandes de déplacement

- **hjkl** — les flèches marchent… normalement ! **nj** etc.
- **nw** — (word) avancer d'un mot (ou **W**, ≠ règles de "mot")
- **nb** — (back) reculer d'un mot (ou **B**, ≠ règles de "mot")
- **0** — début de ligne
- **$** — fin de ligne
- **gg** — début de fichier
- **G** — fin de fichier
- **NG** — aller à la ligne **N** (aussi **:N↵**)
- **N%** — aller aux **N**% du fichier depuis le début
- **(** — début de phrase (et **)** la fin)
- **{** — début de paragraphe (et **}** la fin)
- **%** — aller au délimiteur correspondant {[(<>)]}
- etc.

Plus énormément de commandes de contrôle écran comme **zz zt zb**

Principales commandes de recherche

- **/texte↵** — rechercher texte en avant
- **?texte↵** — rechercher texte en arrière
- ***** et **#** — rechercher mot sous curseur (avant, arrière)
- **n** — répéter recherche dans le même sens (**N**)
- **/↵** et **?↵** — répéter dans le sens indiqué

Sans oublier la recherche sur la ligne :

- **fc** — rechercher **c** en avant, en s'arrêtant sur c
- **tc** — rechercher **c** en avant, mais en s'arrêtant avant
- **Fc** et **Tc** — en arrière
- **;** refait et **,** change de sens

Tous prennent **n** en préfixe

Principales commandes d'édition… page 1 !

- **rX** — remplacer un caractère.
- **R** — entrer en mode frappe-recouvrement

- **nx** — supprimer un ou **n** caractères (**nX** en arrière)

- **nJ** — joindre deux ou **n** lignes.

- **d** pour delete… mais…

Principales commandes d'édition… page 2 !

- **d** — pour delete… mais on supprime quoi ?
  - Beaucoup de commandes ont un déplacement en paramètre
  - **dd** — suppression d'une ligne
  - **ndd** ou **dnd** — suppression de une ou **n** lignes
  - Avec déplacement :
    - **dw** (**ndw** ou **dnw**) — delete **n** words
    - **d$** (ou **D**) — suppression jusque fin de ligne
    - **dn/href**↲ — jusqu'au **n**ième "**href**"
    - **d}** — jusqu'à la prochaine ligne vide
    - **df"** — jusqu'au prochain "
    - etc. avec tout ce qu'on a vu avant
- **c** — changer, combinant suppression et mode insertion
  - Pareil que **d** : **C c$ ncc cnc ncw** etc.

En fonction de délimiteurs :

- Une commande (**dyc**…) suivie de **a** ou **i** et d'un délimiteur
  - **a** inclut le délimiteur, **i** l'exclut
- Délimiteurs :
  - Caractères simples : **([{ < "**
  - **w** et **W** (mot avec délimiteurs plus ou moins stricts)
  - **s** (sentence)
  - **p** (paragraph)
  - **t** (tag HTML / XML)
  - …
- Donc
  - **ci"** coupera/remplacera le texte à l'intérieur des **"**
  - **2ci{** coupera **2** niveaux de **{}**

- Avec un mouvement également : **<** et **>** — indentation

Copier-coller simple

- **x** et **d** et **c** sont en fait des “couper”
- **y** (yank) c’est copier
  - Et ça marche comme **d** et **c** bien sûr

- **np** — coller après le curseur (put ou paste)
- **nP** — coller avant le curseur (cf. **o** et **O** après/avant)
- **]p** — coller en adaptant l’indentation

- Exemples :
  - **yyp** duplique la ligne courante
  - **5yyP** duplique les **5** prochaines lignes

Mais on copie et on coupe quoi exactement ?

- C'est bien joli de compter les mots et les lignes !
- **v** suivi de déplacements (que ce soit flèches, recherche…) sélectionne des caractères
- **V** suivi de déplacements sélectionne des lignes
- Et on suit par une commande qui agira sur la sélection, comme si on avait spécifié un déplacement après !
- Donc par exemple **{V}d** va couper le paragraphe actuel
- Et… **Ctrl-V** sélectionne des zones rectangulaires.

Deuxième inattendu ! (Ou est-ce le troisième déjà ? Plus ?)

- Le “clipboard” n’est qu’un des “registres”, à savoir le "
- Il y en a 48 en tout !
- * pour le clipboard système
- 0 pour le dernier “yank”
- 1 pour le dernier coupé (> 1 ligne), 2–9 historique
- 26 registres a à z pour l’utilisateur
- On spécifie avec "r en normal:
  - "fdd — couper une ligne vers le registre f
  - "3p — coller depuis le registre 3
- On peut les afficher avec :reg↵ (:reg abcdef↵)
- Et… si on utilise une majuscule pour le registre, on y ajoute du texte au lieu d’écraser !

https://www.baeldung.com/linux/vim-registers

Le undo

u va annuler la dernière commande

Attention aux flèches cependant, ils terminent la commande.

Ctrl-R va rejouer la commande annulée

U va annuler les changements sur la ligne
U de nouveau va les remettre

Il y a des macros !

Ils se basent sur les registres, on peut donc écrire une macro dans son fichier, le copier dans un registre, et l'exécuter à partir de là.

- **qr** …tout ce qu'on veut… **q** — pour terminer et enregistrer
- **n@r** — pour exécuter
- **n@@** — répète la dernière macro
- Et comme pour coller, utiliser la majuscule **qR** signifie ajouter à la macro dans **r**

Petite astuce : **:set lazyredraw↵** pour que les macros aillent plus vite

Une… marque ?

- Techniquement, **v** et **V** posent des marques dans le fichier nommées **<** et **>**
- Avec la commande **mm** on peut aussi poser ses propres marques, désignées par **a** à **z** et de **A** à **Z**
- **`m** va à la marque et **'m** va au début de la ligne
- Les marques minuscules sont uniques au fichier
- Les marques majuscules sont uniques à votre compte
- Donc si vous posez votre marque **R** dans le **README** de votre projet, faire **`R** va directement fermer le fichier courant et ouvrir le **README**. **Ctrl-^** permet de revenir.
- Un déplacement qui change de ligne pose la marque **`**

Mais en mode frappe… on ne tape que du texte ?

Et bien il y a aussi des commandes avec **Ctrl** en mode frappe…

- **Ctrl-V**X insérer caractère spécial X (par exemple ^[ou ^c)
- **Ctrl-N** pour autocomplétion (↲ ou **Ctrl-Y** pour accepter)
- **Ctrl-X Ctrl-F** autocomplétion de nom de fichier
- **Ctrl-R**r insertion contenu registre r
- **Ctrl-U** comme **d0**
- **Ctrl-W** comme **dw**
- **Ctrl-O** exécute un commande en mode normal
- **Ctrl-K** insére digraph (voir **:digraph**↲)

Et les commandes ex

- ex est l'éditeur de lignes d'avant, le "pas visuel"
- On lance une commande ex avec :
- Syntaxe: :ScopeCommandeArguments↵
- Scope:
  - Par défaut ligne courante
  - % tout le fichier
  - début,fin (numéros de ligne, $ fin de fichier, marques 'm, . ligne actuelle, .+n…)
- Quelques commandes :
  - read — insérer un fichier
  - read! — lire la sortie d'une commande shell
  - write fichier — écrire un fichier (par défaut %)
  - ! — filtrer par une commande shell
  - s — filtrer par sed (là c'est un tout autre cours !)
- Exemple: :'f,$s,9\tHello,9\tBye Cruel,↵

Petit aparté sur la ligne du bas

Certaines commandes Ctrl marchent comme en mode frappe, par exemple:

- On peut insérer le contenu d'un registre avec **Ctrl-Rr** (par exemple pour faire un chercher/remplacer d'un caractère qui est dans le fichier mais qu'on ne sait pas taper au clavier)
- Ou faire l'autocompletion d'un nom de fichier avec Tab
- Ou faire **Ctrl-V** suivi de Tab pour en insérer un

Fichiers de configuration

- **.vimrc** contient vos instructions de lancement
  - On peut notamment faire ses propres commandes
- **.viminfo** est géré par vim (le contenu des registres, l'historique, les marques…)
- Il vaut mieux les éditer avec un autre éditeur, et/ou en faire des sauvegardes, juste au cas où !

Petites astuces classiques

- **:f↲** affiche des informations sur le fichier
- **:set hlsearch↲** et **:set nohlsearch↲** pour surligner ou pas la recherche (le tab marche pour l'autocompletion)
- **:set nowrap↲** pour ne pas wrapper les lignes logiques
- **:set tw=76↲** pour un retour automatique en fin de ligne
- **:set ts=4↲** pour la largeur des tabulations
- **Ctrl-A** pour incrémenter un nombre sous le curseur
- **Ctrl-X** pour décrémenter
- Tous les outils Unix:
  - **:r!seq 1 20**
  - **:r!cal 2026**
  - **:'<,'>!commande** | commande etc…

Petites astuces avec g

~ g~ gu gU — pour changer la casse

gi recommence l'insertion là où on en était sorti

g; revient au dernier changement (**g,** inverse, cf. **:changes**)

gj et **gk** déplacent le curseur en lignes écran, pas logiques

gf ouvre le fichier sous le curseur

Et beaucoup d'autres…

Bon, vi a 50 ans

- De multiples fichiers sur un écran
- De multiples fenêtres sur un même fichier
- Plier le texte: par ligne, par fonction, par colonne…
- Syntax highlighting
- Reformatage de texte, ou de code selon syntaxe
- Trouver votre fichier avec `:vim` ou `:f`
- Des scripts à n'en plus finir
  - Integration avec git
  - Coloriser le CSS
  - …
- Plugin pour discuter avec des LLM, sisi
- Tout ce que j'ai oublié
- Tout ce que je ne sais pas !

Tout ce que je ne sais pas !

- L'aide interne `:help↲` pour commencer (en sortir avec `:q`)
- Notez `:help votresujet↲`
- Ensuite, O'Reilly a publié un livre (bien sûr)
- vim.wikia.com
- vim.fandom.com
- r/vim
- vimdoc.sourceforge.net (dernière MÀJ en 2011…)
- Et plus généralement www.vim.org
- Avec ma très modeste contribution vi.lorens.org

Et une interview avec le créateur :
https://www.theregister.com/Print/2003/09/11/bill_joys_greatest_gift/

Les basiques, pour en sortir ! (Chose promise, chose due…)

Quand le fichier n'est pas modifié :

- **:q**uit — quitter
- **:n**next ou **:n**previous — fichier suivant ou précédent quand on en a spécifié plusieurs sur la ligne de commande (voir **:args**)

Quand le fichier est modifié, combiner avec **:w**rite pour écrire et/ou avec **!** pour forcer : **:q! :wn :n! :wq :wp!** Etc. Noter **:wp** marche mais **:prev** parce que **:p** est ambigu.

Voir aussi **:set autowrite**

:x et **ZZ** sont mieux que **:wq** : ils n'écrivent que si nécessaire.

ZZ