Vim

David Wagner
deubeuliou@gmail.com

Toulibre

23 Mai 2012

Vim en deux mots

- Éditeur de texte, visant l'ergonomie et l'extensibilité
- "Vi IMproved": version amélioré de l'éditeur historique "vi"
- En guerre sainte contre Emacs: éternel obscur troll libriste

Court historique

- Auteur: Bram Moolenaar
- Première version en novembre 1991
- Tout juste 20 ans !
- Version 7 en 2007, version actuelle (7.3) en 2010 (cf `ftp://ftp.vim.org/pub/vim/unix/`)
- Vim est libre mais est aussi un "charityware": vous êtes encouragés à faire un don à une association caritative aidant les enfants ougandais

Ergonomie

- Quelle est votre définition de l'ergonomie rapporté aux outils ?

Ergonomie

- Quelle est votre définition de l'ergonomie rapporté aux outils ?
- Wikipedia: "qui puissent être utilisés avec le maximum de confort, de sécurité et d'efficacité par le plus grand nombre"

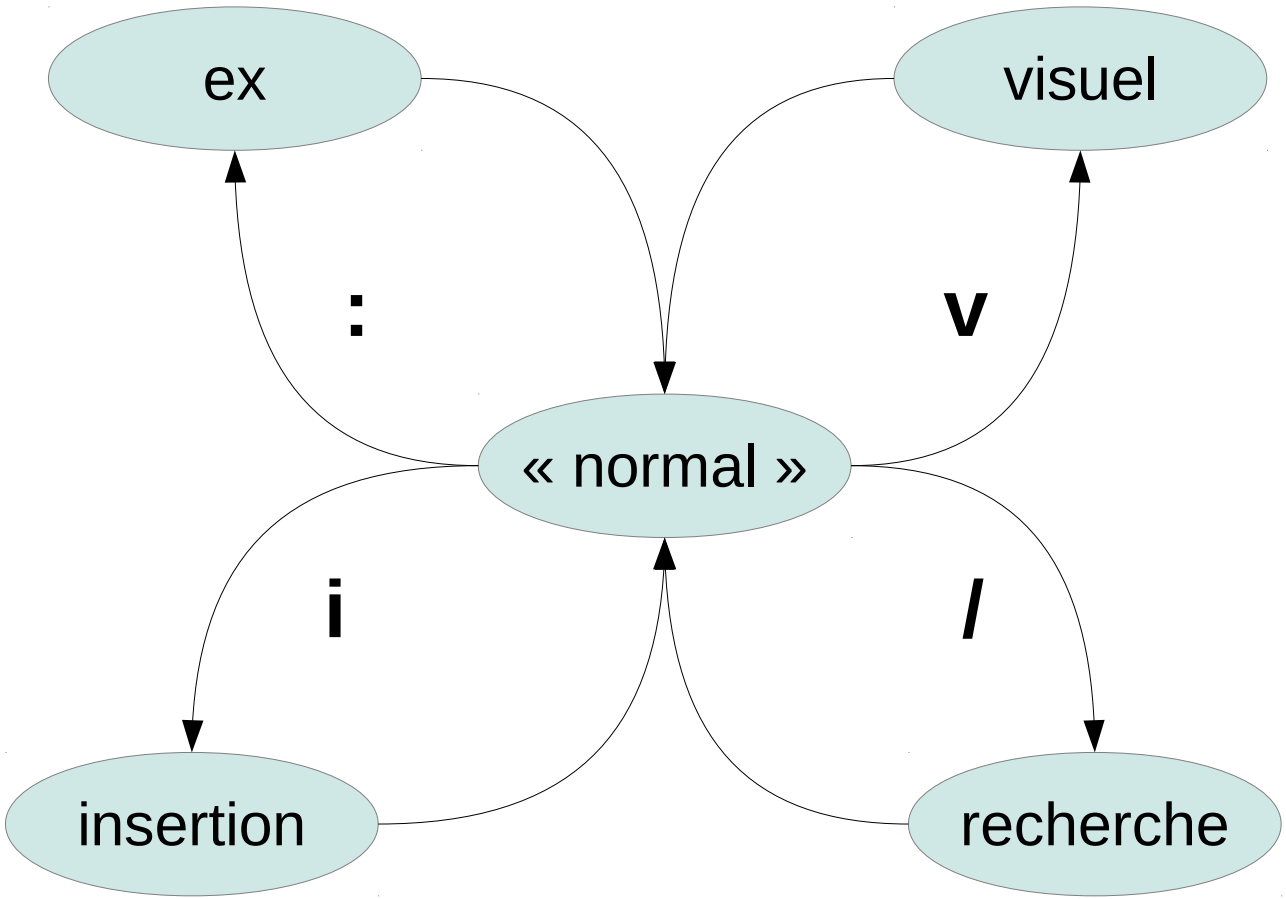
Ergonomie

- Quelle est votre définition de l'ergonomie rapporté aux outils ?
- Wikipedia: "qui puissent être utilisés avec le maximum de confort, de sécurité et d'efficacité par le plus grand nombre"
- Ergonomie à la Vim

C'est parti !

Les diapos sont principalement là pour résumer le propos à la fin de chaque partie et peut-être pour vous servir, plus tard, de référence.

Modes de Vim



Commandes de base

- `i` pour passer en mode insertion
- `Esc` pour retourner en mode normal (Ctrl-c, noté `^c` ou `C-c` parfois possible)
- pour `:w` pour en enregistrer, `:wq` ou `:x` pour enregistrer et quitter; `:q!` pour quitter sans enregistrer (le `!` permet de forcer l'exécution même en cas d'avertissement)
- pour `:e [fichier]` pour ouvrir un fichier (pensez à l'autocomplétion !)
- pour `:e` est un synonyme de `:edit`, `:w` de write, etc.

Mouvements

En théorie, on ne se déplace pas en mode insertion (Vi ne le permet pas). Mais Vim, par défaut, le permet.
Historiquement, il faut utiliser `h j k l` pour gauche, bas, haut, droite. Ça peut être utile à savoir dans certains cas.
Moyens plus rapides de se déplacer: utiliser un commande en mode "normal"

- `0 ^` pour aller au premier caractère/premier caractère non blanc
- `$` pour aller à la fin de la ligne
- `w b` pour aller au mot suivant/précédent

Certaines commandes ont un comportement différent en majuscule: `W B` pour aller au mot suivant/précédent en estimant que les mots sont séparés par des espaces

Couper/Copier/Coller

- commandes de la famille `y` pour copier
- commandes de la famille `d` pour couper
- commandes de la famille `p` pour coller

Toutes ces commandes utilisent les registres (voir `:registers`)
Une command peut être modifiée par (entre autres)

- un mouvement;
- un nombre entré avant la commands (noté `[count]`);
- une variable (vérifié dans l'aide - voir plus loin)

Recherche

Plusieurs manières de chercher:

- `/` `*` pour démarrer une recherche en avant, `?` `#` en arrière
- `n` `N` pour chercher le suivant/précédent

Il est possible d'utiliser des expressions régulières et certaines caractères particuliers avec `/`.
Le pattern recherché est stocké dans le registre `/`.

:s

Une des commandes les plus utiles de Vim: `:substitute`. (ou plus court: `:s`

Syntaxe

```
:s/pattern-re-recherche/pattern-de-remplacement/flags
```

Le pattern de remplacement peut contenir un grand nombre de caractères spéciaux (`:help sub-replace-special`).
Beaucoup de commandes peuvent être modifiées par une plage (`:help range`).
Plages utiles à connaître par coeur: `\%` représente le fichier entier; `.;+2` représente la plage démarrant à la ligne courante et jusqu'à 2 lignes plus loin.

Plusieurs façons de faire les choses

Il existe aussi des commandes (familles `c` et `s`) normales permettant de remplacer des parties de texte.
Il y a très souvent plusieurs solutions à un problème, avec Vim. À vous de trouver celle qui vous va le mieux.
Autres exemples, qui font gagner du temps sur les fautes de frappe: `r` et `~`

Mode Visuel

(Exemple: commenter du code rapidement ou le formater, exemple de commande modifiée par une variable)
Certaines commandes sont modifiées en mode visuel ; les commandes Ex ne s'appliquent qu'à la sélection.
Étendre la sélection à un élément significatif (mot, paragraphe, parenthèses, ...) est très rapide.

- `v` : mode visuel/caractère
- `^v` : mode visuel/block
- `V` : mode visuel/ligne

Tip: `^v` et `I` permet d'ajouter des caractères au début de chaque lignes.

L'aide de Vim

Très complète, découpée, indexée, facile à chercher.

- Recheche: `:help [sujet]`. l'autocomplétion marche ! Il existe aussi `:helpgrep`
- L'indexation de l'aide est faite grâce aux tags !

Aide à la recherche:

- `'variable' :commande-ex ^ctrl v_commande i_commande`

Quelques éléments de syntaxe de l'aide de Vim:

- `[count]` est le nombre entré avant la commande
- `{Visual}` signifie que la commande a été effectuée en mode Visuel
- `{range}` (commande ex) et `{motion}` (commande normale) sont la portion de texte concernée par la commande

Liens utiles

- vim.org
- vim.org/scripts
- http://vim.wikia.com
- stackoverflow.com

Cheat Sheets

- tnerual.eriogerg.free.fr/vimqrc.pdf
- www.glump.net/_media/howto/vi-vim-cheat-sheet-and-tutorial.pdf
- http://www.nathael.org/Data/vi-vim-cheat-sheet.svg version azerty

Vim pour le développeur

Après avoir vu les bases, voyons maintenant plus en détail certaines parties de Vim qui vous permettront d'avoir un flux de travail agréable.

Fenêtres et onglets

`:split` et famille `:tab`

Certaines commandes en rapport avec les fenêtres peuvent être modifiées par `:vert`.

Les commandes en rapport avec les fenêtres commencent par `^w`.

(Vous avez remarqué, on peut utiliser `Ctrl` dans les commandes ! Alt (`Meta`), `Shift` et `F<n>` aussi)

L'art d'ouvrir un fichier

- Commandes `:e`, `:e!` (pour recharger un fichier).
- Commandes `:pwd`, `:file`, `:cd`, expression magique
- Navigateur de fichiers intégré
- Ouverture de fichiers PDF, d'archives, etc. (plugins fourni dans toutes les bonnes distros)

Options à l'ouverture:

- Ouvrir plusieurs fichiers (`-p` en onglets, `-o -O` en splits horizontaux/verticaux)
- Ouvrir à un tag `-t <tag>`/une ligne `+[N]`/une recherche `+/<pattern>` donnée

Lecture depuis stdin.
`scp://` et `ftp://`

Récupérer après un crash

`vim -r` pour vérifier l'existence d'un fichier de swap et l'utiliser le cas échéant.
À plus grande échelle (pour conserver tout un environnement de travail), les sessions sont très pratiques: `:mksession [fichier]` (par défaut: `Session.vim` dans le dossier courant). Pour récupérer une session: `vim -S`
Une session ne concerne pas que ce qui est visible, mais aussi la configuration !

Ligne de commande avancée

- Possède un historique
- Autocomplète un grande quantité d'objets (le comportement se configure via 'wildmode'
- Historique (`:history`, les flèches haut/bas servent aussi de recherche dans l'historique)
- "Coller" dans la ligne de commande: `^r` suivi du registre à coller. Permet d'utiliser `d`/`y` conjointment à la ligne de commande.

Lire/Écrire des options:

- `:set option` ou `:set nooption` pour les options booléennes
- `:set option=<valeur>` pour les autres
- `:set option?` pour voir la valeur d'une option
- `:let @registre=<valeur>` pour setter un registre

`:options` pour avoir un aperçu de TOUTES les options existantes :)

Sauvegarder sa configuration

`/usr/share/vim/vim73` (par exemple) contient le runtime (cf `'runtimepath'`), c'est à dire toute la partie dynamique (les plugins, les syntaxes, la coloration, etc.)
`~/.vim` est un répertoire avec la même arborescence, réservée à l'utilisateur, pour installer des scripts, thèmes, etc. localement
La confguration vim est généralement dans `~/.vimrc`. Elle peut aussi être sourcée automatiquement, suivant la valeur de `'exrc'`, dans un fichier `.exrc` (par exemple) dans le dossier courant. `:mkexrc` permet de créer un fichier à ce nom contenant la configuration courante.
Un commentaire à la fin d'un fichier, avec un format particulier, peut aussi être utilisé.

Configuration par projet

Si votre projet requiert une configuration particulière, ça peut être l'occasion d'utiliser exrc. Votre configuration doit contenir l'option `'exrc'`.
Mon conseil: lancer Vim depuis la racine du projet ; avoir un seul projet dans une instance de Vim (et utilise les onglets).
Collaboration avec Git: avoir un "exrc" par projet git et trouver son emplacement grâce à la commande git adéquate ? (exercice laissé au lecteur)

Vos propres mappings

`:map` vous permet de (re)définir vos raccourcis !
`:imap` ne concerne que le mode insertion, `:nmap` le mode normal, `:vmap` le mode visuel (et il en existe d'autres).

Exemple d'utilisation

`map <Tab> gt` map la touche Tab sur la commande `gt` (onglet suivant)

Le net regorge de plugins qui font le café ! N'hésitez pas à faire votre marché.
Ma recommendation: trouver une config déjà faite, pas trop compliquée, la comprendre, et l'améliorer au fur et à mesure.

Utiliser des outils externes

- Options à l'ouverture, lecture depuis stdin (déjà vu)
- `{Visual}!<commande>` envoie la sélection sur l'entrée standard d'une commande et la remplace par la sortie de la commande
- `:w !<commande>` envoie le fichier (ou la sélection, le cas échéant) sur l'entrée d'une commande et affiche (en bas) la sortie. Par exemple `:'<'>w !wc -c` pour compter les caractères dans la sélection
- `:r <file>` ou `:r !<cmd>`

Outils à l'intégration poussée:

- `:make` (cf `'makeprg'`)
- `:grep` (cf `'grepprg'`) (Vim a aussi une version builtin `:vimgrep`)
- la commande = peut aussi utiliser une commande externe (cf `'equalprg'`)

Automatiser la saisie

`.` permet de répeter une commande.
Une macro s'enregistre avec `q<registre>`, s'arrête avec `q` et se rejoue avec `[count]@`
La commande `:g/<expression>/<commande-ex>` exécute la commande spécifiée sur chaque ligne matchant `<expression>`. La commande `:v` fait la même chose, mais sur les lignes ne matchant pas.
Origine de grep: `:g/re/p` (pour chaque ligne correspondant à l'expression régulière, éxécute `:print`) !!!

Outils externes / tags

Vim peut utiliser "ctags" pour sauter facilement à des définitions de fonctions, classes, etc.
Il faut au préalable créer le fichier de tags: `ctags -R .` (créé un fichier "tags".

- `^]` pour sauter à la définition (empile le tag sur un historique. cf `:tags`)
- `^t` pour dépiler l'historique

Il existe une "autocomplétion naïve" du mot précédent le curseur: `^p` ou `^n`
OmniCompletion est plus intelligent et utilise les tags: `^x^o`
De nombreux plugins utilisent les tags. Par exemple `TagList`.

Plus loin

- souris
- fonctions builtin/user-defined
- GVim, menus "popup"
- communication entre instances de Vim

Dernier conseil

- Trouver/implémenter une solution
- Identifier une tâche répétitive/rébarbative
- En faire une habitude

Dernières questions ?