eBook Gratuit

APPRENEZ vim

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#vim

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: vim

It is an unofficial and free vim ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official vim.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec vim

Remarques

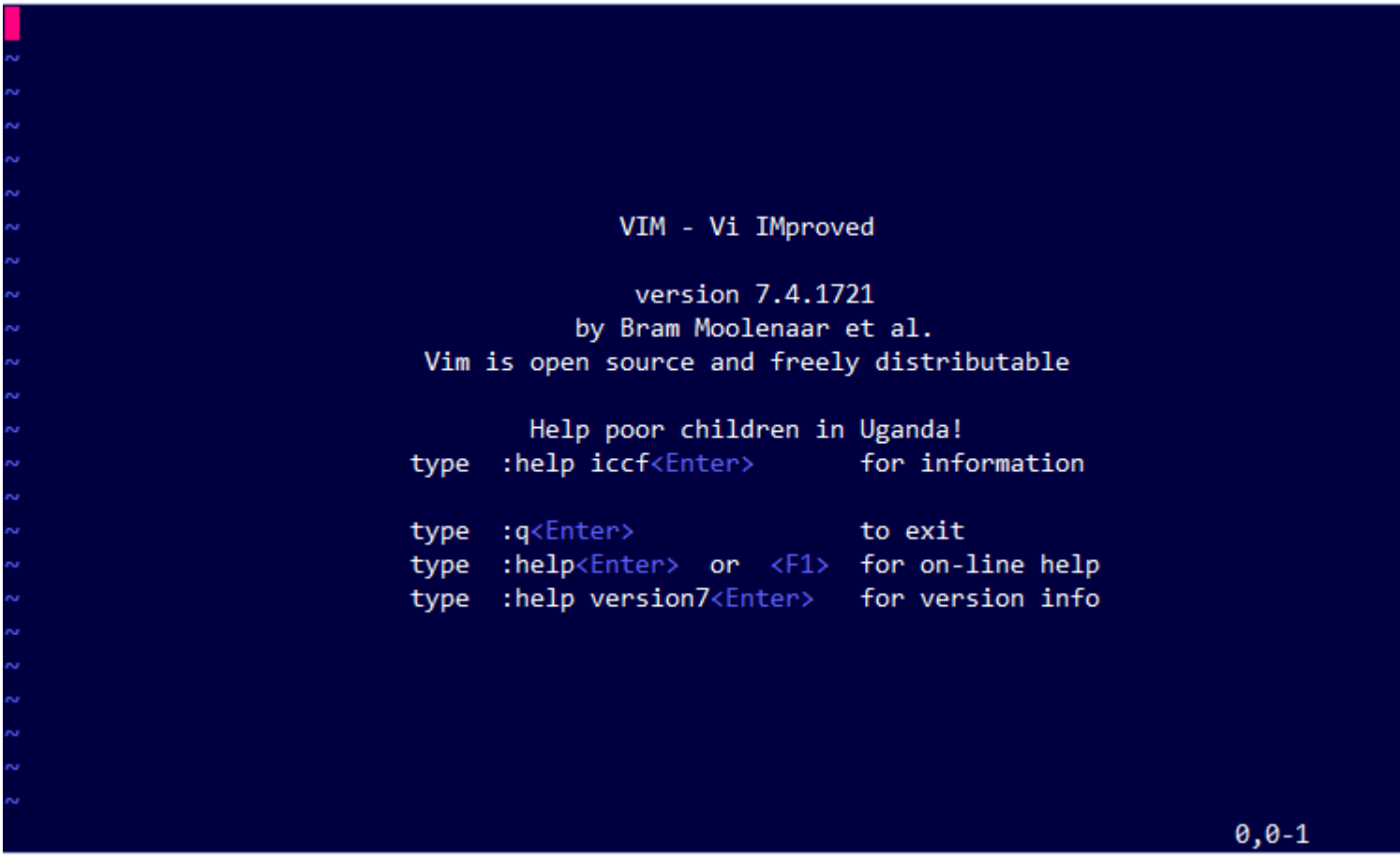
Vim (ou "Vi IMproved") est un éditeur de texte multi-mode (*modal*) basé sur console. Il est largement utilisé et disponible par défaut sur tous les systèmes Unix, Linux et Apple OS X. Vim a une grande communauté active et une large base d'utilisateurs. L'éditeur prend en charge tous les langages de programmation courants et de nombreux plugins sont disponibles pour étendre ses fonctionnalités.

Les développeurs apprécient l'éditeur pour sa rapidité, ses nombreuses options de configuration et sa puissante édition basée sur des expressions. En mode "commande", l'éditeur est contrôlé par des commandes clavier, de sorte que l'utilisateur n'est pas distrait par une interface graphique ou un pointeur de souris.

Vim est basé sur l'ancien éditeur Unix "vi" créé dans les années 70 et il est en développement continu depuis 1991. Avec les macros et les plugins, l'éditeur offre la plupart des fonctionnalités d'un environnement de développement intégré. Il est également capable de traiter de grandes quantités de texte avec son langage de script (vimscript) et ses expressions régulières.

Sujets principaux:

- installation
- modes d'édition
- la navigation
- édition de base
- édition avancée
- configuration
- plugins
- vimscript
- les macros
- communauté
- Projets liés



Versions

Version	Date de sortie
8.0	2016-09-12
7,4	2013-08-10
7.3	2010-08-15
7.2	2008-08-09
7.1	2007-05-12
7.0	2006-05-07
6,0	2001-09-26
5.0	1998-02-19
4.0	1996-05-29
3.0	1994-08-12
2.0	1993-12-14

Version	Date de sortie
1,14	1991-11-02

Examples

Installation

Le Vim sur votre machine - s'il y en a un - est très probablement une "petite" version qui ne dispose pas de fonctionnalités utiles telles que le support de presse-papiers, la coloration syntaxique ou même la possibilité d'utiliser des plugins.

Ce n'est pas un problème si tout ce dont vous avez besoin est un moyen rapide d'éditer des fichiers de configuration, mais vous allez bientôt atteindre un certain nombre de murs si vous avez l'intention de faire de Vim votre éditeur principal.

Il est donc généralement recommandé d'installer une version complète.

Installation sous Linux / BSD

Sur ces systèmes, l’astuce consiste simplement à installer la version GUI fournie avec une commande `gvim` pour démarrer l’interface graphique et une commande `vim` pour démarrer l’UTI.

Distributions basées sur Arch et Arch

```
$ sudo pacman -R vim
$ sudo pacman -S gvim
```

Distributions basées sur Debian et Debian

```
$ sudo apt-get update
$ sudo apt-get install vim-gtk
```

Distributions basées sur Gentoo et Gentoo

```
$ sudo emerge --sync
$ sudo emerge app-editors/gvim
```

Distributions basées sur RedHat et RedHat

```
$ sudo yum check-update
$ sudo yum install vim-X11
```

Feutre

```
$ sudo dnf check-update
$ sudo dnf install vim-X11
```

Distributions basées sur Slackware et Slackware

```
$ sudo slackpkg update
$ sudo slackpkg install-new vim-gvim
```

Distributions basées sur OpenBSD et OpenBSD

```
$ sudo pkg_add vim-x11
```

FreeBSD et distributions basées sur FreeBSD

```
$ sudo pkg install editors/vim
```

Installation sur Mac OS X

La stratégie est similaire à Mac OS X: nous installons la version de l'interface graphique pour obtenir à la fois l'interface graphique et l'interface utilisateur graphique. Au final, nous devrions pouvoir:

- double-cliquez sur l'icône MacVim dans le Finder,
- cliquez sur l'icône MacVim dans le Dock,
- `$ mvim` dans le shell pour ouvrir l'interface graphique de MacVim,
- `$ mvim -v` dans le shell pour ouvrir l'interface utilisateur de MacVim.

Installation régulière

Téléchargez et installez un instantané officiel comme vous le feriez avec toute autre application Mac OS X.

Placez le script `mvim` fourni avec MacVim quelque part dans votre `$PATH` .

Directeur chargé d'emballage

MacPorts:

```
$ sudo port selfupdate
$ sudo port install macvim
```

Homebrew:

```
$ brew install macvim
```

Pour faire de MacVim la console par défaut Vim:

```
$ brew install macvim --with-override-system-vim
```

Installation sous Windows

Il n'y a pas de Vim sur les systèmes Windows par défaut. Vous pouvez télécharger et installer Vim depuis le site de Tuxproject pour obtenir des versions plus récentes et complètes, ou vous pouvez télécharger et installer Vim à partir du site officiel de Vim .

Chocolaté

```
> choco install vim
```

Construire Vim depuis la source

Si les méthodes ci-dessus ne vous conviennent pas, il est toujours possible de créer Vim vous-même, avec *uniquement* les options dont vous avez besoin.

Ce sujet sera discuté dans sa propre section (actuellement en projet).

Sortie de Vim

Pour quitter Vim, assurez-vous d'être en mode *Normal* en appuyant sur `Echap` .

- `:q Entrez` (vous empêchera de quitter si vous avez des modifications non enregistrées - raccourci pour: quitter)

Pour *ignorer les* modifications et quitter Vim:

- `:q! Entrez` pour forcer la sortie et ignorer les modifications (abréviation de `:quit!` ne pas confondre avec `:!q`),
- `ZQ` est un raccourci qui fait la même chose que `:q!` ,
- `:cq Entrez` quit and return error (ignorez toutes les modifications pour que le compilateur ne recompile pas ce fichier)

Pour *enregistrer les* modifications et quitter Vim:

- `:wq Enter` (raccourci pour `:write` et `:quit`),
- `:x Entrez` (identique à `:wq` , mais n'écrira pas si le fichier n'a pas été modifié),
- `ZZ` est un raccourci qui fait la même chose que `:x` (Enregistrer l'espace de travail et quitter l'éditeur),
- `:[range]wq! Entrez` (écrivez les lignes dans [plage])

Pour fermer plusieurs tampons à la fois (même dans plusieurs fenêtres et / ou onglets), ajoutez la lettre `a` à l'une des *commandes* ci-dessus (celles commençant par `:` . Par exemple, pour écrire et quitter toutes les fenêtres que vous pouvez utiliser:

- `:wqa Enter` ou

- `:xa Enter` - Ecrit tous les tampons modifiés et quitte Vim. S'il y a des tampons sans nom de fichier, qui sont en lecture seule ou qui ne peuvent pas être écrits pour une autre raison, Vim ne quittera pas
- `:xa! Enter` - Ecrivez tous les tampons modifiés, même ceux qui sont en lecture seule, et quittez Vim. S'il y a des tampons sans nom de fichier ou qui ne peuvent pas être écrits pour une autre raison, Vim ne quittera pas
- `:qa Enter` - essayez de quitter, mais arrêtez s'il y a des fichiers non enregistrés;
- `:qa! Entrez` - quitter *sans enregistrer* (Rejeter les modifications dans *les* fichiers non enregistrés)

Si vous avez ouvert Vim sans spécifier de fichier et que vous souhaitez enregistrer ce fichier avant de quitter, vous recevrez `E32: No file name` message de `E32: No file name` . Vous pouvez enregistrer votre fichier et quitter en utilisant:

- `:wq filename Entrée` ou;
- `:x filename Entrée`

Explication:

La touche `:` ouvre réellement le mode *Commande* . La commande `q` est une abréviation de `quit` , `w` , de `write` et `x` , de `exit` (vous pouvez aussi taper `:quit` `:write` et `:exit` si vous voulez). Les raccourcis *ne* commençant *pas* par `:` tels que `ZZ` et `ZQ` rapportent aux mappages de touches en mode *normal* . Vous pouvez les considérer comme des raccourcis.

Le `!` La frappe au clavier est parfois utilisée à la fin d'une commande pour forcer son exécution, ce qui permet de supprimer les modifications dans le cas de `:q!` . Placer le `!` au début de la commande a un sens différent. Par exemple, on peut taper:! `:!q` au lieu de `:q!` et vim se terminerait par une erreur 127.

Un moyen facile de s'en souvenir est de penser à `!` comme moyen d'insister sur l'exécution de quelque chose. Tout comme lorsque vous écrivez: "Je veux quitter!"

Tutoriels interactifs Vim (tels que vimtutor)

`vimtutor` est un didacticiel interactif couvrant les aspects les plus fondamentaux de l'édition de texte.

Sur un système de type UNIX, vous pouvez lancer le didacticiel avec:

```
$ vimtutor
```

Sous Windows, «Vim tuteur» se trouve dans le répertoire «Vim 7.x» sous «Tous les programmes» dans le menu Windows.

Voir `:help vimtutor` pour plus de détails.

Parmi les autres didacticiels interactifs, citons ceux basés sur un navigateur:

- Vim Adventures - Une version de jeu interactive de vimtutor disponible sur le web. Seuls les premiers niveaux sont gratuits.
- Open Vim - Un terminal interactif qui vous apprend les commandes de base avec feedback.
- Vim Genius - Un autre terminal interactif qui comprend également des leçons intermédiaires et avancées, y compris des macros et des arglist.

Enregistrement d'un fichier en lecture seule édité dans Vim

Parfois, nous pouvons ouvrir un fichier que nous n'avons pas la permission d'écrire dans Vim sans utiliser `sudo` .

Utilisez cette commande pour enregistrer un fichier en lecture seule modifié dans Vim.

```
:w !sudo tee > /dev/null %
```

Ce que vous pourriez cartographier `:w!!` dans votre `.vimrc` :

```
cmap w!! w !sudo tee > /dev/null %
```

Vous serez présenté une invite comme indiqué dans l'image.

.

Appuyez sur `o` et le fichier sera enregistré. Il reste ouvert dans vi / vim pour plus d'édition ou de lecture et vous pouvez quitter normalement en tapant `:q!` puisque le fichier est toujours ouvert en lecture seule.

Explication de commande

```
:w ........................... isn't modifying your file in this case,
   ........................... but sends the current buffer contents to
   ........................... a substituted shell command
   !sudo ..................... call the shell 'sudo' command
         tee .................. the output of the vi/vim write command is redirected
                                using the 'tee' command
             > /dev/null ...... throws away the standard output, since we don't need
                                to pass it to other commands
                          % .... expands to the path of the current file
```

Sources:

- Blog technique d'Adam Culp
- Stackoverflow, Comment fonctionne le vim "écrit avec sudo"?

Suspendre vim

Lorsque vous utilisez `vim` partir de la ligne de commande, vous pouvez suspendre `vim` et revenir à votre invite sans quitter réellement `vim` . Par conséquent, vous pourrez plus tard récupérer votre session `vim` partir de la même invite.

En *mode normal* (sinon, appuyez sur `esc` pour y arriver), lancez l'une des commandes suivantes:

```
:st entrer

:sus entrer

:stop entrer

:suspend entrer
```

Alternativement, sur certains systèmes, en mode *Normal* ou *Visual* , l'émission de `Ctrl` + `z` aura le même effet.

Remarque: Si `autowrite` est défini, les tampons avec les modifications et les noms de fichiers seront écrits. Ajoutez un `!` avant d' `entrer` pour éviter, par exemple. `:st! entrez` .

Plus tard, lorsque vous souhaitez revenir à votre session `vim` , si vous n'avez pas suspendu d'autres travaux, la publication de ce qui suit restaurera vim en tant que tâche de premier plan.

```
fg entrer
```

Sinon , vous devrez trouver votre `vim` ID d'emploi de sessions en émettant des `jobs entrent` et mettant en avant les emplois correspondant `fg %[job ID] entrer` par exemple. `fg %1 entrée` .

Les bases

Exécutez des tutoriels interactifs vim autant de fois que nécessaire pour vous familiariser avec les bases.

Vim dispose de plusieurs modes, par exemple **mode normal, mode insertion** et **mode ligne de commande** .

Le mode normal est pour l'édition et la navigation du texte. Dans ce mode, `h` , `j` , `k` et `l` correspondent aux touches du curseur ← , ↓ , ↑ et → . La plupart des commandes en mode normal peuvent être précédées d'un "count", par exemple `3j` descend de 3 lignes.

Le mode d'insertion permet d'insérer le texte directement. Dans ce mode, vim est similaire à d'autres éditeurs de texte plus simples. Pour entrer en mode insertion, appuyez sur `i` en mode normal. Pour le quitter, appuyez sur `<ESC>` (touche d'échappement).

Le mode ligne de commande permet d'exécuter des commandes plus complexes, telles que l'enregistrement du fichier et la fermeture de vim. Appuyez sur `:` pour démarrer le mode ligne de commande. Pour quitter ce mode, vous pouvez également appuyer sur `<ESC>` . Pour enregistrer les modifications apportées au fichier, utilisez `:w` (ou `:write`). Pour quitter vim sans enregistrer vos

modifications, utilisez `:q!` (ou `:quit!`).

Voici quelques-unes des commandes les plus utiles de vim:

Commander	La description
`i`	(insert) passe en mode insertion *avant* la position actuelle du curseur
`I`	passe en mode insertion *avant* le premier caractère imprimable de la ligne en cours
`a`	(append) passe en mode insertion *après* la position actuelle du curseur
`A`	passe en mode insertion *après* le dernier caractère imprimable de la ligne en cours
`x`	supprimer le caractère à la position actuelle du curseur
`X`	supprimer le caractère à gauche de la position actuelle du curseur
`w`	passer au mot suivant
`b`	passer au mot précédent
`0`	passer au début de la ligne
`$`	passer à la fin de la ligne
`r`	replace - entre le mode de remplacement pour un caractère. Le prochain caractère que vous tapez remplacera le caractère sous le curseur.
`R`	entre en mode de remplacement indéfiniment. Chaque caractère que vous tapez remplace le caractère sous le curseur et fait avancer le curseur de un.
`s`	substitute - supprime le caractère à la position actuelle du curseur puis passe en mode insertion
`S`	efface la ligne actuelle sur laquelle se trouve le curseur et passe en mode insertion
`<Esc>` , `<Cc>`	quitter le mode insertion et revenir au mode normal
`u`	annuler
`<Cr>`	refaire
`dd` , `dw` , `dl` , `d$`	couper la ligne en cours, du curseur au mot suivant ou le caractère, la position actuelle à la fin de la ligne en cours respectivement, note: `D` est l'équivalent de `d$`
`cc` , `cw` , `cl`	changer la ligne actuelle, du curseur au mot suivant ou au caractère,

Commander	La description
	respectivement
`yy` , `yw` , `yl` , `y$`	yank ("copier") la ligne courante, du curseur au mot suivant, ou le caractère, la position actuelle à la fin de la ligne actuelle respectivement
`p` , `P`	mettre ("coller") après ou avant la position actuelle, respectivement
`o` , `O`	pour créer une nouvelle ligne vide, après ou avant celle en cours et entrer en mode insertion
`:w`	écrire le tampon actuel sur le disque
`:q!` , `ZQ`	quitter sans écrire
`:x` `:wq` , `ZZ`	écrire et quitter
`:help`	ouvrir une fenêtre avec le fichier d'aide
`:help {subject}`	montrer de l'aide pour un sujet spécifique
`qz`	Commencez à enregistrer les actions pour enregistrer `z` , `q` pour terminer l'enregistrement, `@z` pour lire les actions. `z` peut être n'importe quelle lettre: `q` est souvent utilisé pour des raisons de commodité. En savoir plus: Macros

Que faire en cas de crash

Vim enregistre toutes vos modifications non enregistrées dans un *fichier d'échange* , un fichier supplémentaire qui est supprimé une fois que les modifications sont validées par l'enregistrement. Le nom du fichier d'échange est généralement le nom du fichier en cours de modification précédé d'un `.` et avec un suffixe `.swp` (vous pouvez le voir avec `:sw`).

Donc, si votre processus vim se termine avant que vous ayez eu la possibilité de sauvegarder vos modifications, vous pouvez récupérer votre travail en appliquant les modifications contenues dans le fichier d'échange à votre fichier actuel en utilisant l'option de ligne de commande `-r` . Par exemple, si `myFile` est le fichier que vous éditez, utilisez:

```
$ vi -r myFile
```

pour récupérer les modifications non validées.

Si un fichier d'échange existe, vim devrait vous demander quand même des options de récupération.

```
$ vi myFile
E325: ATTENTION
Found a swap file by the name ".myFile.swp"
...
```

```
Swap file ".myFile.swp" already exists!
[O]pen Read-Only, (E)dit anyway, (R)ecover, (D)elete it, (Q)uit, (A)bort:
```

Si vous choisissez (R) ecover, les modifications du fichier `swp` sont appliquées, mais le fichier d'échange ne sera pas supprimé. N'oubliez donc pas de supprimer le fichier d'échange si vous êtes satisfait de la récupération.

Lire Démarrer avec vim en ligne: https://riptutorial.com/fr/vim/topic/879/demarrer-avec-vim

Chapitre 2: :global

Syntaxe

- `:[<range>]g[lobal]/{<pattern>}/[<command>]`
- `:[<range>]g[lobal]!/{<pattern>}/[<command>]` (inversé)
- `:[<range>]v[global]/{<pattern>}/[<command>]` (inversé)

Remarques

La commande "globale" de Vim est utilisée pour appliquer une commande ex à chaque ligne correspondant à une expression régulière.

Examples

Utilisation basique du Global Command

```
:g/Hello/d
```

Supprimer toutes les lignes contenant le texte "Bonjour". **Remarque importante** : ce n'est pas la commande de mode normal `d` , c'est la commande ex `:d` .

Vous pouvez utiliser la commande globale pour appliquer des frappes de mode en mode normal au lieu des commandes ex en ajoutant `normal norm` ou la `norm` à la commande. Par exemple:

```
:g/Hello/norm dw
```

Supprime le premier mot de chaque ligne contenant le texte "Hello".

La commande globale prend également en charge le mode visuel et les plages .

Yank chaque ligne correspondant à un motif

La commande

```
:g/apples/y A
```

toutes les lignes seront Yank contenant le mot *pommes* dans le `a` registre, qui peut être collé avec `"ap` . Toute expression régulière peut être utilisé.

Notez l'espace avant le `A` et la capitalisation de la lettre de registre. Si une lettre majuscule est utilisée comme registre, les correspondances seront *ajoutées* à ce registre. Si une lettre minuscule est utilisée, seul le *dernier* match sera placé dans ce registre.

Déplacer / collecter des lignes contenant des informations clés

une commande simple mais très utile:

```
:g/ending/m$
```

déplace les lignes contenant la `ending` à la fin du tampon.

`m` signifie déplacer
`$` signifie la fin du tampon, tandis que `0` signifie le début du tampon.

Lire :global en ligne: https://riptutorial.com/fr/vim/topic/3861/-global

Chapitre 3: Amélioré défaire et refaire avec un undodir

Examples

Configurer votre vimrc pour utiliser un undodir

Depuis la version 7.3 de vim, la fonctionnalité 'persistent_undo' est prise en charge, ce qui permet d'annuler / rétablir les modifications, même après la fermeture de vim ou le redémarrage de votre ordinateur.

Il est possible de le configurer en ajoutant ce qui suit à votre vimrc, mais commencez par créer un répertoire où vos fichiers sous-jacents doivent être enregistrés. Vous pouvez créer le fichier n'importe où, mais je recommande d'utiliser le répertoire ".vim".

```
if has('persistent_undo')         "check if your vim version supports
  set undodir=$HOME/.vim/undo     "directory where the undo files will be stored
  set undofile                    "turn on the feature
endif
```

Après avoir ajouté ceci à votre vimrc et après avoir à nouveau fourni le vimrc, vous pouvez utiliser la fonctionnalité en utilisant les commandes de base undo / redo

Lire Amélioré défaire et refaire avec un undodir en ligne:
https://riptutorial.com/fr/vim/topic/7875/ameliore-defaire-et-refaire-avec-un-undodir

Chapitre 4: Avantages de vim

Examples

Personnalisation

L'avantage d'utiliser **vim** sur un simple éditeur de texte comme **notepad** ou **gedit** est qu'il permet à l'utilisateur de personnaliser presque tout ce qui le concerne. Si vous vous retrouvez à faire des actions à maintes reprises, vim possède une multitude de fonctionnalités qui vous aideront à effectuer cette action plus rapidement et plus facilement.

La plupart des IDE populaires tels que **MS Visual Studio** ou **IntelliJ IDEA** fournissent à leurs utilisateurs des raccourcis utiles et même une certaine quantité de personnalisation, mais ils sont généralement liés à des actions spécifiques courantes dans un contexte particulier, alors que vim permet de personnaliser différents situations sans se contredire. Il serait peut-être confortable de développer des programmes c ++ dans Visual Studio et Java dans IntelliJ, mais vous n'écririez pas de code python, et bien sûr, il existe un autre IDE, mais dans vim vous pouvez modifier n'importe quelle langue sans perdre la commodité.

Il y a bien sûr d'autres éditeurs personnalisables et je ne suis pas le seul à dire que vim est le meilleur pour tout le monde. C'est une question de préférence personnelle. Je ne pense pas que quelqu'un dira **qu'emacs** permet un niveau de personnalisation inférieur à celui de vim (et certains diraient le contraire), mais il faut vraiment l'essayer pour trouver ce qui vous convient le mieux.

Certains disent qu'ils ne veulent pas passer des mois à apprendre à utiliser un éditeur, juste pour pouvoir y travailler. Mais ceux qui le font, la plupart du temps d'accord, ont estimé que cela en valait la peine. Pour moi, ce n'est jamais un problème, apprendre de nouvelles choses sur vim et gagner en efficacité est tout simplement amusant. Et il y a beaucoup à apprendre.

Poids léger

Vim est léger (comme GNU Nano ou GNU emacs). Il n'a besoin d'aucune interface graphique (comme x11, wayland & co).

Cela rend vim à un mainteneur de système meilleur ami. Vous pouvez l'utiliser avec ssh et c'est très important sur de très petits périphériques qui n'ont pas d'interface graphique.

La programmation et la maintenance des serveurs distants ont pris de plus en plus d'importance au cours des dernières années et l'utilisation de vim (ou emacs) est le meilleur moyen de le faire.

Contrairement à de nombreux IDE, vim offre la possibilité de travailler avec de nombreux types de fichiers prêts à l'emploi et d'écrire vos propres commandes et la syntaxe hl est facile.

Et last but not least, un utilisateur vim doit pouvoir utiliser vi, qui est préinstallé sur la plupart des systèmes UNIX.

Lire Avantages de vim en ligne: https://riptutorial.com/fr/vim/topic/9653/avantages-de-vim

Chapitre 5: Bâtiment de vim

Examples

Commencer une construction

`:mak[e][!] [arguments]` lancera le programme auquel se `makeprg` option `makeprg` . Par défaut, `makeprg` est défini sur "make", mais peut être configuré pour appeler tout programme approprié.

Tous les `[arguments]` (peuvent être plusieurs) sont passés à `makeprg` comme s'ils avaient été `:!{makeprg} [arguments]` avec `:!{makeprg} [arguments]` .

La sortie du programme appelé est analysée pour les erreurs selon l'option `'errorformat'` . Si des erreurs sont détectées, la fenêtre de correction rapide s'ouvre pour les afficher.

`:cnext :cprev` peut être utilisé pour parcourir les erreurs affichées dans la fenêtre de correction rapide. `:cc` passera à l'erreur sous le curseur.

Il convient de noter que sur les systèmes où gnumake est installé et correctement configuré, il n'est généralement pas nécessaire de définir `&makeprg` sur autre chose que sa valeur par défaut pour compiler des projets mono-fichier. Ainsi, en C, C ++, Fortran ... il suffit de taper `:make %<` pour compiler le fichier en cours. Selon que le fichier source est dans le répertoire courant, `:!./%<` l'exécutera. Les options de compilation peuvent être contrôlées via `$CFLAGS` , `$CXXFLAGS` , `$LDFLAGS` , etc. Consultez la documentation de `make` concernant les *règles implicites* .

Lire Bâtiment de vim en ligne: https://riptutorial.com/fr/vim/topic/3321/batiment-de-vim

Chapitre 6: Code de format automatique

Examples

En mode normal:

En mode normal:

`gg` *aller en haut* `=` alors `G`

Lire Code de format automatique en ligne: https://riptutorial.com/fr/vim/topic/7931/code-de-format-automatique

Chapitre 7: Commandes Automatiques

Remarques

Commandes d' `autocmd` Surround

`autocmd` est une commande additive et vous ne voulez probablement pas ce comportement par défaut.

Par exemple, si vous recréez votre `.vimrc` plusieurs fois en le modifiant, vim peut ralentir.

Voici la preuve:

```
:autocmd BufWritePost * if &diff | diffupdate | endif " update diff after save
:autocmd BufWritePost * if &diff | diffupdate | endif " update diff after save
```

Si vous tapez maintenant `:autocmd BufWritePost *` , vous verrez les **deux** lignes dans la sortie, pas juste une. Les deux sont exécutés.

Pour éviter ce comportement, entourez tous vos `autocmd` comme suit:

```
if has ('autocmd')       " Remain compatible with vi which doesn't have autocmd
  augroup MyDiffUpdate   " A unique name for the group.  DO NOT use the same name twice!
      autocmd!           " Clears the old autocommands for this group name
      autocmd BufWritePost * if &diff | diffupdate | endif   " Update diff after save
      " ... etc ...
  augroup END
endif
```

Examples

Source automatique .vimrc après enregistrement

Ajoutez ceci à votre `$MYVIMRC` :

```
" Source vim configuration file whenever it is saved
if has ('autocmd')          " Remain compatible with earlier versions
 augroup Reload_Vimrc       " Group name.  Always use a unique name!
    autocmd!                " Clear any preexisting autocommands from this group
    autocmd! BufWritePost $MYVIMRC source % | echom "Reloaded " . $MYVIMRC | redraw
    autocmd! BufWritePost $MYGVIMRC if has('gui_running') | so % | echom "Reloaded " .
$MYGVIMRC | endif | redraw
  augroup END
endif " has autocmd
```

Caractéristiques:

- `echom` indique à l'utilisateur ce qui s'est passé (et se connecte également aux `:messages`).
- `$MYVIMRC` et `$MYGVIMRC` gèrent les noms spécifiques à la plate-forme pour les fichiers de

configuration,
- et ne correspondent qu'aux fichiers de configuration réels (en ignorant les copies dans d'autres répertoires, ou un `fugitive://` diff)
- `has()` empêchera une erreur si vous utilisez des versions incompatibles, telles que `vim-tiny` .
- `autocmd!` évite la création de plusieurs autocommandes identiques si ce fichier est à nouveau généré. (Il efface toutes les commandes du groupe nommé, le nom du groupe est donc important.)

Actualiser les vues vimdiff si un fichier est enregistré dans une autre fenêtre

```
:autocmd BufWritePost * if &diff | diffupdate | endif
```

Lire Commandes Automatiques en ligne: https://riptutorial.com/fr/vim/topic/4887/commandes-automatiques

Chapitre 8: Commandes en mode normal

Syntaxe

- : [range] sor [t] [!] [b] [f] [i] [n] [o] [r] [u] [x] [/ {pattern} /]
- Remarque: les options [n] [f] [x] [o] [b] sont mutuellement exclusives.

Remarques

Voir le tri dans le manuel vim pour l'explication canonique

Examples

Tri du texte

Tri normal

Mettez en surbrillance le texte à trier et le type:

```
:sort
```

Si vous ne mettez pas de texte en surbrillance ou spécifiez une plage, le tampon entier est trié.

Tri inverse

```
:sort!
```

Tri insensible à la casse

```
:sort i
```

Tri numérique

Trier par le premier numéro à apparaître sur chaque ligne:

```
:sort n
```

Supprimer les doublons après le tri

```
:sort u
```

(u signifie unique)

Combinaison d'options

Pour obtenir un tri inverse à la casse avec suppression des doublons:

```
:sort! iu
```

Lire Commandes en mode normal en ligne: https://riptutorial.com/fr/vim/topic/6005/commandes-en-mode-normal

Chapitre 9: Commandes en mode normal (édition)

Examples

Introduction - Note rapide sur le mode normal

En mode normal, les commandes peuvent être saisies par des combinaisons de touches directes (en tapant `u` pour annuler le dernier changement, par exemple). Ces commandes sont souvent équivalents en mode « ex », accessible en tapant deux `points:` qui vous dépose dans une mémoire tampon de ligne individuelle au bas de la fenêtre Vim.

En mode 'ex', après avoir tapé les deux points, vous tapez un nom de commande ou son abréviation suivi de la `touche Entrée` pour exécuter la commande. `Alors,: undo Entrez` comme `u` accomplit tapant directement la même chose en mode normal.

Vous pouvez voir que les commandes directes seront souvent plus rapides (une fois apprises) que les commandes «ex» pour une édition simple, mais pour être complet, si possible dans la documentation qui suit, si les deux sont disponibles, les deux seront affichés.

La plupart de ces commandes peuvent également être précédées d'un *décompte* en préfixant ou en intercalant un nombre. `Si vous` tapez `3dd` en mode normal, par exemple, vous supprimez trois lignes (en commençant par la position actuelle du curseur).

Annulation et rétablissement de base

annuler

Commander	:	La description
`tu`	`u` , `undo`	Annuler le changement le plus récent
`5u`		Annuler les *cinq* changements les plus récents (utilisez n'importe quel nombre)

Sachez que dans Vim, le «changement le plus récent» varie selon le mode dans lequel vous vous trouvez. Si vous entrez en mode d'insertion (`i`) et que vous tapez un paragraphe entier avant de revenir au mode normal (`Esc`), *ce paragraphe entier* est considéré comme le changement le plus récent.

Refaire

Commander	:	La description
`Ctrl-R`	`red` , `redo`	Refaire le dernier changement annulé
`2Ctrl-R`		Refaire les *deux* dernières modifications annulées (utilisez n'importe quel nombre)

Il existe un autre moyen d'annuler et de rétablir des modifications dans Vim qui sont traitées un peu différemment. Lorsque vous annulez une modification avec `u` , vous effectuez une sauvegarde des noeuds sur une "arborescence" de vos modifications et en appuyant sur `Ctrl-R` , vous redescendez ces noeuds dans l'ordre. (L'arbre d'annulation est un sujet distinct et trop complexe à aborder ici.)

Vous pouvez *également* utiliser `U` (c'est-à-dire majuscule) pour supprimer toutes les dernières modifications sur une seule ligne (la ligne sur laquelle vos dernières modifications ont été apportées). Cela *ne* traverse *pas* les nœuds de l'arbre de la même manière que `u` . Utiliser `U` compte en fait comme un changement lui-même - un autre nœud en *avant* sur l'arborescence - de sorte que si vous appuyez une seconde fois sur `U` immédiatement après le premier, il agira comme une commande Redo.

Chacun a ses usages, mais `u` / `:undo` devrait couvrir la plupart des cas simples.

Répéter le dernier changement

La commande Répéter, exécutée avec la touche point ou point (`.`), Est plus utile qu'elle n'apparaît d'abord. Une fois appris, vous vous retrouverez à l'utiliser souvent.

Commander	:	La description
`.`		Répéter le dernier changement
`dix.`		Répétez le dernier changement 10 fois

Donc, pour un exemple très simple, si vous apportez une modification à la ligne 1 en tapant `i I` Esc, le résultat est le suivant:

```
1 I made a mistake
2  made a mistake
3  made a mistake
```

Votre curseur sera à la position 1 de la ligne 1 et tout ce que vous devez faire pour corriger les deux lignes suivantes est d'appuyer sur `j.` deux fois - c'est-à-dire `j` pour descendre une ligne et `.` répéter le dernier changement, qui était l'ajout du `I` Pas besoin de revenir en mode Insertion deux fois pour corriger ces lignes.

Il devient beaucoup plus puissant lorsqu'il est utilisé pour répéter des macros .

Copier, couper et coller

Dans Vim, ces opérations sont traitées différemment de ce que vous pourriez être habitué dans presque tous les autres éditeurs ou traitements de texte modernes (`Ctrl-C` , `Ctrl-X` , `Ctrl-V`). Pour comprendre, vous devez en savoir un peu plus sur les registres et les mouvements.

Remarque: cette section ne couvre pas la copie en mode visuel et le découpage ou le découpage de gamme car ceux-ci dépassent le cadre du mode normal et de l'édition de base.

Registres

Vim utilise le concept de *registres* pour gérer le texte en mouvement dans le programme lui-même. Windows dispose d'un seul presse-papiers à cet effet, analogue à un seul registre dans Vim. Lors de la copie, de la découpe et du collage dans Vim, il existe des moyens d'utiliser un workflow d'édition aussi simple (sans penser aux registres), mais il existe également des possibilités beaucoup plus complexes.

Un registre est ciblé pour l'entrée / sortie d'une commande en préfixant la commande avec `"` et un nom en minuscule.

Les motions

Un mouvement dans Vim est une commande qui déplace la position du curseur ailleurs. Lors de la copie, de la découpe et du collage en mode normal, les possibilités de sélection de texte pour le mouvement ne sont limitées que par votre connaissance des mouvements. Quelques-uns seront illustrés ci-dessous.

Copier et couper

Les commandes de base copier et couper sont construites sur `y` ('yank', pour copier) et `d` ('delete', pour couper). Vous verrez les similitudes dans le tableau suivant.

Commander	:	La description
`y {mouvement}`		Copier («yank») le texte indiqué par le mouvement dans le registre par défaut
`yy`		Copiez la ligne courante dans le registre par défaut, *linewise*
`Y`		Copiez la ligne actuelle dans le registre par défaut (synonyme de `yy`)
`"ayiw`		Copie le mot sur lequel se trouve le curseur pour enregistrer 'a'
`20 "byy`		Copier vingt lignes, en commençant par le curseur, dans le registre 'b'
`d {motion}`		Couper («supprimer») le texte indiqué par le mouvement dans le registre par défaut

Commander	:	La description
`dd`		Couper la ligne courante dans le registre par défaut, *linewise*
`ré`		Couper du curseur à la fin de la ligne dans le registre par défaut (PAS un synonyme de `dd`)
`"adiw`		Couper le mot sur lequel se trouve le curseur pour enregistrer 'a'
`20 "bdd`		Couper vingt lignes, en commençant par le curseur, dans le registre 'b'

Remarque: quand quelque chose est copié ou coupé *linewise,* le comportement de la pâte ci - dessous se placer du texte avant ou après la *ligne* en cours (plutôt que le curseur). Les exemples suivent pour clarifier.

Coller

Il y a plusieurs façons de coller dans Vim, selon ce que vous essayez d'accomplir.

Commander	:	La description
`p`		Collez ce qui est dans le registre par défaut *après* le curseur
`P`		Collez ce qui est dans le registre par défaut *avant* le curseur
`"ap`		Collez le contenu du registre 'a' après le curseur
`"cp`		Collez le contenu du registre 'c' avant le curseur

Alors, comment effectuer une coupe et un collage vraiment simples?

Si j'ai le texte suivant:

```
1 This line should be second
2 This line should be first
```

Je peux faire le simple copier-coller en plaçant mon curseur quelque part sur la ligne 1 et en tapant `ddp` . Voici les résultats:

```
1 This line should be first
2 This line should be second
```

Qu'est-il arrivé? `dd` 'Coupe la première ligne (ligne par ligne) dans le registre par défaut - qui ne contiendra qu'une seule chose à la fois, comme le presse-papier de Windows - et `p` colle la ligne après celle qui vient d'être modifiée à cause de la commande `dd` .

Voici un exemple pas aussi simple. Je dois déplacer quelques mots. (Ceci est artificiel et inutile, mais vous pouvez appliquer ce principe à des morceaux de code plus importants.)

```
1 These words order out are of
```

Je peux répéter `w` pour arriver au 'o' au début de 'order' (ou `b` si je l'ai juste tapé et réalisé mon erreur).

Alors `"adaw` mettre" ordre "dans le registre" a ".

Alors `w` pour arriver au 'a' dans 'sont'.

Après cela, je tapez `"bdaw` pour mettre 'sont' dans le registre 'b'.

```
1 These words out of
```

Pour être clair, maintenant "order" est dans le registre "a" et "are" dans le registre "b", comme deux presse-papiers séparés.

Pour arranger les mots correctement, je tape `b` pour accéder au 'o' dans 'out', puis `«bP` pour mettre 'are' du registre 'b' devant 'out':

```
1 These words are out of
```

Maintenant, je tape `A` pour arriver à la fin de la ligne, suivi de `Space Esc` (en supposant qu'il n'y ait pas d'espace après «of») et `«ap` pour mettre« order »à l'endroit où il appartient.

```
1 These words are out of order
```

Achèvement

L'achèvement peut être utilisé pour faire correspondre les mots utilisés dans un document. Lors de la saisie d'un mot, `Ctrl p` ou `Ctrl n` correspondra aux mots précédents ou suivants du document.

Cela peut même être combiné avec le mode `Ctrl-X` pour compléter des lignes entières. Par exemple, tapez quelque chose comme:

```
This is an example sentence.
```

puis allez à la ligne suivante et commencez à taper la même phrase:

```
Thi
```

puis appuyez sur `Ctrl p` pour obtenir:

```
This
```

Maintenant, toujours en mode insertion, appuyez sur `Ctrl x Ctrl p` et le mot suivant sera terminé, ce qui donnera:

```
This is
```

Continuez à appuyer sur `Ctrl x Ctrl p` jusqu'à ce que toute la ligne soit terminée.

Si vous savez que vous souhaitez compléter un type de ligne complet, procédez comme suit:

```
This is an example sentence.
```

puis sur le prochain type de ligne:

```
Thi
```

et appuyez sur `x Ctrl l` pour terminer la ligne.

Si la finalisation est un nom de fichier, `Ctrl x Ctrl f` peut être utilisé pour compléter ce répertoire. Type:

```
~/Deskt
```

puis appuyez sur `Ctrl x Ctrl f` et:

```
~/Desktop
```

sera complété (si à cet endroit). `Ctrl x Ctrl f` peut alors être utilisé à plusieurs reprises pour répertorier les fichiers sur le bureau.

Lire Commandes en mode normal (édition) en ligne:
https://riptutorial.com/fr/vim/topic/5250/commandes-en-mode-normal--edition-

Chapitre 10: Comment compiler Vim

Examples

Compiler sur Ubuntu

Pour construire vim depuis la source sur Ubuntu:

1. Obtenez une copie du code source en téléchargeant depuis le dépôt officiel de Vim sur GitHub .
2. Obtenez les dépendances en exécutant `$ sudo apt-get build-dep vim-gnome` ou similaire.
3. Accédez au répertoire du code source de Vim: `cd vim/src`
4. Exécutez `$ ./configure` . Vous pouvez personnaliser la génération (et activer les intégrations de langage Perl, Python, etc.) en passant les options de configuration. Voir `src/INSTALL` pour un aperçu.
5. Exécutez `$ make` .
6. Terminez l'installation en exécutant `$ sudo make install` . Comme votre Vim auto-compilé n'est pas géré par le gestionnaire de paquets, il sera placé dans `/usr/local/bin/vim` , au lieu de `/usr/bin/vim` . Donc, pour l'exécuter, vous devez soit spécifier le chemin complet, soit vous assurer que `/usr/local/bin` est avant `/usr/bin` dans votre `PATH` (c'est généralement le cas).
7. (Facultatif) Supprimez la version de Vim fournie par la distribution si vous l'aviez déjà installée: `$ sudo apt-get remove vim vim-runtime gvim` .

Pour vérifier l'installation, vous pouvez exécuter `$ which vim` qui doit renvoyer quelque chose comme `/usr/local/bin/vim` si l'installation a réussi.

Lire Comment compiler Vim en ligne: https://riptutorial.com/fr/vim/topic/3737/comment-compiler-vim

Chapitre 11: Configuration de Vim

Examples

Le fichier vimrc

Le fichier `.vimrc` (prononcé Vim- `.vimrc`) est un fichier de configuration Vim. Il contient des commandes qui seront exécutées par Vim à chaque démarrage.

Par défaut, le fichier est vide ou inexistant; vous pouvez l'utiliser pour personnaliser votre environnement Vim.

Pour savoir où Vim s'attend à ce que le fichier vimrc soit stocké, ouvrez Vim et exécutez:

```
:echo $MYVIMRC
```

Unix: sur un système Unix tel que Mac ou Linux, votre vimrc s'appellera `.vimrc` et se `.vimrc` généralement dans votre répertoire personnel (`$HOME/.vimrc`).

Windows: sous Windows, il s'appellera `_vimrc` et se `_vimrc` dans votre répertoire personnel (`%HOMEPATH%/_vimrc`).

Au démarrage, Vim recherchera un fichier vimrc à plusieurs endroits. Le premier qui existe est utilisé, les autres sont ignorés. Pour une référence complète, consultez l'article de documentation `:h $MYVIMRC` .

Quelles options puis-je utiliser?

Si vous ne savez pas quelles options utiliser, vous pouvez être intéressé par la commande `:options` .

Cela ouvrira une division avec toutes les options Vim répertoriées et leur valeur actuelle affichée. Il y a 26 sections pour afficher toutes les options que vous pouvez essayer.

par exemple

```
4 displaying text

scroll    number of lines to scroll for CTRL-U and CTRL-D
    (local to window)
     set scr=20
scrolloff    number of screen lines to show around the cursor
     set so=5
wrap    long lines wrap
     set nowrap    wrap

...
```

Sur une ligne de valeur (par exemple, `set nowrap`), vous pouvez appuyer sur `CR` pour changer la valeur (si c'est une valeur binaire). Sur une ligne d'option (par exemple, `wrap long line wrap`), vous pouvez appuyer sur `CR` pour accéder à la documentation de cette option.

Fichiers et répertoires

Quoi que vous fassiez pour personnaliser Vim, il ne devrait JAMAIS arriver en dehors de `$HOME` :

- sur Linux, BSD et Cygwin, `$HOME` est généralement `/home/username/` ,
- sur Mac OS X, `$HOME` est `/Users/username/` ,
- sous Windows, `$HOME` est généralement `C:\Users\username\` .

L'emplacement canonique de votre `vimrc` et de votre répertoire `vim` est à la racine de ce répertoire `$HOME` :

- sur les systèmes de type Unix

```
$HOME/.vimrc       <-- the file
$HOME/.vim/        <-- the directory
```

- sur Windows

```
$HOME\_vimrc       <-- the file
$HOME\vimfiles\    <-- the directory
```

La mise en page ci-dessus est garantie pour fonctionner, maintenant et à l'avenir.

Vim 7.4 a permis de garder votre belle `vimrc` **dans** votre répertoire `vim` . C'est vraiment une bonne idée, ne serait-ce que parce que cela facilite le déplacement de votre configuration.

Si vous utilisez exclusivement 7.4, ce qui suit suffira:

- sur des systèmes de type Unix

```
$HOME/.vim/vimrc
```

- sur Windows

```
$HOME\vimfiles\vimrc
```

Si vous voulez les avantages d'un `vim/` autonome mais utilisez à la fois la version 7.4 et une version antérieure, ou seulement une version antérieure, la solution la plus simple et la plus évolutive consiste à placer cette ligne *et uniquement cette ligne:*

```
runtime vimrc
```

dans ce fichier:

- sur les systèmes de type Unix

```
$HOME/.vimrc
```

- sur Windows

```
$HOME\_vimrc
```

et faites votre configuration dans `$HOME/.vim/vimrc` ou `$HOME\vimfiles\vimrc` .

Les options

Il existe trois types d’options:

- options booléennes,
- options de chaîne,
- options de nombre.

Pour vérifier la valeur d'une option,

- utiliser `:set option?` vérifier la valeur d'une option,
- utiliser `:verbose set option?` pour voir aussi où il a été mis en dernier.

Définition des options booléennes

```
set booloption      " Set booloption.
set nobooloption    " Unset booloption.

set booloption!     " Toggle booloption.

set booloption&     " Reset booloption to its default value.
```

Définition des options de chaîne

```
set stroption=baz   " baz

set stroption+=buzz " baz,buzz
set stroption^=fizz " fizz,baz,buzz
set stroption-=baz  " fizz,buzz

set stroption=      " Unset stroption.

set stroption&      " Reset stroption to its default value.
```

Définition des options numériques

```
set numoption=1     " 1

set numoption+=2    " 1 + 2 == 3
set numoption-=1    " 3 - 1 == 2
set numoption^=8    " 2 * 8 == 16
```

Utiliser une expression comme valeur

- en utilisant la concaténation:

```
execute "set stroption=" . my_variable
```

- en utilisant `:let` :

```
let &stroption = my_variable
```

Voir `:help :set` et `:help :let` .

Cartographie

- Ne mettez pas de commentaires après les mappages, cela va casser des choses.
- Utilisez `:map <F6>` pour voir ce qui est mappé sur `<F6>` et dans quel mode.
- Utilisez `:verbose map <F6>` pour voir également où il a été mappé pour la dernière fois.
- `:map` et `:map!` sont trop génériques. Use `:n[nore]map` pour les mappages en mode normal,: `:i[nore]map` pour le mode insert `:x[nore]map` pour le mode visuel, etc.

Mappages récursifs

Utilisez *les* mappages *récursifs* **uniquement** si vous avez l'intention d'utiliser d'autres mappages dans vos mappages:

```
nnoremap b     B
nmap     <key> db
```

Dans cet exemple, `b` est conçu pour fonctionner comme `B` en mode normal. Puisque nous utilisons `b` dans un mappage *récursif* , appuyer sur `<key>` fonctionnera comme `dB` , pas comme `db` .

Mappages non récursifs

Utilisez *des* mappages *non récursifs* **uniquement** si vous avez l'intention d'utiliser des commandes par défaut dans vos mappages, ce qui est presque toujours ce que vous voulez:

```
nnoremap <key> db
```

Dans cet exemple, nous utilisons `b` dans un mappage *non récursives* de manière à toujours pressant la touche fonctionnera comme `db` si nous Réaffectés, `b` ou non.

Exécuter une commande à partir d'un mappage

```
nnoremap <key> :MyCommand<CR>
```

Exécution de plusieurs commandes à partir d'un mappage

```
nnoremap <key> :MyCommand <bar> MyOtherCommand <bar> SomeCommand<CR>
```

Appeler une fonction à partir d'un mappage

```
nnoremap <key> :call SomeFunction()<CR>
```

Mappage d'un mappage `<Plug>`

```
map <key> <Plug>name_of_mapping
```

Voir `:help map-commands` `:help key-notation` et `:help <plug>` .

voir les mappages de touches dans Vim pour en savoir plus

Les variables

Comme la plupart des langages de script, vimscript a des variables.

On peut définir une variable avec la commande: `:let` :

```
let variable = value
```

et le supprimer avec `:unlet` :

```
unlet variable
```

Dans Vim, les variables peuvent être définies en ajoutant une lettre et un deux-points à leur nom. Les auteurs de plug-ins utilisent cette fonctionnalité pour simuler les options:

```
let g:plugin_variable = 1
```

Voir `:help internal-variables` .

Commandes

- N'oubliez pas le bang pour permettre à Vim d'écraser cette commande la prochaine fois que vous rechargerez votre vimrc.
- Les commandes personnalisées doivent commencer par un caractère majuscule.

Exemples

```
command! MyCommand call SomeFunction()
```

```
command! MyOtherCommand command | Command | command
```

- Voir `:help user-commands` .

Les fonctions

- N'oubliez pas le bang pour permettre à Vim d'écraser cette fonction la prochaine fois que vous rechargerez le script où la fonction est définie.
- Les fonctions personnalisées doivent commencer soit par un caractère majuscule (fonctions globales), soit par `s:` (fonctions locales de script), soit être précédées du nom associé au plug-in de chargement automatique où elles sont définies (par exemple dans `{&rtp}/autoload/foo/bar.vim` nous pourrions définir `foo#bar#functionname()`).
- Pour pouvoir utiliser les paramètres de la fonction, utilisez `a:parameter_name` . Les fonctions variadiques peuvent être définies avec les points de suspension `...` , pour accéder aux paramètres, utilisez `a:000` (liste de tous les paramètres), ou `a:0` (nombre de paramètres égal à `len(a:000)`), `a:1` paramètres, et ainsi de suite.
- Les fonctions peuvent être appelées comme `:call MyFunction(param1, param2)`
- Chaque ligne d'une fonction commence implicitement par a `:` toutes les commandes sont donc des deux-points
- Pour éviter que la fonction continue son exécution en cas d'erreur, il est préférable d'annoter la signature de la fonction avec `abort`

Exemple

```
function! MyFunction(foo, bar, ... ) abort
    return a:foo . a:bar . (a:0 > 0 ? a:1 : '')
endfunction
```

Fonctions de script

Si vous prévoyez d'utiliser uniquement votre fonction dans le fichier où elle est définie (soit parce que vous avez cassé une fonction plus grande dans des parties plus petites, soit parce que vous l'utiliserez dans une commande, un mappage, ...), vous pouvez préfixer avec `s:` :, en évitant de joncher votre espace de noms global avec des fonctions internes inutiles:

```
function! s:my_private_function() " note we don't need to capitalize the first letter this
time
    echo "Hi!"
endfunction
```

Utilisation de s: fonctions à partir de mappages

Si votre fonction de script locale doit être utilisée dans un mappage, vous devez la référencer en utilisant le préfixe `<SID>` spécial:

```
nnoremap <your-mapping-key> :call <SID>my_private_function()<CR>
```

Voir `:help user-functions` .

Notez cependant que depuis Vim 7, il est recommandé de définir des abréviations, des commandes et des menus dans les plug-ins (ft) et de définir des fonctions dans les plug-ins de chargement automatique, à l'exception des fonctions que les plug-ins doivent utiliser lorsqu'ils sont chargés. Cela signifie que, de nos jours, le besoin d'appeler les fonctions locales de scripts à partir de mappages n'est pas aussi pertinent que par le passé.

Groupes de commandes automatiques

- Les groupes de commandes automatiques sont bons pour l'organisation, mais ils peuvent également être utiles pour le débogage. Considérez-les comme de petits espaces de noms que vous pouvez activer / désactiver à volonté.

Exemple

```
augroup MyGroup
    " Clear the autocmds of the current group to prevent them from piling
    " up each time you reload your vimrc.
    autocmd!

    " These autocmds are fired after the filetype of a buffer is defined to
    " 'foo'. Don't forget to use 'setlocal' (for options) and '<buffer>'
    " (for mappings) to prevent your settings to leak in other buffers with
    " a different filetype.
    autocmd FileType foo setlocal bar=baz
    autocmd FileType foo nnoremap <buffer> <key> :command<CR>

    " This autocmd calls 'MyFunction()' everytime Vim tries to create/edit
    " a buffer tied to a file in /'path/to/project/**/'.
    autocmd BufNew,BufEnter /path/to/project/**/* call MyFunction()
augroup END
```

Voir `:help autocommand` .

Conditionnels

```
if v:version >= 704
    " Do something if Vim is the right version.
endif

if has('patch666')
    " Do something if Vim has the right patch-level.
endif

if has('feature')
    " Do something if Vim is built with 'feature'.
endif
```

Voir `:help has-patch` et `:help feature-list` .

Options de réglage

Généralement, vous utiliseriez `:set` pour définir les options à votre convenance dans votre `.vimrc` . De nombreuses options peuvent être modifiées.

Par exemple, pour utiliser des espaces pour l'indentation:

```
:set expandtab
:set shiftwidth=4
:set softtabstop=4
```

Mise en évidence de la syntaxe

Mise en surbrillance de la syntaxe, lorsque le terminal a des couleurs

```
if &t_Co > 2 || has("gui_running")
    syntax on
end
```

Afficher les espaces blancs et les onglets. L'affichage des onglets peut être particulièrement utile lors de la recherche d'erreurs dans les Makefiles.

```
set list listchars=tab:\|_,trail:.
highlight SpecialKey ctermfg=DarkGray
```

Schémas de couleurs

Vim est livré avec plusieurs combinaisons de couleurs préinstallées. Sous Linux, les schémas de couleurs fournis avec Vim sont stockés dans `/usr/share/vim/vim74/colors/` (où 74 correspond à votre numéro de version, sans points); MacVim les stocke dans `/Applications/MacVim.app/Contents/Resources/vim/runtime/colors` .

Modification des schémas de couleurs

La commande `colorscheme` change le `colorscheme` actuel.

Par exemple, pour définir le schéma de couleurs sur "robokai":

```
:colorscheme robokai
```

Le schéma de couleurs par défaut est nommé `default` manière créative, donc, pour y revenir, utilisez

```
:colorscheme default
```

Pour afficher tous les schémas de couleurs actuellement installés, tapez `:colorscheme` suivi d'un `espace` puis de l' `onglet` ou de la `touche ctrl d` .

Installation de schémas de couleurs

Les schémas de couleurs installés par l'utilisateur peuvent être placés dans `~/.vim/colors/` . Une fois qu'un jeu de couleurs est ajouté à ce répertoire, il apparaîtra comme une option pour la commande `colorscheme` .

Pour trouver de nouveaux schémas de couleurs, il existe des sites tels que vimcolors qui contiennent divers schémas de couleurs. Il existe également des outils comme vim.ink et Vivify pour vous aider à créer vos propres jeux de couleurs, ou vous pouvez les créer à la main.

Basculer la ligne en énumérant

Pour activer - tapez:

`:set number` ou `:set nu` .

Pour désactiver - tapez:

`:set nonumber` ou `:set nonu` .

Pour activer l'énumération *relative* à l'emplacement du curseur - tapez:

`:set relativenumber` .

Pour désactiver l'énumération *relative* à l'emplacement du curseur - tapez:

`:set norelativenumber` .

Remarque: Pour modifier si la ligne en cours affiche le numéro de ligne actuel ou `0` , utilisez les commandes `:set number` et `:set nonumber` lorsque l'attribut `relativenumber` est actif.

Plugins

Les plugins Vim sont des addons pouvant être utilisés pour modifier ou améliorer les fonctionnalités de vim.

Il y a une bonne liste de plugins à vimawesome

Lire Configuration de Vim en ligne: https://riptutorial.com/fr/vim/topic/2235/configuration-de-vim

Chapitre 12: Configurations utiles pouvant être placées dans .vimrc

Syntaxe

- définir la souris = a
- set wrap
- nmap j gj
- nmap k gk

Examples

Déplacer les lignes affichées vers le haut / bas

En général, `J` et `K` déplacent les lignes de fichiers vers le haut et vers le bas. Mais quand vous avez l' emballage, vous pouvez les vouloir monter et descendre les lignes **affichées** au lieu.

```
set wrap " if you haven't already set it
nmap j gj
nmap k gk
```

Activer l'interaction de la souris

```
set mouse=a
```

Cela permettra l'interaction de la souris dans l'éditeur `vim` . La souris peut

- changer la position actuelle du curseur
- sélectionnez le texte

Configurez le registre par défaut à utiliser comme presse-papiers système

```
set clipboard=unnamed
```

Cela permet de copier / coller entre Vim et le presse-papier du système sans spécifier de registre spécial.

`yy` tire la ligne actuelle dans le presse-papier du système

`p` colle le contenu du presse-papiers du système dans Vim

Cela ne fonctionne que si votre installation Vim prend en charge le Presse-papiers. Exécutez la commande suivante dans le terminal pour vérifier si l'option du Presse-papiers est disponible: `vim --version | grep clipboard`

Lire Configurations utiles pouvant être placées dans .vimrc en ligne:
https://riptutorial.com/fr/vim/topic/6560/configurations-utiles-pouvant-etre-placees-dans--vimrc

Chapitre 13: Conversion de fichiers texte de DOS en UNIX avec vi

Remarques

Le caractère `^M` représente un retour à la ligne dans Vim (`<cm>` ou juste `<CR>`). Vim affiche ce caractère lorsque au moins en ligne dans le fichier utilise des fins de ligne `LF` . En d'autres termes, lorsque Vim considère qu'un fichier a un `fileformat=unix` fichier `fileformat=unix` mais que certaines lignes ont des retours de chariot (`CR`), les retours de chariot sont affichés sous la forme `^M`

Un fichier ayant une seule ligne avec une fin de ligne `LF` et plusieurs lignes avec des fins de ligne `CRLF` est le plus souvent créé en éditant à tort un fichier créé sur un système basé sur MSDOS. Par exemple, en créant un fichier sous un système d'exploitation MSDOS, en le copiant sur un système UNIX, puis en ajoutant une méthode de hachage (par exemple `#!/bin/sh`) à l'aide des outils du système d'exploitation UNIX.

Examples

Conversion d'un fichier texte DOS en fichier texte UNIX

Très souvent, vous avez un fichier qui a été édité sous DOS ou Windows et que vous affichez sous UNIX. Cela peut ressembler à ce qui suit lorsque vous affichez le fichier avec vi.

```
First line of file^M
Next Line^M
And another^M
```

Si vous souhaitez supprimer le ^ M, il se peut que vous supprimiez chaque ^ M à la main. Alternativement, en vi après avoir `appuyé` sur `Échap`, vous pouvez entrer les informations suivantes à l'invite du mode de commande:

```
:1,$s/^M//g
```

Où ^ M est entré avec `Ctrl` et `v` ensemble, puis `Ctrl` et `m` ensemble.

Ceci exécute la commande de la première ligne '1' à la dernière ligne '$', la commande consiste à substituer 's' le '^ M' pour rien '' et à cela globalement 'g'.

Utiliser le format de fichier de VIm

Lorsque Vim ouvre un fichier avec les extrémités de ligne `<CR><NL>` (commun sur les systèmes d'exploitation basés sur MSDOS, également appelé `CRLF`), il définira `fileformat` à `dos` , vous pouvez vérifier avec:

```
:set fileformat?
  fileformat=dos
```

Ou juste

```
:set ff?
  fileformat=dos
```

Pour le convertir en terminaisons `<NL>` (commun sur la plupart des systèmes d'exploitation UNIX, également appelés `LF`), vous pouvez modifier le paramètre `fileformat` et Vim modifiera le tampon.

```
:set ff=unix
```

Lire Conversion de fichiers texte de DOS en UNIX avec vi en ligne: https://riptutorial.com/fr/vim/topic/3827/conversion-de-fichiers-texte-de-dos-en-unix-avec-vi

Chapitre 14: Correcteur orthographique

Examples

Vérification orthographique

Pour activer le vérificateur orthographique vim, exécutez `:set spell` . Pour le désactiver, `:set nospell` . Si vous voulez toujours que le correcteur orthographique soit activé, ajoutez un `set spell` à votre vimrc. Vous ne pouvez activer l'orthographe que pour certains types de fichiers à l'aide d'une commande automatique.

Une fois le correcteur orthographique activé, les mots mal orthographiés sont mis en surbrillance. Tapez `]s` pour passer au mot mal orthographié suivant et `[s` pour passer au mot précédent. Pour voir une liste d'orthographes corrigées, placez le curseur sur un mot mal orthographié et tapez `z=` . Vous pouvez taper le numéro du mot que vous souhaitez remplacer par le mot mal orthographié et appuyer sur `<enter>` pour le remplacer, ou vous pouvez simplement appuyer sur Entrée pour laisser le mot inchangé.

Avec le curseur sur un mot mal orthographié, vous pouvez également taper `<number>z=` pour passer à la `<number>` correction `<number>` sans voir la liste. Typiquement, vous utiliserez `1z=` si vous pensez que la première estimation de `vim` est susceptible d'être le mot correct.

Set Word List

Pour définir la liste de mots que `vim` utilisera pour la vérification orthographique, définissez l'option de `spelllang` . Par exemple

```
:set spelllang=en        # set to English, usually this is the default
:set spelllang=en_us     # set to U.S. English
:set spelllang=en_uk     # set to U.K. English spellings
:set spelllang=es        # set to Spanish
```

Si vous souhaitez définir le `spelllang` et activer la vérification orthographique en une seule commande, vous pouvez le faire:

```
:setlocal spell spelllang=en
```

Lire Correcteur orthographique en ligne: https://riptutorial.com/fr/vim/topic/3653/correcteur-orthographique

Chapitre 15: Défilement

Examples

Défilement vers le bas

Commander	La description
`Ctrl + E`	Faites défiler une ligne vers le bas.
`Ctrl + D`	Faites défiler la moitié d'un écran vers le bas (configurable à l'aide de l'option de `scroll`).
`Ctrl + F`	Faites défiler un écran entier vers le bas.
`z +`	Dessinez la première ligne sous la fenêtre en haut de la fenêtre.

Faire défiler vers le haut

Commander	La description
`Ctrl + Y`	Faites défiler une ligne
`Ctrl + U`	Faites défiler une demi-page vers le haut (configurable à l'aide de l'option de `scroll`).
`Ctrl + B`	Faites défiler un écran complet vers le haut.
`z ^`	Dessinez la première ligne au-dessus de la fenêtre au bas de la fenêtre.

Défilement par rapport à la position du curseur

Commander	La description
`z`	Redessinez la ligne en haut de la fenêtre et placez le curseur sur le premier caractère non vide de la ligne.
`zt`	Comme `z` mais laissez le curseur dans la même colonne.
`z.`	Redessinez la ligne actuelle au centre de la fenêtre et placez le curseur sur le premier caractère non vide de la ligne.
`zz`	Comme `z.` mais laissez le curseur dans la même colonne.
`z-`	Redessinez la ligne en bas de la fenêtre et placez le curseur sur le premier

Commander	La description
	caractère non vide de la ligne.
`zb`	Comme `z-` mais laissez le curseur dans la même colonne.

Lire Défilement en ligne: https://riptutorial.com/fr/vim/topic/3000/defilement

Chapitre 16: Demandez à créer des répertoires inexistants en enregistrant un nouveau fichier

Introduction

Si vous éditez un nouveau fichier: `vim these/directories/dont/exist/newfile` , vous ne pourrez pas enregistrer le fichier car le répertoire dans `vim` tente de sauvegarder n'existe pas.

Examples

Invitez à créer des répertoires avec: w ou créez-les temporairement avec: w!

Ce code vous invitera à créer le répertoire avec `:w` , ou simplement le faire avec `:w!` :

```
augroup vimrc-auto-mkdir
  autocmd!
  autocmd BufWritePre * call s:auto_mkdir(expand('<afile>:p:h'), v:cmdbang)
  function! s:auto_mkdir(dir, force)
    if !isdirectory(a:dir)
          \   && (a:force
          \       || input("'" . a:dir . "' does not exist. Create? [y/N]") =~? '^y\%[es]$')
      call mkdir(iconv(a:dir, &encoding, &termencoding), 'p')
    endif
  endfunction
augroup END
```

Lire Demandez à créer des répertoires inexistants en enregistrant un nouveau fichier en ligne: https://riptutorial.com/fr/vim/topic/9470/demandez-a-creer-des-repertoires-inexistants-en-enregistrant-un-nouveau-fichier

Chapitre 17: Différences entre Neovim et Vim

Examples

Fichiers de configuration

Dans Vim, votre fichier de configuration est `~/.vimrc` , avec d'autres fichiers de configuration dans `~/.vim` .

Dans Neovim, les fichiers de configuration se trouvent dans `~/.config/nvim` . Il n'y a pas non `~/.nvimrc` fichier `~/.nvimrc` . Au lieu de cela, utilisez `~/.config/nvim/init.vim` .

Vous pouvez importer votre configuration Vim directement dans Neovim en utilisant cette commande Unix:

```
ln -s ~/.vimrc ~/.config/nvim/init.vim
```

Lire Différences entre Neovim et Vim en ligne: https://riptutorial.com/fr/vim/topic/7848/differences-entre-neovim-et-vim

Chapitre 18: Échancrure

Examples

Indenter un fichier entier en utilisant le moteur d'indentention intégré

En mode commande (Esc), entrez `:gg=G` pour utiliser le moteur d'indentation intégré de Vim.

Partie commande	La description
gg	début du fichier
=	`equalprg` (quand `equalprg` est vide)
g	fin de fichier

Vous pouvez définir `equalprg` dans votre fichier .vimrc pour utiliser un outil de mise en forme automatique plus sophistiqué.

Par exemple, pour utiliser `clang-format` for C / C ++, placez la ligne suivante dans votre fichier `.vimrc` :

```
autocmd FileType c,cpp setlocal equalprg=clang-format
```

Pour les autres types de fichiers, remplacez `c,cpp` par le type de fichier que vous souhaitez formater et `clang-format` par votre outil de formatage préféré pour ce type de fichier.

Par exemple:

```
" Use xmllint for indenting XML files. Commented out.
"autocmd FileType xml setlocal equalprg=xmllint\ --format\ --recover\ -\ 2>/dev/null
" Tidy gives more formatting options than xmllint
autocmd FileType xml setlocal equalprg=tidy\ --indent-spaces\ 4\ --indent-attributes\ yes\ --
sort-attributes\ alpha\ --drop-empty-paras\ no\ --vertical-space\ yes\ --wrap\ 80\ -i\ -xml\
2>/dev/null
```

Lignes en retrait ou en retrait

Pour mettre en retrait notre outdent la ligne courante en mode normal appuyez sur la plus grande que `>` touche ou moins `<` deux fois en conséquence. Pour faire la même chose sur plusieurs lignes, ajoutez simplement un numéro au préalable `6>>`

Commander	La description
`>>`	ligne de retrait
`<<`	ligne actuelle

Commander	La description
`6>>`	indent 6 lignes suivantes

Vous pouvez également indenter en utilisant des mouvements . Voici quelques exemples utiles.

Commander	La description
`>gg`	indentation de la ligne courante à la première ligne du fichier
`>G`	indentation de la ligne courante à la dernière ligne du fichier
`>{`	indenter paragraphe précédent
`>}`	indent paragraphe suivant

En mode visuel, appuyez une seule fois sur la touche supérieure ou inférieure à. Notez que cela provoque une sortie du mode visuel . Ensuite, vous pouvez utiliser `.` pour répéter l'édition si vous en avez besoin et `u` pour annuler.

Lire Échancrure en ligne: https://riptutorial.com/fr/vim/topic/6324/echancrure

Chapitre 19: Économie

Examples

Enregistrer un tampon dans un répertoire inexistant

```
:!mkdir -p %:h
```

pour créer les répertoires manquants, puis

```
:w
```

Lire Économie en ligne: https://riptutorial.com/fr/vim/topic/6440/economie

Chapitre 20: Espace blanc

Introduction

Voici comment vous pouvez nettoyer les espaces.

Remarques

Voir vimcast 4 transcription

Examples

Supprimer les espaces de fin dans un fichier

Vous pouvez supprimer les espaces à l'aide de la commande suivante.

```
:%s/\s\+$//e
```

Cette commande est expliquée comme suit:

- Entrez le mode de commande avec `:`
- faire cela dans le fichier entier avec `%` (par défaut serait pour la ligne actuelle)
- action de substitution `s`
- `/` début du motif de recherche
- `\s` caractère d' espacement
- `\+` échappé + signe, un ou plusieurs espaces doivent correspondre
- avant la fin `$` la ligne `$`
- `/` fin du motif de recherche, début du motif de remplacement
- `/` fin du motif de remplacement. Fondamentalement, remplacez par rien.
- `e` supprimer les messages d'erreur si aucune correspondance n'a été trouvée

Supprimer les lignes vierges dans un fichier

Vous pouvez supprimer toutes les lignes vides d'un fichier à l'aide de la commande suivante: g / ^ $ / d

Cette commande est expliquée comme suit:

- Entrez le mode de commande avec `:`
- `g` est une commande globale qui doit apparaître sur tout le fichier
- `/` début du motif de recherche
- le motif de recherche de la ligne vide est `^g`
- `/` fin du motif de recherche
- Commande Ex `d` supprime une ligne

Convertir les onglets en espaces et en espaces en onglets

Vous pouvez convertir des onglets en espaces en procédant comme suit:

Vérifiez d'abord que expandtab est désactivé

```
:set noexpandtab
```

alors

```
:retab!
```

qui remplace les espaces d'une certaine longueur par des onglets

Si vous activez à nouveau expandtab `:set expandtab` puis exécutez le `:retab!` commande alors tous les onglets deviennent des espaces.

Si vous voulez faire cela pour le texte sélectionné, sélectionnez d'abord le texte en mode visuel .

Lire Espace blanc en ligne: https://riptutorial.com/fr/vim/topic/8288/espace-blanc

Chapitre 21: Expressions régulières

Remarques

Exécuter `:h pattern` pour voir beaucoup d'informations liées aux regex

Examples

Mot

Vim a des opérateurs spéciaux pour faire correspondre les mots début, mot, fin, etc. `\<` représente le début d'un mot et `\>` représente la fin d'un mot.

Rechercher `/\<foo\>` dans le texte suivant ne renverra que le dernier foo.

> le football n'est pas `foo`

Lire Expressions régulières en ligne: https://riptutorial.com/fr/vim/topic/6533/expressions-regulieres

Chapitre 22: Expressions régulières en mode Ex

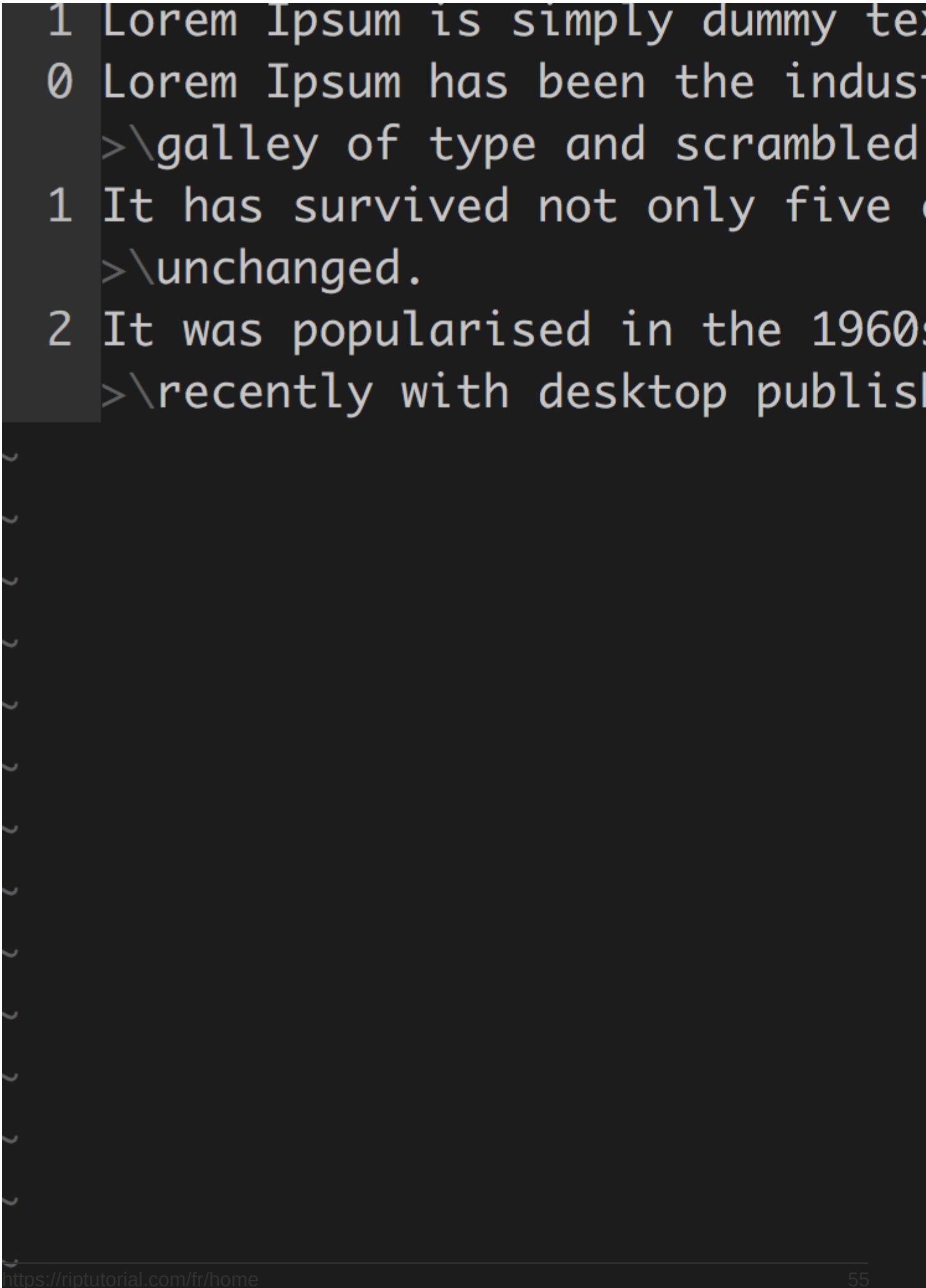
Examples

Modifier une expression régulière en mode Ex

Supposons que vous recherchiez un modèle de `Title Case` dans un fichier texte volumineux et que vous souhaitiez modifier une expression régulière incorrecte:

1. Tout d'abord, passez en mode `Ex` en tapant `q:`
2. Vous verrez maintenant toutes les commandes que vous avez tapées en mode `commandline` de `commandline` , appuyez sur `j` pour accéder à l'expression régulière que vous souhaitez modifier (`/\v[AZ]\w+\s[AZ]\w+`)
3. Une fois terminé, appuyez sur `ESC` pour passer en mode normal
4. Puis appuyez sur `Enter` pour lancer la recherche

Voici une capture d'écran montrant une recherche par `Title Case`


1 Lorem Ipsum is simply dummy tex
0 Lorem Ipsum has been the indus
>\galley of type and scrambled
1 It has survived not only five
>\unchanged.
2 It was popularised in the 1960
>\recently with desktop publis

https://riptutorial.com/fr/vim/topic/6472/expressions-regulieres-en-mode-ex

Chapitre 23: Fenêtres fendues

Syntaxe

- `:split <file>`
- `:vsplit <file>`
- `:sp` <- raccourci pour split
- `:vsp` <- raccourci pour vsplit

Remarques

Lorsqu'elle est appelée à partir de la ligne de commande, plusieurs fichiers peuvent être fournis dans l'argument et vim crée un fractionnement pour chaque fichier. Appelé à partir du mode ex, un seul fichier peut être ouvert par invocation de la commande.

Examples

Ouverture de plusieurs fichiers dans des divisions à partir de la ligne de commande

Horizontalement

```
vim -o file1.txt file2.txt
```

Verticalement

```
vim -O file1.txt file2.txt
```

Vous pouvez éventuellement spécifier le nombre de divisions à ouvrir. L'exemple suivant ouvre deux divisions horizontales et charge `file3.txt` dans un tampon:

```
vim -o2 file1.txt file2.txt file3.txt
```

Ouvrir une nouvelle fenêtre fractionnée

Vous pouvez ouvrir une nouvelle division dans Vim avec les commandes suivantes, en mode *normal* :

Horizontalement:

```
:split <file name>
:new
```

Verticalement:

```
:vsplit <file name>
:vnew
```

split ouvrira le fichier dans une nouvelle division en haut ou à gauche de votre écran (ou division actuelle) `:sp` et `:vs` sont des raccourcis pratiques.

new ouvrira un split vide

Changer la taille d'un split ou vsplit

Vous pouvez parfois vouloir changer la taille d'un split ou vsplit.

Pour modifier la taille du fractionnement actuellement actif, utilisez `:resize <new size>` . `:resize 30` par exemple rendrait le fractionnement de 30 lignes.

Pour modifier la taille du vsplit actif, utilisez `:vertical resize <new size>` . `:vertical resize 80` par exemple rendrait le vsplit de 80 caractères.

Raccourcis

- `Ctrl + w` et `+` augmentent la taille de la fenêtre fractionnée
- `Ctrl + w` et `-` diminue la taille de la fenêtre fractionnée
- `Ctrl + w` et `=` définir une taille égale à la fenêtre divisée

Ferme tous les splits mais le courant

Mode normal

`Ctrl-w o`

Mode ex

```
:only
```

ou court

```
:on
```

Gestion des fenêtres fractionnées ouvertes (raccourcis clavier)

Après avoir ouvert une fenêtre fractionnée dans vim (comme le montrent de nombreux exemples sous cette balise), vous souhaiterez probablement contrôler rapidement Windows. Voici comment contrôler les fenêtres fractionnées à l'aide des raccourcis clavier.

Déplacer pour diviser au-dessus / au-dessous:

- `Ctrl + w` et `k`
- `Ctrl + w` et `j`

Déplacer pour diviser Gauche / Droite:

- `Ctrl + w` et `h`
- `Ctrl + w` et `l`

Déplacer pour diviser ci-dessus / ci-dessous (wrap):

- `Ctrl + w` et `w`

Créer une nouvelle fenêtre vide:

- `Ctrl + w` et `n` -ou-: nouveau

Créer un nouveau split horizontal / vertical:

- `Ctrl` + `W` , `S` (majuscule)
- `Ctrl` + `W` , `v` (minuscule)

Faites en sorte que la division actuellement active soit celle affichée à l'écran:

- `Ctrl + w` et `o` -ou-: on

Déplacer entre les divisions

Pour vous déplacer sur la gauche, utilisez `<Cw><Ch>`
Pour passer à la division ci-dessous, utilisez `<Cw><Cj>`
Pour vous déplacer sur la droite, utilisez `<Cw><Ck>`
Pour passer à la division ci-dessus, utilisez `<Cw><Cl>`

Sane split opening

C'est une meilleure expérience pour ouvrir split ci-dessous et à droite

mettre en utilisant

```
set splitbelow
set splitright
```

Lire Fenêtres fendues en ligne: https://riptutorial.com/fr/vim/topic/1705/fenetres-fendues

Chapitre 24: Gammes de ligne de commande

Examples

Numéros de ligne absolus

La commande suivante s'exécute `:command` sur les lignes `23` à `56` :

```
:23,56command
```

NB: les plages sont *inclus* par défaut.

Numéros de ligne relatifs

Dans la commande suivante, la plage commence 6 lignes au-dessus de la ligne actuelle et termine 3 lignes ci-dessous:

```
:-6,+3command
```

Raccourcis de ligne

- `.` représente *la ligne actuelle,* mais elle peut également être omise entièrement.
- `$` représente *la dernière ligne* .
- `%` représente *la totalité du tampon* , c'est un raccourci pour `1,$` .

Les deux commandes ci-dessous exécutent `:command` sur chaque fichier de la ligne en cours jusqu'à la dernière ligne:

```
:.,$command
:,$command
```

La commande ci-dessous s'exécute `:command` sur tout le tampon:

```
:%command
```

Des notes

La commande ci-dessous exécute `:command` sur chaque ligne de celle contenant la marque manuelle `f` à celle contenant la marque manuelle `t` :

```
:'f,'tcommand
```

Les marques automatiques peuvent également être utilisées:

```
:'<,'>command    " covers the visual selection
```

```
:'{,'}command    " covers the current paragraph
:'[,']command    " covers the last changed text
```

Voir `:help mark-motions`.

Chercher

Les commandes ci - dessous exécuter `:command` sur chaque ligne à partir de la première mise en correspondance `from` la première correspondance `to` :

```
:/from/,/to/command    " from next 'from' to next 'to'
:?from?,/to/command    " from previous 'from' to next 'to'
:?from?,?to?command    " from previous 'from' to previous 'to'
```

Voir `:help search-commands`.

Décalage de ligne

Les décalages de lignes peuvent être utilisés pour ajuster les lignes de début et de fin:

```
:/foo/-,/bar/+4command    " from the line above next 'foo' to 4 lines below next 'bar'
```

Voir `:help search-offset`.

Gammes mixtes

Il est possible de combiner tout ce qui précède dans des gammes expressives:

```
:1267,/foo/-2command
:'{,command
:'f,$command
```

Soyez créatif et n'oubliez pas de lire `:help cmdline-ranges`.

Lire Gammes de ligne de commande en ligne: https://riptutorial.com/fr/vim/topic/3383/gammes-de-ligne-de-commande

Chapitre 25: Insérer du texte

Examples

Quitter le mode d'insertion

Commander	La description
`<Esc>`	Quitte le mode insertion, déclenche les autocommandes et les abréviations
`<C-[>`	Exactement synonyme de `<Esc>`
`<Cc>`	Laisse le mode insertion, ne déclenche pas de autocommandes

Certaines personnes aiment utiliser une paire de caractères relativement peu commune comme `jk` comme raccourci pour `<Esc>` ou `<C-[>` ce qui peut être difficile à atteindre sur certains claviers:

```
inoremap jk <Esc>l
```

Différentes façons d'entrer en mode insertion

Commander	La description
`a`	Ajouter du texte après la position actuelle du curseur
`A`	Ajouter du texte à la fin de la ligne en cours
`i`	Insérer du texte avant la position actuelle du curseur
`I`	Insérer du texte avant le premier caractère non vide de la ligne en cours
`gI`	Insérer du texte dans la première colonne de la ligne du curseur
`gi`	Insérer du texte au même endroit où il a été laissé la dernière fois en mode Insertion
`O`	Ouvrez une nouvelle ligne au-dessus de la ligne actuelle et ajoutez-y du texte (CAPITAL `o`)
`o`	Ouvrez une nouvelle ligne sous la ligne actuelle et ajoutez-y du texte (`o` minuscule)
`s` ou `cl`	Supprimer le caractère sous le curseur et passer en mode insertion
`S` ou `cc`	Supprimer toute la ligne et passer en mode Insertion

Commander	La description
`C`	Supprimer de la position du curseur à la fin de la ligne et lancer le mode d'insertion
`c{motion}`	Supprimer `{motion}` et lancer le mode d'insertion (voir Mouvement de base)

Raccourcis du mode d'insertion

Commander	La description
`<Cw>`	Supprimer le mot avant le curseur
`<Ct>`	Ligne de courant de `shiftwidth` avec une `shiftwidth`
`<Cd>`	Ligne de courant non indenté avec une `shiftwidth`
`<Cf>`	réindenter la ligne, (déplacer le curseur sur la position d'indentation automatique)
`<Ca>`	Insérer du texte précédemment inséré
`<Ce>`	Insérer le caractère ci-dessous
`<Ch>`	Supprimer un caractère en arrière
`<Cy>`	Insérer le caractère ci-dessus
`<Cr>{register}`	Insérer le contenu de `{register}`
`<Co>{normal mode command}`	exécuter `{normal mode command}` sans quitter le mode insertion
`<Cn>`	Option autocomplete suivante pour le mot actuel
`<Cp>`	Option de saisie semi-automatique précédente pour le mot actuel
`<Cv>`	Insérer un caractère par sa valeur ASCII en décimal
`<Cv>x`	Insérer un caractère par sa valeur ASCII en hexadécimal
`<Cv>u`	Insérer un caractère par sa valeur unicode en hexadécimal
`<Ck>`	Entrez un digraphe

Exécution de commandes normales à partir du mode insertion

En mode insertion, appuyez sur `<Co>` pour quitter temporairement le mode insertion et exécuter une commande normale unique.

Exemple

`<Co>2w` saute au deuxième mot à gauche et retourne au mode insertion.

Note: Répéter avec . ne répètera que les actions de retour en mode insertion

Cela permet des mappages utiles, par exemple

```
inoremap <C-f> <Right>
inoremap <C-b> <Left>
inoremap <C-a> <C-o>^
inoremap <C-e> <C-o>$
```

Maintenant, ctrl + a placera le curseur au début de la ligne et ctrl + e - à la fin de la ligne. Ces mappages sont utilisés par défaut dans `readline` . Ils peuvent donc être utiles pour les personnes qui souhaitent une cohérence.

Insérer du texte sur plusieurs lignes à la fois

Appuyez sur `Ctrl` + `v` pour entrer en mode bloc visuel.

Utilisez `↑` / `↓` / `j` / `k` pour sélectionner plusieurs lignes.

Appuyez sur `Maj` + `i` et commencez à taper ce que vous voulez.

Après avoir `appuyé` sur `Echap` , le texte sera inséré dans toutes les lignes sélectionnées.

Rappelez-vous que `Ctrl` + `c` n'est pas équivalent à 100% à `Esc` et ne fonctionnera pas dans cette situation!

Il y a de légères variations de `Maj` + `i` que vous pouvez appuyer en mode bloc visuel:

Clé	La description
`c` ou `s`	Supprimer le bloc sélectionné et entrer en mode insertion
`C`	Supprimer les lignes sélectionnées (du curseur jusqu'à la fin) et entrer en mode insertion
`R`	Supprimer les lignes sélectionnées et entrer en mode insertion
`UNE`	Ajouter aux lignes sélectionnées, avec la colonne à la fin de la première ligne

Notez également que vous appuyez sur . après une opération de bloc visuel répétera cette opération où le curseur est!

Coller du texte en utilisant la commande "coller" du terminal

Si vous utilisez la commande paste de votre programme d'émulation de terminal, Vim interprétera le flux de caractères comme s'ils étaient tapés. Cela provoquera toutes sortes d'effets indésirables, en particulier une mauvaise indentation.

Pour corriger cela, à partir du mode de commande:

```
:set paste
```

Ensuite, passez au mode d'insertion, avec `i` , par exemple. Notez que le mode est maintenant `-- INSERT (paste) --` . Collez maintenant avec votre commande d'émulateur de terminal ou avec la souris. Une fois terminé, passez en mode commande, avec `Esc` et exécutez:

```
:set nopaste
```

Il y a un moyen plus simple, quand on veut coller juste une fois. Mettez ceci dans votre *fichier .vimrc* (ou utilisez le plugin *unimpaired.vim*):

```
function! s:setup_paste() abort
  set paste
  augroup unimpaired_paste
    autocmd!
    autocmd InsertLeave *
      \ set nopaste |
      \ autocmd! unimpaired_paste
  augroup end
endfunction

nnoremap <silent> yo :call <SID>setup_paste()<CR>o
nnoremap <silent> yO :call <SID>setup_paste()<CR>O
```

Maintenant, vous pouvez simplement appuyer sur `yo` pour coller le code sous le curseur, puis sur `<Esc>` pour revenir au mode normal / nopaste.

Collage depuis un registre en mode insertion

En mode insertion, vous pouvez utiliser `<Cr>` pour coller depuis un registre spécifié par la frappe suivante. `<Cr>"` par exemple les pâtes du registre sans nom (`"`).

Voir `:help registers` .

Commandes et raccourcis d'insertion avancés

Voici une référence rapide pour l'insertion avancée, le formatage et le filtrage des commandes / raccourcis.

Commande / raccourci	*Résultat*
`g +?` + *m*	Effectuer le codage rot13, en mouvement *m*
n + `ctrl + a`	+ *n* pour numéroter sous le curseur

Commande / raccourci	*Résultat*
n + `ctrl + x`	- *n* au numéro sous le curseur
`g + q` + *m*	Formater les lignes de mouvement *m* en largeur fixe
: *r* ce w	Lignes centrales dans la plage *r* à la largeur *w*
: *r* le i	Aligner les lignes dans la plage *r* avec l'indentation *i*
: *r* ri *w*	Aligner à droite les lignes dans la plage *r* à la largeur *w*
! *mc*	Filtrer les lignes de mouvement *m* par la commande *c*
n !! *c*	Filtrer *n* lignes par la commande *c*
: *r* ! *c*	Filtrer la plage *r* lignes par la commande *c*

Désactiver l'indentation automatique pour coller le code

Lors du collage de texte via un émulateur de terminal, la fonctionnalité de retrait automatique *peut* détruire l'indentation du texte collé.

Par exemple:

```
function () {
    echo 'foo'
    echo 'bar'
    echo 'baz'
}
```

sera collé comme:

```
function () {
    echo 'foo'
        echo 'bar'
            echo 'baz'
            }
```

Dans ces cas, utilisez l'option `paste` / `nopaste` pour désactiver / activer la fonctionnalité de retrait automatique:

```
:set paste
:set nopaste
```

À cela s'ajoute une approche plus simple du problème: ajoutez la ligne suivante dans votre fichier .vimrc:

```
set pastetoggle=<F3>
```

Et si vous voulez coller tel quel du presse-papiers. Appuyez simplement sur `F3` en mode `insert` et collez. Appuyez à nouveau sur `F3` pour quitter le mode `paste` .

Lire Insérer du texte en ligne: https://riptutorial.com/fr/vim/topic/953/inserer-du-texte

Chapitre 26: L'opérateur de points

Examples

Utilisation de base

L'opérateur point répète la dernière action effectuée, par exemple:

fichier `test.tx`

```
helo, World!
helo, David!
```

(curseur sur [1] [1])
Maintenant, effectuez un `cwHello<Esc>` (changez le mot `helo` en `Hello`)
Maintenant, le tampon ressemble à ça:

```
Hello, World!
helo, David!
```

(curseur sur [1] [5])
Après avoir tapé `j_` le curseur est sur [2] [1].

Maintenant, entrez le `.` et la dernière action est effectuée à nouveau:

```
Hello, World!
Hello, David!
```

(curseur sur [2] [5])

Définir l'indentation

Ceci est très utile lors de la mise en retrait de votre code

```
if condition1
if condition2
# some commands here
endif
endif
```

déplacez votre curseur sur la 2ème ligne, puis `>>` , le code sera indenté à droite.

Maintenant, vous pouvez répéter votre action en continuant à la 3ème ligne, puis appuyez sur `.` deux fois, le résultat sera

```
if condition1
    if condition2
```

```
        # some commands here
endif
endif
```

Lire L'opérateur de points en ligne: https://riptutorial.com/fr/vim/topic/3665/l-operateur-de-points

Chapitre 27: Macros

Examples

Enregistrer une macro

Une façon de créer une macro est de l' *enregistrer* .

Commencez à enregistrer une macro et enregistrez-la dans un registre (dans cet exemple, nous utiliserons `a` , mais il peut s'agir de n'importe quel registre sur lequel vous pouvez normalement taper du texte):

```
qa
```

Ensuite, exécutez les commandes que vous souhaitez enregistrer dans la macro (ici, nous allons entourer le contenu d'une ligne avec des balises `<li>`):

```
I<li><ESC>A</li>
```

Lorsque nous en avons fini avec les commandes que nous voulons enregistrer dans la macro, arrêtez l'enregistrement:

```
q
```

Maintenant, chaque fois que nous voulons exécuter la séquence enregistrée de commandes stockées dans `a` , utilisez:

```
@a
```

et vim répétera la séquence enregistrée.

La prochaine fois que vous souhaitez répéter la dernière macro utilisée, vous pouvez taper deux fois `@` :

```
@@
```

Et comme bonus supplémentaire, il est bon de se rappeler que si vous mettez un nombre avant une commande, il le répétera plusieurs fois. Donc, vous répétez la macro enregistrée dans le registre `a` 20 fois avec:

```
20@a
```

Modification d'une macro vim

Parfois, vous commettez une erreur avec une longue macro, mais vous préférez la modifier plutôt

que de la réenregistrer entièrement. Vous pouvez le faire en utilisant le processus suivant:

1. Placez la macro sur une ligne vide avec `"<register>p` .

   Si votre macro est enregistrée dans le registre `a` , la commande est `"ap` .

2. Modifiez la macro selon vos besoins.

3. Yank la macro dans le registre correct en déplaçant le curseur au début de la ligne et en utilisant `"<register>y$` .

   Vous pouvez réutiliser le registre d'origine ou en utiliser un autre. Si vous voulez utiliser le registre `b` , la commande est `"by$` . Ou en utilisant `"<register>d$` (supprime la ligne inutilisée)

Macros récursives

Les macros Vim peuvent également être récursives. Ceci est utile lorsque vous devez agir sur chaque ligne (ou autre objet texte) jusqu'à la fin du fichier.

Pour enregistrer une macro récursive, commencez par un registre vide. (Un registre peut être vidé en utilisant `q<register>q` .)

Choisissez un point de départ cohérent sur chaque ligne pour commencer et terminer.

Avant de terminer l'enregistrement, appelez la macro comme dernière commande. (C'est pourquoi le registre doit être vide: il ne fera rien car la macro n'existe pas encore).

Exemple, donné le texte:

```
line 1
line 2
line 3
foo bar
more random text
.
.
.
line ???
```

En mode normal, avec le curseur sur la première ligne et un registre vide `a` , on pourrait enregistrer cette macro:

```
qaI"<Esc>A"<Esc>j@aq
```

Ensuite, avec une seule invocation de `@a` , toutes les lignes du fichier seront désormais entre guillemets.

Qu'est-ce qu'une macro?

Une macro est une série de touches destinées à être "relues" par Vim sans délai. Les macros peuvent être stockées dans des registres ou des variables, liées à des clés ou exécutées sur la

ligne de commande.

Voici une macro simple qui majuscule le troisième `word` sur une ligne:

```
0wwgUiw
```

Cette macro pourrait être *enregistrée* dans le registre `q` :

```
qq          start recording into register q
0wwgUiw
q           stop recording
```

ou enregistré directement dans le registre `q` :

```
:let @q = '0wwgUiw'
```

à lire avec:

```
@q
```

Mais il pourrait également être saisi directement dans la ligne de commande:

```
:normal 0wwgUiw
```

pour une lecture instantanée via la commande `:normal` .

Ou mettre dans une variable:

```
:let myvar = '0wwgUiw'
```

à lire avec:

```
@=myvar
```

Ou enregistré en tant que mappage:

```
nnoremap <key> 0wwgUiw
```

à lire en appuyant sur `<key>` .

Si vous souhaitez stocker une macro pour une réutilisation ultérieure, vous pouvez taper en mode insertion:

```
<C-r>q
```

Cela insère la macro dans le registre `q` (dans cet exemple: `0wwgUiw`). Vous pouvez utiliser cette sortie par exemple pour définir la macro dans votre `vimrc` :

```
let @q='0wwgUiw'
```

En procédant ainsi, le registre `q` est initialisé avec cette macro chaque fois que vous démarrez vim.

Action d'enregistrement et de relecture (macros)

avec la commande `q` nous pourrions simplifier beaucoup de travail fastidieux en vim.

exemple 1. générer une séquence de tableaux (1 à 20).

ÉTAPE 1. Appuyez sur `i` pour accéder au mode d'insertion, entrez `1`

```
1
```

ÉTAPE 2. Enregistrez l' action suivante: "ajoutez le dernier numéro à la ligne suivante et augmentez le nombre"

1. tapez `esc` pour quitter le mode de saisie
2. tapez `qa` pour entrer en mode enregistrement, en utilisant `a` tampon `a`
3. tapez `yy` et `p` pour copier la ligne courante et la coller comme ligne suivante
4. tapez `ctrl` + `a` pour augmenter le nombre
5. tapez `q` nouveau pour terminer l'enregistrement

```
1
2
```

ÉTAPE 3. Relisez l' action 18 fois.

tapez `18@a` pour relire l'action 3 et l'action 4 à l'étape 2.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Lire Macros en ligne: https://riptutorial.com/fr/vim/topic/1447/macros

Chapitre 28: Manipulation du texte

Remarques

Pour augmenter et diminuer les choses comme `11:59AM` , `3rd` et `XVIII` , utilisez le plugin vim-Speeddating

Examples

Conversion de casse

En mode normal:

- `~` inverse la casse du caractère sous le curseur,
- `gu{motion}` minuscule le texte couvert par `{motion}` ,
- `gU{motion}` majuscule le texte couvert par `{motion}`

Exemple (`^` marque la position du curseur):

```
Lorem ipsum dolor sit amet.
      ^
Lorem ipSum dolor sit amet.    ~
Lorem IPSUM DOLOR sit amet.    gU2w
Lorem IPsum DOLOR sit amet.    gue
```

En mode visuel:

- `~` inverse la casse du texte sélectionné,
- `u` minuscule le texte sélectionné,
- `U` majuscule le texte sélectionné

Exemple (`^^^` marque la sélection visuelle):

```
Lorem ipsum dolor sit amet.
       ^^^^^^^^^^^^
Lorem ipSUM DOLOR SIT amet.    ~
Lorem ipSUM DOLOR SIT amet.    U
Lorem ipsum dolor sit amet.    u
```

Incrémentation et décrémentation des nombres et des caractères alphabétiques

En mode normal, nous pouvons incrémenter le nombre le plus proche sur la ligne à ou après le curseur avec `<Ca>` et le décrémenter avec `<Cx>` . Dans les exemples suivants, la position du curseur est indiquée par `^` .

Incrémenter et décrémenter des nombres

```
for i in range(11):
      ^
```

`<Cx>` décrémente le nombre:

```
for i in range(10):
               ^
```

`10<Ca>` incrémente de `10` :

```
for i in range(20):
               ^
```

Incrémentation et décrémentation des caractères alphabétiques

Pour que l'incrémentation et la décrémentation fonctionnent aussi avec des lettres, utilisez la commande ex `:set nrformats+=alpha` ou ajoutez `set nrformats+=alpha` à votre `.vimrc` .

Exemple d'incrément:

```
AAD
 ^
```

`<Ca>` incrémente à `B` :

```
ABD
 ^
```

Décrément exemple:

```
ABD
  ^
```

`<Cx>` décrémente `D` à `C` :

```
ABC
  ^
```

Incrémentation et décrémentation des nombres lorsque l'incrémentation / décrémentation alphabétique est activée

Notez que l'activation de l'incrémentation / décrémentation pour travailler avec des caractères alphabétiques signifie que vous devez faire attention à ne pas les modifier lorsque vous souhaitez simplement modifier les nombres. Vous pouvez soit désactiver l'incrémentation / décrémentation alphabétique en utilisant la commande ex `:set nrformats-=alpha` ou vous pouvez simplement être

conscient de cela et être sûr de passer au numéro avant incrémenter ou décrémenter. Voici le " `for i in range(11):` " `for i in range(11):` exemple ci-dessus est repris pour fonctionner alors que l'incrémentation / décrémentation alphabétique est définie:

Disons que vous voulez diminuer `11` à `10` et l'incrémentation / décrémentation alphabétique est active.

```
for i in range(11):
      ^
```

Comme l'incrémentation / décrémentation alphabétique est active, pour éviter de modifier le caractère sous le curseur, avancez d'abord vers le premier `1` utilisant la commande de déplacement en *mode normal* `f1` (c'est-à-dire minuscule `f` suivi du chiffre `1`):

```
for i in range(11):
               ^
```

Maintenant, puisque le curseur est sur le nombre, vous pouvez le décrémenter avec `<Cx>` . Lors du décrément, le curseur est repositionné sur le dernier chiffre du chiffre:

```
for i in range(10):
                ^
```

Code de mise en forme

En mode normal:

`gg` aller en haut

`=` alors `G`

Utiliser des "verbes" et des "noms" pour l'édition de texte

L'une des façons de penser aux commandes à exécuter, de modifier un texte d'une certaine manière, est d'utiliser des phrases entières.

Une commande est une action effectuée sur un objet. Par conséquent, il a un verbe:

```
:normal i    " insert
:normal a    " append
:normal c    " overwrite
:normal y    " yank (copy)
:normal d    " delete
```

Certains de ces mots fonctionnent avec un objet comme `d` , `c` , `y` . Ces objets peuvent être **mot, ligne, phrase, paragraphe, tag** . On peut les utiliser en combinaison:

```
:normal dw    " deletes the text from the position of the cursor to the end of the next word
:normal cw    " deletes the text from the cursor to the end of the next word and
```

```
              " enters insert mode
```

On pourrait aussi utiliser un **modificateur** pour spécifier précisément où l'action doit être exécutée:

```
:normal diw    " delete inside word. I.e. delete the word in which is the cursor.
:normal ciw    " removes the word, the cursor points at and enters insert mode
:normal ci"    " removes everything between the opening and closing quotes and
               " enters insert mode
:normal cap    " change the current paragraph
:normal ct8    " remove everything until the next number 8 and enter insert mode
:normal cf8    " like above but remove also the number
:normal c/goal " remove everything until the word 'goal' and enter insert mode
:normal ci{    " change everything inside the curly braces
```

Davantage de ressources:

Apprenez à parler vim - verbes, noms et modificateurs!

Learning Vim en 2014: Vim en tant que langue

Modification de VimSpeak à l'aide de la grammaire vocale

Lire Manipulation du texte en ligne: https://riptutorial.com/fr/vim/topic/1707/manipulation-du-texte

Chapitre 29: Mappages de touches dans Vim

Introduction

La mise à jour des mappages de clés Vim vous permet de résoudre deux types de problèmes: Réaffecter des raccourcis clavier aux lettres les plus mémorables ou accessibles, et créer des raccourcis clavier pour les fonctions qui n'en ont pas. Vous apprendrez ici les différentes façons de mapper les commandes clés et le contexte auquel elles s'appliquent *(par exemple les modes vim)*

Examples

Cartographie de base

carte Aperçu

Une séquence de touches peut être ré-associée à une autre séquence de touches à l'aide de l'une des variantes de la `map` .

Par exemple, la `map` va sortir du *mode Insertion* lorsque vous appuyez sur `j k` en séquence rapide:

```
:inoremap jk <Esc>
```

Opérateur de carte

Il existe plusieurs variantes de `:map` pour différents modes.

Commandes	Modes
`:map` `:noremap` `:unmap`	Mode normal, visuel et en attente d'opérateur
`:map!` , `:noremap!` , `:unmap!`	Mode d'insertion et de ligne de commande
`:nmap` `:nnoremap` `:nunmap`	Mode normal
`:imap` `:inoremap` `:iunmap`	Mode d'insertion et de remplacement
`:vmap` `:vnoremap` `:vunmap`	Mode visuel et sélection
`:xmap` `:xnoremap` `:xunmap`	Mode visuel
`:smap` `:snoremap` `:sunmap`	Sélectionnez le mode
`:cmap` `:cnoremap` `:cunmap`	Mode ligne de commande

Commandes	Modes
`:omap :onoremap :ounmap`	Mode en attente de l'opérateur

En règle générale, vous devez utiliser les variantes `:noremap` ; Cela rend le mappage insensible aux remappages et aux récursions.

commande de carte

- Vous pouvez afficher tous les mappages en utilisant `:map` (ou l'une des variantes ci-dessus).
- Pour afficher le mappage actuel pour une séquence de touches spécifique, utilisez `:map <key>` où `<key` > est une séquence de clés
- Les touches spéciales comme `Esc` sont mappées en utilisant une notation spéciale `<>` , comme `<Esc>` . Pour la liste complète des codes clés, voir http://vimdoc.sourceforge.net/htmldoc/intro.html#keycodes
- `:nmapclear` - Effacer toutes les cartes en mode normal
- `:nunmap` - Décompressez une carte en mode normal
- Vous pouvez configurer le temps maximum entre les clés d'une séquence en modifiant les variables `timeout` et `ttimeout`

Exemples

- `imap jk <Esc>` : taper `jk` en mode insertion vous ramènera au mode normal
- `nnoremap tt :tabnew<CR>` : `tt` en mode normal ouvrira une nouvelle page d'onglet
- `nnoremap <Cj> <Cw>j` : taper `<Cj>` en mode normal vous fera sauter à la fenêtre ci-dessous et à gauche
- `vmap <Cc> \cc` : taper `<Cc>` en mode visuel exécutera `\cc` (commande NERDCommenter pour commenter la ligne). Comme cela repose sur un mappage de plugin, vous ne pouvez pas utiliser `:vnoremap` ici!

  Lire plus ici

Combinaison de touches du leader de la carte

La clé principale peut être utilisée pour créer un mappage avec une liaison de clé pouvant être remplacée par l'utilisateur final.

Le leader est la touche `\` par défaut. Pour le remplacer, l'utilisateur final doit exécuter `:let g:mapleader='somekey(s)'` avant de définir le mappage.

Dans un scénario classique, le mapleader est défini dans le `.vimrc` , et les plug-ins utilisent `<Leader>` dans la partie keybinding de leurs mappages pour les personnaliser.

Dans le plugin, nous définirions des mappages avec:

```
:nnoremap <Leader>a somecomplexaction
```

Ce mapperait l'action `somecomplexaction` à la `\` + `une` combinaison de touches.

`L'une` action sans le leader ne change pas.

Il est également possible d'utiliser des `<Plug>Mappings` pour laisser plus d'espace pour personnaliser les raccourcis des plugins.

Illustration du mappage de base (raccourcis pratiques).

Dans la plupart des éditeurs de texte, le raccourci standard pour enregistrer le document en cours est `Ctrl` + `S` (ou `Cmd` + `S` sur macOS).

Vim n'a pas cette fonctionnalité par défaut mais cela peut être mappé pour faciliter les choses. L'ajout des lignes suivantes dans le fichier `.vimrc` fera l'affaire.

```
nnoremap <c-s> :w<CR>
inoremap <c-s> <c-o>:w<CR>
```

La commande `nnoremap` mappe `Ctrl` + `s` sur `:w` (commande le contenu actuel dans le fichier) alors que la commande `inoremap` mappe la commande `Ctrl` + `S` sur `:w` et revient au mode insertion (`<co>` passe en mode normal pour une commande et retourne ensuite au mode d'insertion sans modifier la position du curseur que d'autres solutions comme `<esc>:w<cr>a` ne peuvent pas assurer).

De même,

```
" This is commented, as Ctrl+Z is used in terminal emulators to suspend the ongoing
program/process.
" nnoremap <c-z> :u<CR>

" Thus, Ctrl+Z can be used in Insert mode
inoremap <c-z> <c-o>:u<CR>

" Enable Ctrl+C for copying selected text in Visual mode
vnoremap <c-c> <c-o>:y<CR>
```

PS: Cependant, il faut noter que `Ctrl` + `S` peut ne pas fonctionner comme prévu lors de l'utilisation de ssh (ou PuTTY). La solution à ce problème n'entre pas dans le cadre de ce document, mais peut être trouvée ici .

Lire Mappages de touches dans Vim en ligne: https://riptutorial.com/fr/vim/topic/3535/mappages-de-touches-dans-vim

Chapitre 30: Modes - insérer, normal, visuel, ex

Examples

Les bases des modes

`vim` est un éditeur modal. Cela signifie qu'à tout moment dans une session `vim` , l'utilisateur sera dans l'un des modes de fonctionnement. Chacun offre un ensemble de commandes, d'opérations, de raccourcis clavier différents ...

Mode normal (ou mode commande)

- Le mode `vim` commence dans
- Depuis les autres modes, généralement accessibles par `Esc` .
- **A la plupart des commandes de navigation et de manipulation de texte.**

Voir `:help normal-mode` .

Mode d'insertion

- Habituellement consulté par: `a` , `i` , `A` , `I` , `c` , `s` .
- **Pour insérer du texte**

Voir `:help insert-mode` .

Mode visuel

- Habituellement consulté par: `v` (par caractère), `V` (par ligne), `<Cv>` (par blocs).
- Fondamentalement, pour la **sélection de texte** ; la plupart des commandes normales sont disponibles, plus des commandes supplémentaires pour agir sur le texte sélectionné.

Voir `:help visual-mode` .

Sélectionnez le mode

- Accessible depuis le mode insertion avec `<Cg>` .
- Semblable au mode visuel mais avec beaucoup moins de commandes disponibles.
- Contrairement au mode insertion, il est possible de taper tout de suite.
- Rarement utilisé.

Voir `:help select-mode` .

Mode de remplacement

- Accessible depuis le mode normal avec `R`
- Permet de remplacer le texte existant.

Voir `:help replace-mode`.

Mode ligne de commande

Voir `:help command-line-mode`.

Mode ex

Voir `:help Ex-mode`.

Lire Modes - insérer, normal, visuel, ex en ligne: https://riptutorial.com/fr/vim/topic/2231/modes---inserer--normal--visuel--ex

Chapitre 31: Motions et objets texte

Remarques

Un objet texte dans Vim est un autre moyen de spécifier un morceau de texte sur lequel travailler. Ils peuvent être utilisés avec des opérateurs ou en mode visuel, au lieu de mouvements.

Examples

Modification du contenu d'une chaîne ou d'une liste de paramètres

Disons que vous avez cette ligne de code:

```
printf("Hello, world!\n");
```

Maintenant, dites que vous voulez changer le texte en "Programme sortant".

Commander	Tampon	Mnémonique
`ci"`	`printf("¦");`	**c** hange **dans** le **"** .
`Program exiting.\n<esc>`	`printf("Program exiting.\n");`	

Lire Motions et objets texte en ligne: https://riptutorial.com/fr/vim/topic/4107/motions-et-objets-texte

Chapitre 32: Mouvement

Examples

Recherche

Sauter aux personnages

`f {char}` - passe à la prochaine occurrence de `{char}` à droite du curseur sur la même ligne

`F {char}` - passe à la prochaine occurrence de `{char}` à gauche du curseur sur la même ligne

`t {char}` - se déplace vers la gauche de l'occurrence suivante de `{char}` à droite du curseur sur la même ligne

`T {char}` - se déplace vers la droite de l'occurrence suivante de `{char}` à gauche du curseur sur la même ligne

Avancer / reculer entre les résultats via `;` et `,`.

De plus, vous pouvez rechercher des mots entiers via `/<searchterm> Enter`.

Recherche de chaînes

`*` - passer à l'occurrence suivante du mot sous le curseur

`#` - déplacer à l'occurrence précédente du mot sous le curseur

`/ searchterm Enter` vous amène au prochain match (recherche avancée). Si vous utilisez `?` au lieu de `/`, la recherche recule.

Sautez entre les matchs via `n` (suivant) et `N` (précédent).

Pour afficher / modifier vos recherches précédentes, tapez `/` et appuyez sur la touche fléchée `vers` le `haut`.

Utiles sont aussi ces paramètres: (note `:se` est égal à `:set`)

- `:se hls` HighLightSearch, met en évidence toutes les correspondances de recherche; utiliser `:noh` pour désactiver temporairement la recherche / mise en surbrillance (`:set noh` ou `:set nohls` est désactivé.)
- `:se is` ou `:set incs` active la recherche incrémentielle, le curseur passe automatiquement à la correspondance suivante. (`:se nois` désactive.)
- `:se ic` IgnoreCase, désactive la sensibilité à la casse. (`:se noic` s'allume à nouveau.)
- `:se scs` SmartCaSe, peut être utilisé lorsque IgnoreCase est défini; rend la sensibilité **à la**

casse **intelligente** ! par exemple , `/the` recherchera `the` , `The` , `ThE` , etc. tout `/The` seule recherchera `The` .

Mouvement de base

Remarques

- Chaque mouvement peut être utilisé après une commande d'opérateur, de sorte que la commande opère sur le texte compris dans la portée du mouvement.
- Tout comme les commandes d'opérateur, les mouvements peuvent inclure un décompte, vous pouvez par `exemple` vous déplacer par `2w ords` .

Flèches

Dans Vim, les touches fléchées / curseur normales (`←` `↓` `↑` `→`) fonctionnent comme prévu. Cependant, pour les retouches typers, il est plus facile d'utiliser les touches `h` `j` alternatives `k` `l`. Sur un clavier typique, ils sont situés l'un à côté de l'autre sur la même ligne et facilement accessibles avec la main droite. La technique mnémonique à retenir qui est celle qui parmi eux va comme ceci:

- `h` / `l` - ceux-ci sont situés "le plus à gauche / à droite" parmi les quatre lettres du clavier, ils sont donc équivalents à "aller à gauche / droite" respectivement;
- `j` - la minuscule "j" a sa queue allant "en bas" au-dessous des lettres typiques, comme une petite flèche - donc elle équivaut à "descendre";
- `k` - à l'inverse, les minuscules "k" ont leur "ascendeur" qui "monte" au-dessus des lettres typiques, comme un petit pointeur - ce qui équivaut à "monter".

Mouvements de base

Toutes les commandes ci-dessous doivent être effectuées en **mode normal** .

Commander	La description
`h` ou à `gauche`	aller `[compter]` caractères à gauche
`j` ou `vers le bas`	aller `[compter]` caractères ci-dessous
`k` ou en `haut`	aller `[compter]` caractères ci-dessus
`l` ou `droit`	aller `[compter] les` caractères à droite
`gg`	aller à la première ligne, ou `[compter]` la ligne, si elle est donnée
`H`	aller à la première ligne de l'écran visible

Commander	La description
`M`	aller à la ligne médiane dans l'écran visible
`L`	aller à la dernière ligne de l'écran visible
`g`	aller à la dernière ligne, ou `[compter]` la ligne, si donnée
`Maison` ou `0`	aller au premier caractère de la ligne
`^`	aller au premier caractère non vide de la ligne
`+`	descendre une ligne au premier caractère non vide
`-`	remonter d'une ligne au premier caractère non vide
`$` ou `fin`	aller à la fin de la ligne (si `[count]` est donné, aller `[count - 1]` lignes vers le bas)
`|`	aller au `[comte]` 'e caractère ou aller au début de la ligne si `count` non spécifié
`f {char}`	aller à `[count]` 'occurrence de `{char}` à droite *inclusivement*
`F {char}`	aller à `[count]` 'occurrence de `{char}` à gauche *inclus*
`t {char}`	aller à `[count]` 'occurrence de `{char}` à droite *exclusive*
`T {char}`	aller à `[count]` 'occurrence de `{char}` à gauche *exclusive*
`;`	répéter les dernières fois `f` , `t` , `F` ou `T` `[count]`
`,`	Répéter les dernières `f` , `t` , `F` ou `T` , dans la direction opposée, `[compter]` fois
`w`	aller au début du mot suivant
`b`	aller au début du mot précédent
`e`	aller à la fin du mot suivant
`ge`	aller à la fin du mot précédent
`% De`	aller aux paires correspondantes, par exemple `(), [], {}` , `/* */` ou `#if, #ifdef, #else, #elif, #endif`
`{ }`	paragraphe précédent / suivant
`[{ ]}`	début / fin du bloc
`'{carboniser}`	Aller à la marque (marquer avec `m {char}`)
`<CB> <CF>`	page précédente / suivante

Commander	La description
`<CO> <CI>`	Retournez ou foward dans la "liste de saut" (nécessite `jumplist` fonctionnalité de `jumplist` , voir `:help jumps`)

Remarque: `b` , `e` et `w` considèrent un mot comme étant des lettres, des chiffres et des traits de soulignement par défaut (cela peut être configuré avec le paramètre `iskeyword`). Chacun de ces éléments peut également être mis en majuscule, ce qui les oblige à ignorer tout ce qui n'est pas non plus blanc.

Remarque: Vim reconnaît deux types de mouvements: le mouvement de l'opérateur (`:help movement`) et les sauts (`:help jumplist`). Les mouvements comme ceux exécutés avec `g` (`gg` , `G` , `g,`) comptent comme des sauts, tout comme les changements. Les changements obtiennent leur propre liste, qui est navigable comme mentionné ci-dessus via `g,` et `g;` (voir `:help changelist`). Les sauts ne sont pas traités comme des commandes de mouvement par Vim

Lorsque vous vous déplacez vers le haut ou vers le bas sur plusieurs lignes, le curseur conserve sa colonne comme prévu. Si la nouvelle ligne est trop courte, le curseur se déplace à la fin de la nouvelle ligne. Si la colonne est au-delà de la fin de la ligne, le curseur est affiché à la fin de la ligne. Le numéro de colonne initial est conservé jusqu'à ce qu'une action soit prise pour le modifier (par exemple, modifier du texte ou déplacer explicitement une colonne).

Si la longueur d'une ligne dépasse la largeur de l'écran, le texte est encapsulé (sous les paramètres par défaut, ce comportement peut être configuré). Pour vous déplacer parmi les lignes affichées à l'écran, plutôt que sur les lignes du fichier, ajoutez `g` devant la commande habituelle. Par exemple, `gj` déplace le curseur sur la position affichée sur une ligne en dessous de sa position actuelle, même si celle-ci se trouve sur la même ligne du fichier.

Recherche de motif

Vim prend en charge l'utilisation d'expressions régulières lors de la recherche dans un fichier.

Le caractère pour indiquer que vous souhaitez effectuer une recherche est `/` .

La recherche la plus simple que vous pouvez effectuer est la suivante

```
/if
```

Cela recherchera le fichier entier pour toutes les instances de `if` . Cependant, notre recherche `if` est en fait une expression régulière qui correspondra à toute occurrence du mot `if` , y compris ceux à l' intérieur d'autres mots.

Par exemple, notre recherche dirait que tous les mots suivants correspondent à notre recherche: `if` , `spiffy` , `endif` , etc.

Nous pouvons effectuer des recherches plus compliquées en utilisant des expressions régulières plus complexes.

Si notre recherche était:

```
/\<if\>
```

alors notre recherche ne renverrait que des correspondances exactes au mot complet `if` . Le `spiffy` ci-dessus et `endif` ne seraient pas retournés par la recherche, seulement `if` .

Nous pouvons également utiliser des plages. Donné un fichier:

```
hello1
hello2
hello3
hello4
```

Si nous voulons rechercher ces lignes contenant "bonjour" suivi d'un chiffre entre 1 et 3, nous dirions:

```
/hello[1-3]
```

Un autre exemple:

```
/(?:\d*\.)?\d+
```

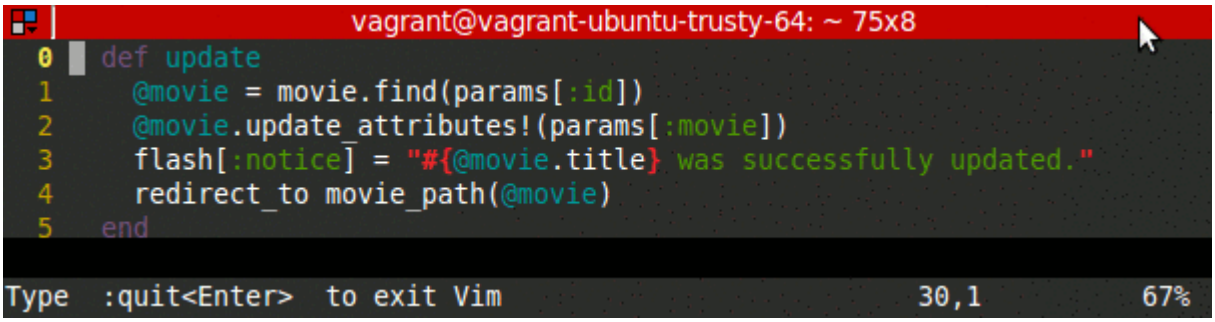
trouverait tous les nombres entiers et décimaux dans le fichier.

Naviguer jusqu'au début d'un mot spécifique

Lors de l'édition de texte, une tâche courante consiste à naviguer jusqu'à un mot particulier à l'écran. Dans ces exemples, nous explorons comment nous pouvons accéder au mot `updated` . Dans un souci de cohérence entre les exemples, nous visons la première lettre du mot.

Saut en plein écran

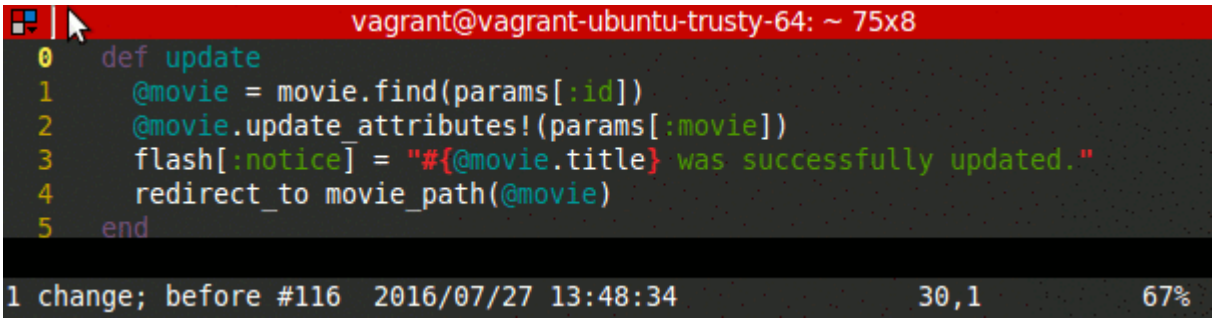
`M $ B`



Cette approche est rapide, en utilisant seulement 3 touches. L'inconvénient est que ce n'est pas très général, car il n'est pas courant que notre ligne cible se trouve au milieu de l'écran. Cependant, c'est un mouvement utile lorsque vous faites des mouvements moins granulaires.

Utiliser un compte

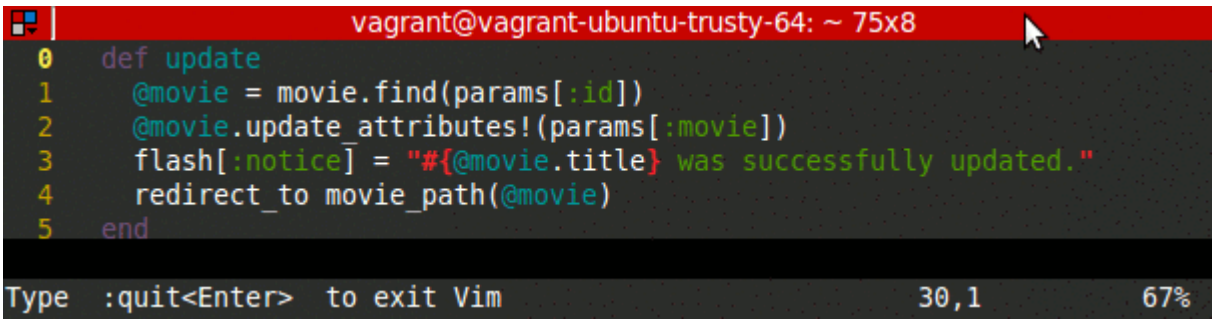
`3 j f u ; ;`



À première vue, cela peut sembler être un pas en arrière par rapport à la première approche en raison du nombre de frappes. Mais comme nous utilisons un décompte ici au lieu de `M` , il est plus flexible. Nous pouvons rapidement identifier le nombre correct à utiliser si relativenumber est activé. Pour passer au mot cible, utilisez `f` en combinaison avec `;` peut être étonnamment efficace - et certainement mieux que d'appuyer plusieurs fois sur `w` . Si vous dépassez votre cible avec `;` , Vous pouvez revenir en arrière avec.

Recherche explicite

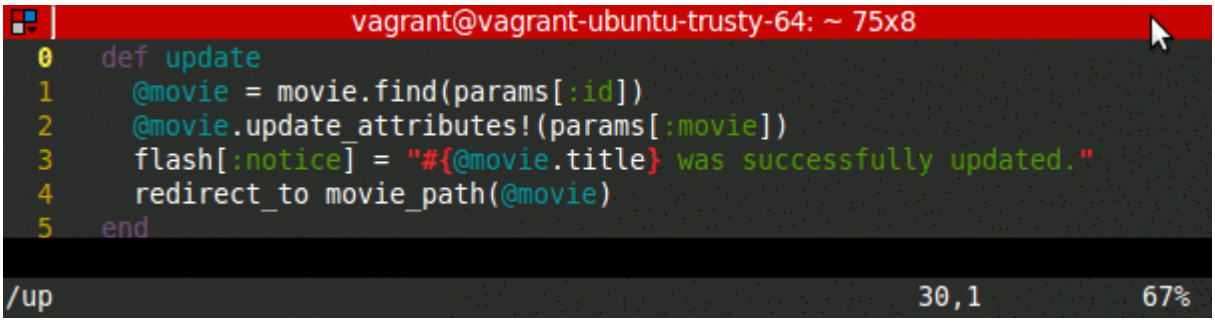
`/ up Enter n n`



Naviguer via `/` peut être très puissant. Nous pouvons souvent accéder directement à notre mot cible en le tapant. Ici, nous ne tapons que les deux premiers caractères dans l'espoir qu'il corresponde uniquement à notre mot. Malheureusement, il y a plusieurs matchs, mais nous pouvons rapidement passer au match suivant avec `n` .

Recherche implicite

`/ y Espace Entrez w`



Dans certains cas, il peut être plus efficace de sauter *près de* notre cible plutôt que de viser directement. Ici, nous observons qu'il y a une lettre peu fréquente, `y` , juste à côté de la cible. Nous pouvons ajouter un `espace` à notre terme de recherche pour diminuer les chances que nous avons

touché un autre `y` caractère le long du chemin. Cela peut aussi être très utile avec `f {char}` , comme dans l'exemple *Utilisation d'un décompte* .

Utiliser les marques pour se déplacer

Les marques sont comme des signets; ils vous aident à trouver les endroits où vous êtes déjà allé.

TLDR

Définissez-les en mode normal avec `m{a-zA-Z}` et `m{a-zA-Z}` les en mode normal ou visuel avec `'{a-zA-Z}` (guillemet simple) ou `` `{a-zA-Z} `` (backtick). Les lettres minuscules sont destinées aux marques dans un tampon et les lettres majuscules et les chiffres sont globaux. Voir vos marques actuellement définies avec `:marks` , et pour plus d'informations, voir `:help mark` .

Définir une marque

L'aide intégrée de Vim dit:

```
m{a-zA-Z}               Set mark {a-zA-Z} at cursor position (does not move
                        the cursor, this is not a motion command).
```

La marque gardera une trace de la ligne et de la colonne où elle a été placée. Il n'y a aucune confirmation visuelle qu'une marque a été définie ou si une marque avait une valeur antérieure et a été écrasée.

Aller à une marque

L'aide intégrée de Vim dit:

```
Jumping to a mark can be done in two ways:
1. With ` (backtick):     The cursor is positioned at the specified location
                          and the motion is exclusive.
2. With ' (single quote): The cursor is positioned on the first non-blank
                          character in the line of the specified location and
                          the motion is linewise.
```

Backtick utilise la position de la colonne, alors que Single-quote ne le fait pas. La différence entre simplement vous permet d'ignorer la position de la colonne de votre marque si vous le souhaitez.

Vous pouvez sauter entre les marques non globales en mode visuel en plus du mode normal pour permettre la sélection du texte en fonction des repères.

Marques mondiales

Les marques globales (majuscules) permettent de passer d'un fichier à un autre. Cela signifie que si, par exemple, la marque `A` est définie dans `foo.txt` , puis à partir de `bar.txt` (n'importe où dans mon système de fichiers), si je passe `bar.txt` à la marque `A` , mon tampon actuel sera remplacé par `foo.txt` . Vim vous invitera à enregistrer les modifications.

Passer à une marque dans un autre fichier n'est **pas** considéré comme un mouvement, et les sélections visuelles (entre autres choses) ne fonctionneront pas comme des sauts de marques dans un tampon.

Pour revenir au fichier précédent (`bar.txt` dans ce cas), utilisez `:b[uffer] #` (c'est-à-dire `:b#` ou `:buffer#`).

Remarque:

Marques spéciales

Vim définit automatiquement certaines marques (que vous pouvez écraser vous-même, mais que vous n'aurez probablement pas besoin).

Par exemple (paraphrasée de l'aide de Vim):

```
`[` and `]`: jump to the first or last character of the previously changed or
             yanked text.  {not in Vi}

`<` and `>`: jump to the first or last line (with `'`) or character (with
             <code>`</code>) of the last selected Visual area in the current
             buffer.  For block mode it may also be the last character in the
             first line (to be able to define the block).  {not in Vi}.
```

Plus, de l'aide intégrée de Vim:

```
''  ``              To the position before the latest jump, or where the
                    last "m'" or "m`" command was given.  Not set when the
                    :keepjumps command modifier was used.
                    Also see restore-position.


'"  `"              To the cursor position when last exiting the current
                    buffer.  Defaults to the first character of the first
                    line.  See last-position-jump for how to use this
                    for each opened file.
                    Only one position is remembered per buffer, not one
                    for each window.  As long as the buffer is visible in
                    a window the position won't be changed.
                    {not in Vi}.

'.  `.              To the position where the last change was made.  The
                    position is at or near where the change started.
                    Sometimes a command is executed as several changes,
                    then the position can be near the end of what the
                    command changed.  For example when inserting a word,
                    the position will be on the last character.
                    {not in Vi}
```

```
'"  `"              To the cursor position when last exiting the current
                    buffer.  Defaults to the first character of the first
                    line.  See last-position-jump for how to use this
                    for each opened file.
                    Only one position is remembered per buffer, not one
                    for each window.  As long as the buffer is visible in
                    a window the position won't be changed.
                    {not in Vi}.

'^  `^              To the position where the cursor was the last time
                    when Insert mode was stopped.  This is used by the
                    gi command.  Not set when the :keepjumps command
                    modifier was used.  {not in Vi}
```

De plus, les caractères `(` , `)` , `{` et `}` sont des marques qui sautent à la même position que leurs commandes en mode normal - c'est-à-dire que `'}` fait la même chose en mode normal que `}` .

Aller à une ligne spécifique

Pour sauter à une ligne spécifique avec un nombre deux-points. Passer à la première ligne d'un fichier utiliser

```
:1
```

Passer à la ligne 23

```
:23
```

Lire Mouvement en ligne: https://riptutorial.com/fr/vim/topic/1117/mouvement

Chapitre 33: Objets texte Vim

Examples

Sélectionnez un mot sans entourer les espaces blancs

Supposons que nous voulions sélectionner un mot sans espaces blancs environnants, utilisez l'objet texte `iw` pour le mot interne en utilisant le mode visuel:

1. Entré en mode normal en appuyant sur `ESC`
2. Tapez `viw` au début d'un mot
3. Cela sélectionnera le mot intérieur

Sélectionnez un mot avec un espace blanc environnant

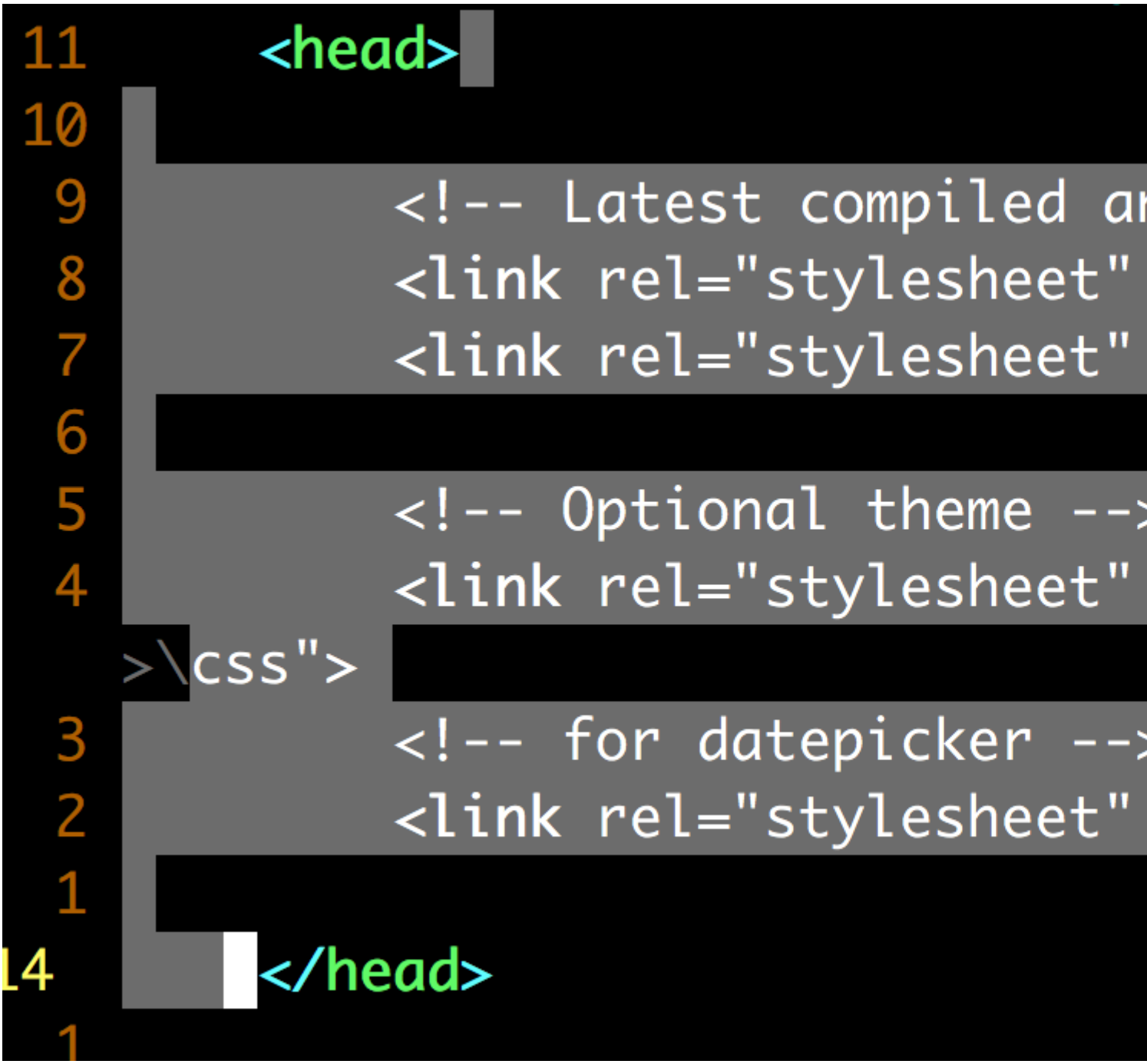
Supposons que nous voulions sélectionner un mot avec un espace blanc environnant, utilisez l'objet texte `aw` pour un mot en utilisant le mode visuel:

1. Entré en mode normal en appuyant sur ESC
2. Type `vaw` au début d'un mot
3. Cela sélectionnera le mot avec des espaces blancs

Sélectionner du texte dans une balise

Nous pouvons sélectionner un texte dans une balise `html` ou `xml` en utilisant la sélection visuelle `v` et l'objet texte `it` .

1. Passer au mode normal en appuyant sur `ESC`
2. Tapez `vit` de n'importe où dans la section `html` ou `xml`
3. Cela sélectionnera visuellement tout le texte dans la `tag`



Tous les autres objets texte peuvent également être utilisés pour agir sur le texte à l'intérieur de la balise

1. `cit` - supprime le texte dans la balise et le place en mode `insert`
2. `dit` - efface le texte dans la balise et reste en mode `normal`
3. `cat` - supprime le tag et le place en mode `insert`
4. `dat` - efface le texte autour de l'étiquette et reste en mode `normal`

Lire Objets texte Vim en ligne: https://riptutorial.com/fr/vim/topic/4050/objets-texte-vim

Chapitre 34: Obtenir: aide (en utilisant le manuel intégré de Vim)

Introduction

Le manuel intégré de Vim est la source d'informations et de documentation faisant autorité sur toutes les fonctionnalités de Vim, y compris les configurations, les fonctions intégrées et même Vimscript. Bien que ce ne soit pas l'interface la plus conviviale pour les débutants, si vous savez comment la parcourir, vous pouvez trouver ce dont vous avez besoin.

Lancez la recherche en exécutant `:help` `:help [subject]` , ou `:help :help` .

Syntaxe

- `:h[elp] [keyword]`

Paramètres

Paramètres	Détails
`keyword`	Configuration, nom de la fonction ou tout autre mot clé ayant une signification pour Vim. Mots-clés avec un deux `:` points `:` rechercher les commandes Vim; Exemple `:help :split` génère la commande de fractionnement de fenêtre et `:help split` génère la fonction Vimscript `split()` .

Examples

Mise en route / Navigation dans les fichiers d'aide

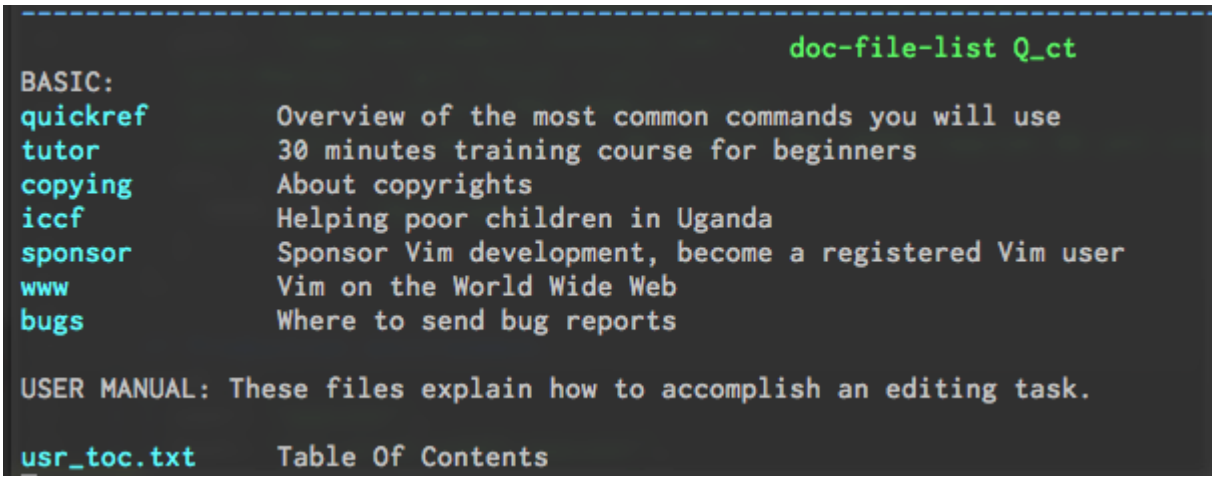
De n'importe où dans Vim, exécutez `:help :help` . Cela ouvrira une fenêtre divisée horizontalement avec la page de manuel de la commande `:help` . `:help` seule vous amènera à la table des matières du manuel lui-même.

Les fichiers d'aide de Vim sont navigables comme des fichiers normaux (vous pouvez rechercher des mots-clés dans un fichier comme d'habitude, avec `/`), et ils sont liés entre eux par des balises. Aller à la destination d'une balise avec `CTRL-]` .

Les tags sont des mots entourés de pipe `|` personnages. Les versions 7.3 et suivantes «cachent» ces personnages (`:help conceal`) et les mettent en évidence.

Par exemple, la page Table des matières affiche les éléments suivants. Tous les mots surlignés en bleu sont des balises et sont entourés de caractères en forme de tuyau. Taper `CTRL-]` avec le curseur sur `quickref` amènera à une page utile avec une liste de balises aux fonctionnalités utiles

de Vim.



Recherche dans le manuel

`:help [subject]` tente de trouver la "meilleure" correspondance pour l'argument que vous fournissez. L'argument "peut inclure des caractères génériques comme `* ?` Et `[az]` (n'importe quelle lettre).

Vous pouvez également utiliser l'achèvement de la ligne de commande de Vim avec `CTRL+D` : `:help spli<Ctrl-D>` affichera une liste de rubriques d'aide correspondant aux `spli` du modèle, y compris `split()` , et `:split` .

Pour rechercher des commandes basées sur `Ctrl` , telles que `Ctrl-V` , tapez:
`:help ^V` avec un caractère caret littéral, ou plus précisément,
`:help i_^V` à obtenir de l'aide sur `Ctrl-V` en mode insertion.

Comme vous le voyez, vim a une nomenclature pour ses rubriques d'aide. Par exemple, les options sont indiqués (voir `:h 'sw'`), les commandes commencent par deux points (voir `:h :split`), fonctions se terminent par des crochets vides (voir `:h split()`), les mappages du mode d'insertion commencent par `i_` , commande Les mappages de mode commencent par `c_` , et ainsi de suite, à l'exception des mappages en mode normal sans préfixe.

Terme de recherche	Page d'aide
`:help textwidth`	Configuration pour longueur de ligne / largeur de texte
`:help normal`	`:normal` commande `:normal` , pour exécuter des commandes en mode normal à partir de la ligne de commande
`:help cursor`	Commande Vimscript pour déplacer le curseur
`:help buffers`	Travailler avec des tampons; identique à `:help windows`
`:help :buffer`	La commande `:buffer`
`:help :vs`	Fractionnement vertical

Lire Obtenir: aide (en utilisant le manuel intégré de Vim) en ligne: https://riptutorial.com/fr/vim/topic/8837/obtenir--aide--en-utilisant-le-manuel-integre-de-vim-

Chapitre 35: Œufs de Pâques

Examples

Aidez-moi!

Pour l'utilisateur en détresse, vim fournit des mots de sagesse.

```
:help!
```

Quand tu te sens mal

Problème: les utilisateurs de Vim ne sont pas toujours contents.

Solution: Rendez-les heureux.

7,4

```
:smile
```

Remarque: Nécessite la version de correctif ≥ 7.4.1005

La réponse

Vim connaît la réponse:

```
:help 42
```

Vim ouvrira le document `usr_42.txt` partir du manuel et affichera le texte suivant:

> Quel est le sens de la vie, de l'univers et de tout? **42**
>
> Douglas Adams, la seule personne qui connaissait vraiment la question, est morte malheureusement. Alors maintenant, vous pourriez vous demander quel est le sens de la mort est ...

À la recherche du Saint Graal

Regarde ça:

```
:help holy-grail
```

Ceci n'est pas une pipe

Si vous cherchez la section d'aide de `|` ou `bar` :: `:h bar` vous pouvez voir:

```
                                                        bar
```

```
|                       To screen column [count] in the current line.
                        exclusive motion.  Ceci n'est pas une pipe.
```

Ceci est une référence à la peinture *La trahison des images* de René Magritte.

Lorsqu'un utilisateur s'ennuie

Rechercher `:h UserGettingBored`

```
                        *UserGettingBored*
UserGettingBored        When the user presses the same key 42 times.
                        Just kidding! :-)
```

Cuillère

Au lieu de chercher l'aide de la `fork` , vous pouvez rechercher l'aide de la `spoon` :

```
:h spoon

                                                        fork spoon
For executing external commands fork()/exec() is used when possible, otherwise
system() is used, which is a bit slower.  The output of ":version" includes ...
```

Chevaliers qui disent Ni!

Regarde ça:

```
:Ni!
```

Monty Python et le Saint Graal

nunmap

```
:help map-modes
```

```
:nunmap can also be used outside of a monastery.
```

Lire Œufs de Pâques en ligne: https://riptutorial.com/fr/vim/topic/4656/oufs-de-paques

Chapitre 36: Options Vim

Syntaxe

- `:set [no](option|shortcut)`
- `:set (option|shortcut)=value`
- `:set (option|shortcut)(?|&)`
- ne pas utiliser : dans le fichier vimrc

Remarques

Voir vimcast 1 vidéo

Voir vimcast 1 transcription

Examples

Ensemble

Pour définir les options, utilisez `:set` instruction. Exemple:

```
:set ts=4
:set shiftwidth=4
:set expandtab
:set autoindent
```

Pour afficher la valeur actuelle de l'option - tapez `:set {option}?` . Exemple:

```
:set ts?
```

Pour réinitialiser la valeur de l'option à sa valeur par défaut - tapez `:set {option}&` . Exemple:

```
:set ts&
```

Échancrure

Largeur

Pour faire des empreintes de 4 espaces:

```
:set shiftwidth=4
```

Les espaces

Pour utiliser des espaces en retrait, 4 espaces de largeur:

```
:set expandtab
:set softtabstop=4
```

`softtabstop` et `sts` sont équivalents:

```
:set sts=4
```

Onglets

Pour utiliser des tabulations en retrait, 4 espaces de largeur:

```
:set noexpandtab
:set tabstop=4
```

`tabstop` et `ts` sont équivalents:

```
:set ts=4
```

Indentation Automatique

```
:set autoindent
```

Descriptions d'instruction

Instruction	La description	Défaut
tabstop	largeur du caractère de tabulation	8
expandtab	provoque l'utilisation des espaces à la place du caractère de tabulation	de
softabstop	accorder les espaces	0
shiftwidth	détermine le nombre d'espaces en mode normal	8

Personnages invisibles

Afficher ou masquer les caractères invisibles

Pour afficher les caractères invisibles:

```
:set list
```

Pour masquer les caractères invisibles:

```
:set nolist
```

Pour basculer entre l'affichage et le masquage des caractères invisibles:

```
:set list!
```

Caractères de symbole par défaut

symbole	Personnage
^ Je	Languette
$	Nouvelle ligne

Personnaliser les symboles

Pour définir le caractère de tabulation sur **> ** et le nouveau caractère de ligne sur ¬

```
set listchars=tab:>\ ,eol:¬
```

Pour définir les espaces à _

```
set listchars=spaces
```

Pour voir une liste d'options de caractères

```
:help listchars
```

Lire Options Vim en ligne: https://riptutorial.com/fr/vim/topic/2407/options-vim

Chapitre 37: Pliant

Remarques

Le pliage entraîne la *réduction de* plusieurs lignes de texte et leur affichage sur une seule ligne. Il est utile pour masquer des parties d'un document considérées comme sans importance pour la tâche en cours. Le pliage est purement un changement visuel du document: les lignes pliées sont toujours présentes, inchangées.

Un pli est persistant. Une fois créé, un pli peut être ouvert et fermé sans avoir à le recréer. Lorsqu'ils sont fermés, les plis peuvent être déplacés et placés comme s'ils étaient une seule ligne, même si l'opération sous-jacente opère sur tout le texte sous le pli.

Examples

Configuration de la méthode de pliage

`:set foldmethod={method}` définit la méthode de pliage pour la fenêtre en cours. Cela détermine comment les plis sont manipulés dans cette fenêtre. Les options valides pour "méthode" sont:

- `manual` (les plis sont créés manuellement et détruits par l'utilisateur)
- `indent` (les plis sont créés pour les lignes d'égale indentation)
- `marker` (les marqueurs de sous-chaîne sont utilisés pour indiquer le début et la fin d'un pli)
- `syntax` (les éléments de mise en évidence de la syntaxe définissent les plis)
- `expr` (une expression Vimscript est évaluée par ligne pour définir son niveau de pliage)
- `diff` (le changement de texte n'est pas modifié dans un affichage différentiel est plié)

La valeur par défaut est `manual` .

Créer un pli manuellement

- `zf{motion}` crée un pli qui couvre le texte que "motion" couvrirait.
- `{count}zF` crée un pli qui couvre les lignes "count".
- `{range}fo[ld]` crée un pli pour les lignes de la plage fournie.

Les trois commandes ne sont valides que lorsque `foldmethod` est défini sur `manual` ou `marker` . Dans le cas de l'ancienne méthode de pliage, les nouveaux plis sont immédiatement fermés.

Pli d'ouverture, de fermeture et de basculement

- `zo` ouvre un pli sous le curseur.
- `zO` ouvre tous les plis sous le curseur, récursivement.
- `zc` ferme un pli sous le curseur.
- `zC` ferme tous les plis sous le curseur, récursivement.
- `za` bascule un pli sous le curseur (un pli fermé est ouvert, un pli ouvert est fermé).

- `zM` ferme tous les plis du tampon.
- `zR` ouvre tous les plis dans le tampon.
- `zm` ferme un niveau de pli dans le tampon.
- `zr` ouvre un niveau de pli dans le tampon.

Afficher la ligne contenant le curseur

`zv` assurera que la ligne contenant le curseur n'est pas pliée. Le nombre minimum de plis requis pour exposer la ligne du curseur sera ouvert.

Blocs de pliage en C

Ceci est notre tampon:

```
void write_buffer(size_t size, char ** buffer)
{
    char * buf = *buffer;
    size_t iter;
    for(iter = 0; iter < size; iter++)
    {
        putc(*(buf + iter));
    }
}
```

Le curseur est à [1] [1] ([ligne] [col]). Déplacez le curseur sur le crochet de la boucle for: `/for<Enter>j` curseur est [6] [2].

Entrez maintenant `zf%` (créez un pliage, déplacez-vous vers le crochet correspondant). Vous avez réussi à créer le premier pliage.

Maintenant, entrez `:2<Enter>_` , le curseur est maintenant à [2] [1] et `zf%` : le corps complet de la fonction est plié.

Vous pouvez ouvrir tous les plis que vous venez de créer avec `zO` et les refermer à l'aide de `zC` .

Lire Pliant en ligne: https://riptutorial.com/fr/vim/topic/3791/pliant

Chapitre 38: Plugins

Examples

Fugitive Vim

Fugitive Vim est un plugin de Tim Pope qui permet d'accéder aux commandes git que vous pouvez exécuter sans quitter vim.

Certaines commandes courantes incluent:

`:Gedit` - éditer un fichier dans l'index et l'écrire pour mettre en scène les modifications
`:Gstatus` - équivalent du `git status`
`:Gblame` - fait apparaître une division verticale de la sortie de `git blame`
`:Gmove` - pour git mv
`:Gremove` - pour git rm
`:Git` - lance n'importe quelle commande

Il ajoute également des éléments à la `statusline` comme pour indiquer la branche en cours.

S'il vous plaît voir leur GitHub pour plus de détails et les instructions d'installation.

Arbre NERD

NERD TREE est un plugin de scrooloose qui vous permet d'explorer le système de fichiers en utilisant vim. Vous pouvez ouvrir des fichiers et des répertoires via un arbre que vous pouvez manipuler avec le clavier ou la souris.

Ajoutez ceci à votre .vimrc pour démarrer NERDTree automatiquement au démarrage de vim:

```
autocmd vimenter * NERDTree
```

Pour fermer automatiquement NERDTree si c'est la seule fenêtre à gauche, ajoutez ceci à votre fichier .vimrc:

```
autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTree") && b:NERDTree.isTabTree()) | q
| endif
```

Il est recommandé de mapper une combinaison de touches avec la commande NERDTreeToggle. Ajoutez ceci à votre .vimrc (cet exemple utilise Ctrl + N)

```
map <C-n> :NERDTreeToggle<CR>
```

Les détails complets et les instructions d'installation peuvent être visualisés sur leur Github .

Lire Plugins en ligne: https://riptutorial.com/fr/vim/topic/9976/plugins

Chapitre 39: Plugins de type de fichier

Examples

Où placer des plugins de type de fichier personnalisés?

Les plugins de type de fichier pour `foo` filetype proviennent de cet ordre:

1. `$HOME/.vim/ftplugin/foo.vim` . Soyez prudent avec ce que vous avez mis dans ce fichier car il peut être remplacé par `$VIMRUNTIME/ftplugin/foo.vim` - sous windows, ce sera plutôt `$HOME/vimfiles/ftplugin/foo.vim`

2. `$VIMRUNTIME/ftplugin/foo.vim` . Comme tout ce qui est sous `$VIMRUNTIME` , ce fichier ne doit pas être modifié.

3. `$HOME/.vim/after/ftplugin/foo.vim` . Ce fichier est fourni en dernier, ce qui en fait l'endroit idéal pour *vos* paramètres spécifiques à un type de fichier.

Lire Plugins de type de fichier en ligne: https://riptutorial.com/fr/vim/topic/7734/plugins-de-type-de-fichier

Chapitre 40: Recherche dans le tampon actuel

Examples

Recherche d'un motif arbitraire

Les commandes de recherche standard de Vim sont `/` pour la recherche directe et `?` pour la recherche en arrière.

Pour lancer une recherche à partir du mode normal:

1. appuyez sur `/` ,
2. tapez votre motif,
3. appuyez sur `<CR>` pour effectuer la recherche.

Exemples:

```
/foobar<CR>      search forward for foobar
?foo\/bar<CR>    search backward for foo/bar
```

`n` et `N` peuvent être utilisés pour passer à l'occurrence suivante et précédente:

- Si vous appuyez sur `n` après une recherche avancée, le curseur se positionne sur la prochaine occurrence, en *avant* .
- Si vous appuyez sur `N` après une recherche avancée, le curseur se positionne sur la prochaine occurrence, *vers l'arrière* .
- En appuyant sur `n` après une recherche en arrière, le curseur se positionne sur la prochaine occurrence, en *arrière* .
- Appuyez sur `N` après une recherche en arrière pour positionner le curseur sur la prochaine occurrence, en *avant* .

Recherche du mot sous le curseur

En mode normal, déplacez le curseur sur n'importe quel mot, puis appuyez sur `*` pour rechercher la prochaine occurrence du mot sous le curseur ou appuyez sur `#` pour rechercher en arrière.

`*` ou `#` recherche le mot exact sous le curseur: la recherche de `big` trouverait seulement `big` et pas `bigger` .

Sous le capot, Vim utilise une recherche simple avec des atomes de *frontières de mots* :

- `/\<big\>` for `*` ,
- `?\<big\>`

pour `#` .

`g*` ou `g#` ne recherche pas le mot exact sous le curseur: la recherche de `big` trouverait `bigger` .

Sous le capot, Vim utilise une recherche simple sans atomes *limitant les mots* :

- `/\<big\>` for `*` ,
- `?\<big\>` pour `#` .

exécuter la commande sur les lignes contenant du texte

La commande `:global` a déjà son propre sujet: La commande globale

Lire Recherche dans le tampon actuel en ligne: https://riptutorial.com/fr/vim/topic/3269/recherche-dans-le-tampon-actuel

Chapitre 41: Registres Vim

Paramètres

Fonctionnalité	Registres
registre par défaut	`""`
registres d'histoire	`"[1-9]`
yank registre	`"0`
les registres nommés	`"[az]` , `"[AZ]` identique à `"[az]` mais annexe
rappeler le motif de recherche actuel	`"/`
petites suppressions (diw, cit, ...)	`"-`
registres d'expression pour les mathématiques simples	`"=`
trou noir enregistrer pour éliminer les gros morceaux de texte supprimé de mem	`"_`
dernière commande	`":`
dernier texte inséré	`".`
nom de fichier	`"%`
presse-papiers	`"*`
texte sélectionné	`"+`
texte abandonné	`"~`

Examples

Supprimer une plage de lignes dans un registre nommé

En mode Normal, tapez ce qui suit pour supprimer une plage de lignes dans un registre nommé

```
:10,20d a
```

Cela supprimera les lignes 10,20 dans le registre `"a` . Nous pouvons le vérifier en tapant

```
:reg
```

Cela affichera le texte qui a été supprimé dans le registre `"a` .

Pour coller le contenu dans `"a` , tapez simplement

```
"ap
```

Collez le nom de fichier en mode insertion en utilisant le nom de fichier register

En mode Insertion, appuyez sur `<Cr>` puis sur `%` pour insérer le nom du fichier.

Cette technique est applicable à tous les registres.

Par exemple, si vous êtes en mode insertion, vous souhaitez coller le modèle de recherche en cours, vous pouvez taper `<Cr>` , puis `/` .

Copier / coller entre Vim et le presse-papiers du système

Utiliser le registre quotestar pour copier / coller entre Vim et le presse-papiers du système

`"*yy` copie la ligne en cours dans le presse-papier du système

`"*p` colle le contenu du presse-papiers du système dans Vim

Ajouter à un registre

Yank toutes les lignes contenant TODO dans un registre en utilisant l'opération d'ajout

```
:global/TODO/yank A
```

Ici, nous recherchons un mot-clé `TODO` globalement, en tirant toutes les lignes dans le registre `a` (`A` registre ajoute toutes les lignes à `a` registre).

REMARQUE: Il est généralement recommandé d'effacer un registre avant d'effectuer l'opération d'ajout.

Pour effacer un registre, en mode normal, tapez `qaq` . Vérifiez que le `a` registre est vide en tapant `:reg` et en observant que `a` registre est vide.

Lire Registres Vim en ligne: https://riptutorial.com/fr/vim/topic/4278/registres-vim

Chapitre 42: Ressources Vim

Remarques

Ce sujet concerne les **miroirs de code source** , les **livres** , les **vim-wikis** . Il ne s'agit **pas** de blogs, de Wikipedia, de didacticiels. Les ressources ne doivent pas être basées sur des opinions.

Examples

Apprendre le Vimscript à la dure

Un livre expliquant comment fonctionne Vimscript, plein d'exemples. On peut le trouver sur http://learnvimscriptthehardway.stevelosh.com/

Lire Ressources Vim en ligne: https://riptutorial.com/fr/vim/topic/6383/ressources-vim

Chapitre 43: Sortie de Vim

Paramètres

Paramètre	Détails
`:`	Entrer en mode ligne de commande
`w`	Écrire
`q`	Quitter
`a`	Tout
`!`	Passer outre

Remarques

Le mode ligne de commande est entré en mode normal. Vous saurez que vous êtes en mode ligne de commande lorsqu'il y a un `:` dans le coin inférieur gauche de la fenêtre de votre terminal.

Le mode normal est le mode par défaut de vi / vim et peut être commuté en appuyant sur la `touche ESC` .

Vi / Vim dispose de vérifications intégrées pour empêcher toute perte de travail non enregistré et d'autres fonctionnalités utiles. Pour contourner ces vérifications, utilisez la substitution `!` dans votre commande.

Dans Vi / Vim, il est possible d'afficher plusieurs fichiers simultanément (dans différentes fenêtres). Utilisez `a` pour fermer toutes les fenêtres ouvertes.

Examples

Quitter avec save

```
: wq
```

```
ZZ
```

Quitter sans enregistrer

```
: q!
```

Quitter avec force (sans enregistrer)

```
: q!
```

```
ZQ
```

Quitter avec force (avec save)

```
: wq!
```

Quittez avec force toutes les fenêtres ouvertes (sans enregistrer)

```
: qa!
```

si plusieurs fichiers sont ouverts

```
:wqall
```

Sortie de plusieurs fichiers avec sauvegarde du contenu

```
:qall!
```

Quitter plusieurs fichiers sans enregistrer le contenu

Lire Sortie de Vim en ligne: https://riptutorial.com/fr/vim/topic/5074/sortie-de-vim

Chapitre 44: Substitution

Syntaxe

- `s/<pattern>/<pattern>/optional-flags`
- `<pattern>` est une expression régulière

Paramètres

Drapeau	Sens
Et	Gardez les drapeaux du remplaçant précédent.
c	Invite à confirmer chaque substitution.
e	Ne pas signaler les erreurs.
g	Remplacez toutes les occurrences de la ligne.
je	Correspondance insensible à la casse.
je	Correspondance sensible à la casse.
n	Indiquez le nombre de correspondances, ne le remplacez pas réellement.

Remarques

Utilisez `set gdefault` pour ne pas avoir à spécifier le drapeau 'g' sur chaque substitut.

Exemple

Lorsque `gdefault` est défini, exécutez `:s/foo/bar` sur la ligne `foo baz foo` donnera `bar baz bar` au lieu de `bar baz foo` .

Examples

Remplacement simple

`:s/foo/bar` Remplace la **première** instance de *foo* par la *barre* sur la ligne en cours.

`:s/foo/bar/g` Remplace chaque instance de *toto* par une *barre* sur la ligne en cours.

`:%s/foo/bar/g` Remplacez *foo* par *bar* dans tout le fichier.

Réorganiser rapidement le mot sous le curseur

1. `*` sur le mot que vous voulez substituer.
2. `:%s//replacement/g` , laissant le modèle de *recherche* vide.

Remplacement avec approbation interactive

`:s/foo/bar/c` Marque la première instance de *foo* sur la ligne et demande confirmation de la substitution par une *barre*

`:%s/foo/bar/gc` Marque chaque correspondance de *foo* dans le fichier et demande confirmation de la substitution par une *barre*

Raccourci clavier pour remplacer le mot surligné

Par exemple, avec `nmap` suivant:

```
nmap <expr> <S-F6> ':%s/' . @/ . '//gc<LEFT><LEFT><LEFT>'
```

sélectionnez un mot avec `*` , tapez `Shift` - `F6` , tapez un remplacement et appuyez sur `Entrée` pour renommer toutes les occurrences de manière interactive.

Lire Substitution en ligne: https://riptutorial.com/fr/vim/topic/3384/substitution

Chapitre 45: Tampons

Examples

Gestion des tampons

Vous pouvez utiliser des tampons pour travailler avec plusieurs fichiers. Lorsque vous ouvrez un fichier en utilisant

```
:e path/to/file
```

il s'ouvre dans un nouveau tampon (la commande signifie éditer le fichier). Nouveau tampon contenant une copie temporaire du fichier.

Vous pouvez aller au tampon précédent avec `:bp[rev]` et le tampon suivant avec `:bn[ext]` .

Vous pouvez accéder à un tampon particulier avec `b{n}` pour accéder au n-ième tampon. `b2` va au deuxième tampon.

Utilisez `:ls` ou `:buffers` pour lister tous les tampons

Tampons cachés

S'éloigner d'un tampon avec des modifications non enregistrées provoquera cette erreur:

```
E37: No write since last change (add ! to override)
```

Vous pouvez désactiver cela en ajoutant le `set hidden` à votre fichier .vimrc. Avec cette option, vos modifications persisteront dans le tampon, mais ne seront pas enregistrées sur le disque.

Changer de tampon en utilisant une partie du nom de fichier

Pour sélectionner facilement un tampon par nom de fichier, vous pouvez utiliser:

```
:b [part_of_filename]<Tab><Tab><Tab>...<Enter>
```

Le premier `onglet` étendra le mot à un nom de fichier complet, et les opérations de `tabulation` suivantes feront défiler la liste des correspondances possibles.

Lorsque plusieurs correspondances sont disponibles, vous pouvez voir une liste de correspondances *avant* l'extension du mot en définissant cette option:

```
:set wildmode=longest:full:list,full
```

Cela vous permet d'affiner votre mot si la liste des correspondances est trop longue, mais cela nécessite une pression supplémentaire sur `Tab` pour effectuer l'extension. Ajoutez le paramètre à

votre `$MYVIMRC` si vous souhaitez le conserver.

Certaines personnes aiment lancer ce processus avec une keymap qui répertorie d'abord les tampons:

```
:nnoremap <Leader>b :set nomore <Bar> :ls <Bar> :set more <CR>:b<Space>
```

Cela facilite la sélection d'un tampon par son numéro:

```
:b [buffer_num]
```

Passez rapidement au tampon précédent ou à un tampon quelconque par numéro

`<C-^>` basculera vers et depuis le fichier modifié précédent. Sur la plupart des claviers, `<C-^>` correspond à CTRL-6.

`3<C-^>` passera au numéro de tampon 3. Ceci est très rapide, mais seulement si vous connaissez le numéro de tampon.

Vous pouvez voir les numéros de tampon de `:ls` ou d'un plugin tel que MiniBufExplorer .

Lire Tampons en ligne: https://riptutorial.com/fr/vim/topic/2317/tampons

Chapitre 46: Trouver et remplacer

Examples

Commande de substitution

Cette commande:

```
:s/foo/bar/g
```

remplace chaque occurrence de `foo` par une `bar` sur la ligne en cours.

```
fool around with a foodie
```

devient

```
barl around with a bardie
```

Si vous omettez le dernier `/g` , il ne remplacera que la première occurrence sur la ligne. Par exemple,

```
:s/foo/bar
```

Sur la ligne précédente deviendrait

```
barl around with a foodie
```

Cette commande:

```
:5,10s/foo/bar/g
```

effectue la même substitution aux lignes 5 à 10.

Cette commande

```
 :5,$s/foo/bar/g
```

effectue la même substitution de la ligne 5 à la fin du fichier.

Cette commande:

```
:%s/foo/bar/g
```

effectue la même substitution sur l'ensemble du tampon.

Si vous êtes en mode visuel et que vous appuyez sur les deux points, le symbole `'<,'>` apparaîtra. Vous pouvez alors le faire

```
:'<,'>s/foo/bar/g
```

et faire la substitution dans votre sélection de mode visuel.

Cette commande:

```
:%s/foo/bar/gc
```

est équivalent à la commande ci-dessus mais demande confirmation sur chaque occurrence grâce au drapeau `/c` (pour "confirmation").

Voir `:help :s` et `:help :s_flags` .

Voir aussi cette section sur les plages de lignes de commande .

Remplacer avec ou sans expressions régulières

Cette commande de substitution peut utiliser des expressions régulières et correspondra à toute instance de `foo` *suivie de tout (un) caractère* depuis la période `.` dans les expressions régulières correspond à n'importe quel caractère, la commande suivante correspondra à toutes les instances de `foo` suivies de tout caractère de la ligne en cours.

```
:s/foo./bar/g
```

```
  1 fooing fooes fool foobar foosup
```

va devenir

```
  1 barng bars bar barar barup
```

Si vous voulez faire correspondre le littéral `.` période, vous pouvez y échapper dans le champ de recherche avec une barre oblique inverse `\` .

```
:s/foo\./bar/g
```

```
  1 fooing fooes foo.l foo.bar foosup
```

va devenir

```
  1 fooing fooes barl barbar foosup
```

Ou désactivez toutes les correspondances en suivant la commande `s` avec `no` .

```
:sno/foo./bar/g
```

```
1 fooing fooes foo.l foo.bar foosup
```

va soulever une erreur

```
E486: Pattern not found
```

Lire Trouver et remplacer en ligne: https://riptutorial.com/fr/vim/topic/3533/trouver-et-remplacer

Chapitre 47: Trucs et astuces pour augmenter la productivité

Syntaxe

- : set relativenumber
- : définir le numéro
- : set nonumber /: set nonu
- : pwd

Remarques

Ce rechargement automatique ne se produira que si vous éditez votre `vimrc` dans une version complète de vim qui supporte `autocmd` .

Examples

Macros rapides "jetables"

Ajoutez ceci à votre vimrc:

```
nnoremap Q @q
```

Pour commencer à enregistrer la macro "throwaway", utilisez `qq` . Pour terminer l'enregistrement, appuyez sur `q` (en mode normal pour les deux).

Pour exécuter la macro enregistrée, utilisez `Q`

Ceci est utile pour les macros que vous devez répéter plusieurs fois de suite mais qui ne seront probablement pas réutilisées par la suite.

Utiliser la fonction d'achèvement de chemin à l'intérieur de Vim

C'est très commun, vous mémorisez un chemin vers un fichier ou un dossier, vous ouvrez Vim et essayez d'écrire ce que vous venez de mémoriser, mais vous n'êtes pas sûr à 100% que vous avez raison, fermez l'éditeur et recommencez.

Lorsque vous souhaitez utiliser la fonctionnalité de complétion de chemin et que vous avez un fichier `/home/ubuntu/my_folder/my_file` et que vous modifiez un autre fichier faisant référence au chemin de la précédente:

Entrez le mode d'insertion: `insérez` ou faites comme vous le souhaitez. Ensuite, écrivez `/h` . Lorsque le curseur est sous `h` , appuyez sur `Ctrl x` , puis sur `Ctrl f` pour que l'éditeur l'achève vers `/home/` car le motif `/h` est unique

Maintenant, supposons que vous avez deux dossiers dans `/home/ubuntu/` appelé `my_folder_1` `my_folder_2`

et vous voulez le chemin `/home/ubuntu/my_folder_2`

comme d'habitude:

Entrer en mode insertion

écrivez `/h` et appuyez sur `Ctrl x` , puis sur `Ctrl f` . Maintenant, vous avez `/home/` Next ajoutez u après / home / et appuyez sur `Ctrl x` , puis sur `Ctrl f` . Maintenant, vous avez `/home/ubuntu/` parce que ce chemin est unique. Maintenant, écrivez `my /home/ubuntu/` et appuyez sur `Ctrl x` , puis sur `Ctrl f` . L'éditeur complétera votre mot jusqu'à `my_folder_` et vous verrez l'arborescence du répertoire. Utilisez les touches fléchées pour choisir celle que vous voulez.

Activer les numéros de ligne relatifs

Pour supprimer des lignes de texte lorsque vous ne connaissez pas le nombre exact de lignes à supprimer, essayez `10dd` , `5dd` , `3dd` jusqu'à ce que toutes les lignes soient supprimées.

Les numéros de ligne relatifs résolvent ce problème, supposons que nous ayons un fichier contenant:

```
sometimes, you see a block of
text. You want to remove
it but you
cannot directly get the
exact number of
lines to delete
so you try
10d , 5d
3d until
you
remove all the block.
```

Entrer en mode NORMAL: `Esc`

Maintenant, exécutez `:set relativenumber` . Une fois terminé, le fichier sera affiché comme suit:

```
3 sometimes, you see a block of
2 text. You want to remove
1 it but you
0 cannot directly get the
1 exact number of
2 lines to delete
3 so you try
4 10d , 5d
5 3d until
6 you
7 remove all the block.
```

où `0` est le numéro de ligne de la ligne en cours et indique également le numéro de ligne réel devant le numéro relatif, vous n'avez donc plus besoin d'estimer le nombre de lignes de la ligne en

cours pour les couper ou les supprimer par un.

Vous pouvez maintenant exécuter votre commande habituelle comme `6dd` et vous êtes sûr du nombre de lignes.

Vous pouvez également utiliser la forme abrégée de la même commande `:set rnu` pour activer les nombres relatifs et `:set nornu` pour désactiver les mêmes.

Si vous aussi `:set number` ou avez `:set number` déjà le `:set number` , vous obtiendrez le numéro de ligne de la ligne dans laquelle se trouve le curseur.

```
3 sometimes, you see a block of
2 text. You want to remove
1 it but you
4 cannot directly get the
1 exact number of
2 lines to delete
3 so you try
4 10d , 5d
5 3d until
6 you
7 remove all the block.
```

Affichage des numéros de ligne

Pour afficher les numéros de ligne de la vue par défaut, entrez

```
:set number
```

Pour masquer les numéros de ligne

```
:set nonumber
```

Il y a aussi un raccourci pour ci-dessus. `nu` est identique au `number` .

```
:set nonu
```

Pour masquer les numéros de ligne, nous pouvons également utiliser

```
:set nu!
```

Mappages pour quitter le mode Insertion

De nombreux utilisateurs de Vim trouvent la commande `Esc` trop difficile à atteindre et finissent par trouver une autre carte facile à atteindre depuis la ligne d'accueil. Notez que `Ctrl` - `[` peut être équivalent à `Esc` sur un clavier anglais et est beaucoup plus facile à atteindre.

---

j k

```
inoremap jk <ESC>
```

Celui-ci est vraiment facile à déclencher; Il suffit de fracasser vos deux premiers doigts sur la rangée d'accueil en même temps. Il est également difficile de se déclencher accidentellement car "jk" n'apparaît jamais dans aucun mot anglais, et si vous êtes en mode normal, il ne fait rien. Si vous ne tapez pas trop "blackjack", pensez à ajouter également `inoremap kj <ESC>` pour ne pas avoir à vous soucier de la synchronisation des deux clés.

verrouillage des majuscules

Linux

Sous Linux, vous pouvez utiliser `xmodmap` pour que `Caps Lock` fasse la même chose que `Esc` . Mettez ceci dans un fichier:

```
!! No clear Lock
clear lock
!! make caps lock an escape key
keycode 0x42 = Escape
```

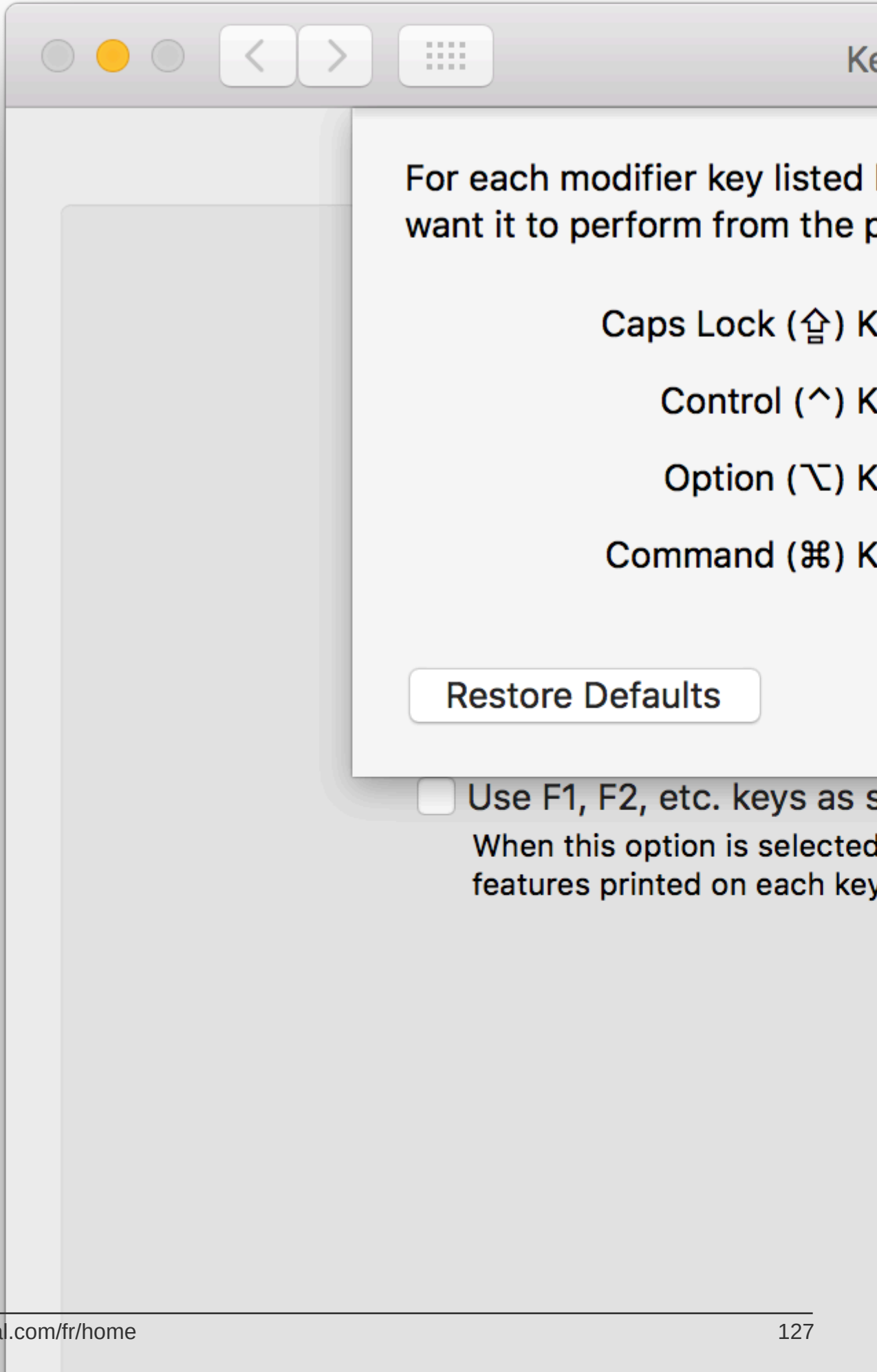
Ensuite, exécutez le `xmodmap file` . Cela remappe `Caps Lock` à `Esc` .

les fenêtres

Sous Windows, vous pouvez utiliser SharpKey ou AutoHotkey .

macOS

Si vous avez macOS 10.12.1 ou ultérieur, vous pouvez remapper Caps Lock sur Escape en utilisant les Préférences Système. Sélectionnez Clavier, accédez à l'onglet Clavier et cliquez sur Modifier les touches.


For each modifier key listed
want it to perform from the
Caps Lock (⇪) K
Control (^) K
Option (⌥) K
Command (⌘) K
Restore Defaults
Use F1, F2, etc. keys as
When this option is selected
features printed on each key

entourant le curseur, utilisez la commande suivante (`<ESC>` - touche d'échappement, `<CR>` - touche d'accès):

```
vi{<ESC>/\%Vfoo<CR>
```

maintenant vous pouvez sauter entre les matchs dans le bloc en appuyant sur `n` et `p` . Si l'option `hlsearch` activée, cela mettra en évidence toutes les correspondances. `\%V` est une partie de modèle de recherche spéciale, qui indique à vim de rechercher uniquement dans la zone sélectionnée visuellement. Vous pouvez également faire un mappage comme celui-ci:

```
:vnoremap g/ <ESC>/\%V
```

Après cela, la commande ci-dessus est raccourcie comme suit:

```
vi{g/foo<CR>
```

Une autre astuce utile consiste à imprimer toutes les lignes contenant le motif:

```
vi{
:'<,'>g/foo/#
```

La plage `'<,'>` est insérée automatiquement.

Voir `:help range` et `:help :g` .

Copier, déplacer ou supprimer une ligne trouvée

De nombreux utilisateurs se trouvent dans une situation où ils souhaitent simplement copier, déplacer ou supprimer une ligne rapidement et revenir à leur emplacement d'origine.

En général, si vous souhaitez déplacer une ligne contenant le `lot` mots sous la ligne en cours, vous devez taper quelque chose comme:

```
/lot<Esc>dd<C-o>p
```

Mais pour augmenter la productivité, vous pouvez utiliser ce raccourci dans les cas suivants:

```
" It's recommended to turn on incremental search when doing so
set incsearch

" copy the found line
cnoremap $t <CR>:t''<CR>
" move the found line
cnoremap $m <CR>:m''<CR>
" delete the found line
cnoremap $d <CR>:d<CR>``
```

Donc, une recherche comme celle-ci:

```
/lot$m
```

déplacerait la ligne contenant le `lot` sous la ligne sur laquelle se trouvait le curseur lorsque vous avez lancé la recherche.

Ecrivez un fichier si vous oubliez `sudo` avant de lancer vim

Cette commande enregistre le fichier ouvert avec les droits sudo

```
:w !sudo tee % >/dev/null
```

Vous pouvez également cartographier `w!!` pour écrire un fichier en tant que root

```
:cnoremap w!! w !sudo tee % >/dev/null
```

Recharger automatiquement vimrc lors de la sauvegarde

Pour recharger automatiquement `vimrc` lors de la sauvegarde, ajoutez ce qui suit à votre `vimrc` :

```
if has('autocmd') " ignore this section if your vim does not support autocommands
    augroup reload_vimrc
        autocmd!
        autocmd! BufWritePost $MYVIMRC,$MYGVIMRC nested source %
    augroup END
endif
```

et puis pour la dernière fois, tapez:

```
:so $MYVIMRC
```

La prochaine fois que vous sauvegarderez votre `vimrc` , il sera automatiquement rechargé.

`nested` est utile si vous utilisez vim-airline. Le processus de chargement de la compagnie aérienne déclenche des autocommandes, mais comme vous êtes en train d'exécuter une autocommande, elles sont ignorées. `nested` permet de déclencher des autocommandes imbriquées et permet aux compagnies aériennes de se charger correctement.

Achèvement de la ligne de commande

définir le `wildmenu` pour activer les suggestions d'achèvement pour la ligne de commande. Exécuter le suivant

```
set wildmenu
set wildmode=list:longest,full
```

Maintenant, si vous dites, `onglet de :color` ,

Tu auras

```
256-jungle  Benokai  BlackSea  C64  CandyPaper  Chasing_Logic  ChocolateLiquor
:color 0x7A69_dark
```

Lire Trucs et astuces pour augmenter la productivité en ligne:
https://riptutorial.com/fr/vim/topic/3382/trucs-et-astuces-pour-augmenter-la-productivite

Chapitre 48: Utiliser ex de la ligne de commande

Examples

Substitution à partir de la ligne de commande

Si vous souhaitez utiliser vim d'une manière similaire à `sed` , vous pouvez utiliser l' `-c` pour exécuter une commande ex à partir de la ligne de commande. Cette commande s'exécutera automatiquement avant de vous présenter le fichier. Par exemple, pour remplacer `foo` avec `bar` :

```
vim file.txt -c "s/foo/bar"
```

Cela ouvrira le fichier avec toutes les instances de `foo` remplacées par la `bar` . Si vous souhaitez apporter des modifications au fichier *sans* avoir à enregistrer manuellement, vous pouvez exécuter plusieurs commandes ex et faire écrire et quitter la dernière commande. Par exemple:

```
vim file.txt -c "s/foo/bar" -c "wq"
```

Note importante:

Vous *ne* pouvez *pas* exécuter plusieurs commandes ex séparées par une barre `|` . Par exemple

```
vim file.txt -c "s/foobar | wq"
```

N'est *pas* correct Cependant, cela peut être fait si vous utilisez `ex` .

```
ex -c ":%s/this/that/g | wq" file.txt
```

Lire Utiliser ex de la ligne de commande en ligne: https://riptutorial.com/fr/vim/topic/6819/utiliser-ex-de-la-ligne-de-commande

Chapitre 49: Utiliser les scripts Python for Vim

Syntaxe

- : [range] py [thon] {statement}

Examples

Vérifiez la version de Python dans Vim

Vim a son propre interpréteur Python intégré. Ainsi, il pourrait utiliser une version différente de l'interpréteur par défaut pour le système d'exploitation.

Pour vérifier avec quelle version de Python Vim a été compilée, tapez la commande suivante:

```
:python import sys; print(sys.version)
```

Cela importe le module `sys` et imprime sa propriété `version` , contenant la version de l'interpréteur Python actuellement utilisé.

Exécuter les commandes du mode normal Vim via l'instruction Python

Pour pouvoir utiliser les commandes vim en Python, le module `vim` doit être importé.

```
:python import vim
```

Après avoir importé ce module, l'utilisateur a accès à la fonction de `command` :

```
:python vim.command("normal iText to insert")
```

Cette commande exécutera `i` en mode normal, puis tapez `Text to insert` et revenir au mode normal.

Exécution de code Python multiligne

Chaque instruction Python dans Vim doit être précédée de la commande `:python` , pour indiquer à Vim que la commande suivante n'est pas Vimscript mais Python.

Pour éviter de taper cette commande sur chaque ligne, lors de l'exécution du code Python multiligne, il est possible d'indiquer à Vim d'interpréter le code entre deux expressions de marqueur comme Python.

Pour ce faire, utilisez:

```
:python << {marker_name}
a = "Hello World"
print(a)
{marker_name}
```

où `{marker_name}` est le mot que vous souhaitez utiliser pour désigner la fin du bloc python.

Par exemple:

```
:python << endpython
surname = "Doe"
forename = "Jane"
print("Hello, %s %s" % (forename, surname))
endpython
```

imprimerait:

```
Hello, Jane Doe
```

Lire Utiliser les scripts Python for Vim en ligne: https://riptutorial.com/fr/vim/topic/5604/utiliser-les-scripts-python-for-vim

Chapitre 50: vglobal: exécute les commandes sur les lignes qui ne correspondent pas globalement

Introduction

: vglobal ou: v est l'opposé de: global ou: g qui opère sur des lignes ne correspondant pas au modèle spécifié (inverse).

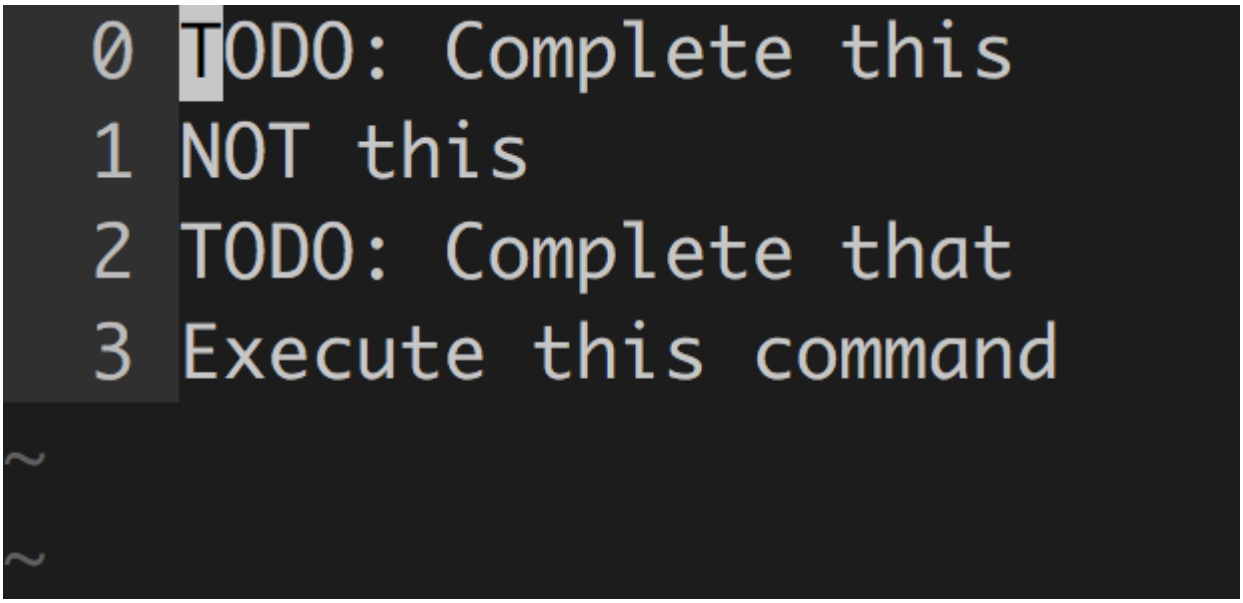
Examples

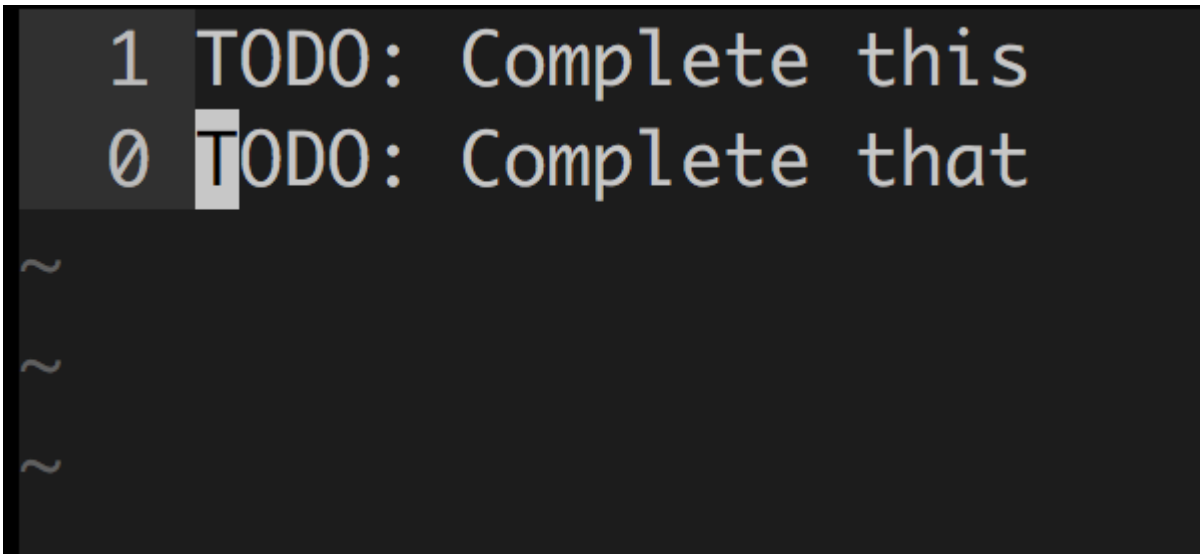
: v / pattern / d

Exemple:

```
> cat example.txt
  TODO: complete this
  NOT this
  NOT that
  TODO: Complete that
```

Ouvrez le `example.txt` utilisant `vim` et tapez `:v/TODO/d` en mode `Ex` . Cela supprimera toutes les lignes ne contenant pas le motif `TODO` .





Lire vglobal: exécute les commandes sur les lignes qui ne correspondent pas globalement en ligne: https://riptutorial.com/fr/vim/topic/9867/vglobal--execute-les-commandes-sur-les-lignes-qui-ne-correspondent-pas-globalement

Chapitre 51: Vim extensible

Remarques

Un plugin est un script ou un ensemble de scripts qui modifie le comportement par défaut de Vim, soit en ajoutant des fonctionnalités non existantes, soit en étendant des fonctionnalités existantes.

Les «fonctionnalités non existantes» souvent ajoutées incluent:

- commentant,
- détection d'indentation,
- auto-complétion,
- correspondance floue
- prise en charge d'une langue spécifique,
- etc.

Les «fonctionnalités existantes» souvent étendues incluent:

- omni-achèvement,
- objets-texte et mouvements,
- arracher et mettre,
- ligne d'état,
- rechercher et remplacer,
- changement de page tampon / fenêtre / onglet,
- pliant,
- etc.

Examples

Comment fonctionnent les plugins

Un plugin peut se présenter sous la forme d'un fichier unique contenant 30 lignes de vimscript ou 20 Mo de vimscript / python / ruby / what divisé en plusieurs fichiers sur une douzaine de répertoires dépendant d'un certain nombre d'outils externes.

Le premier est évidemment facile à installer et à gérer, mais le second peut poser un véritable défi.

Le principe

L'option `'runtimepath'` indique à Vim où rechercher les scripts d'exécution. La valeur par défaut fait que Vim recherche les scripts dans les répertoires suivants *dans l'ordre* :

- sur des systèmes de type UNIX
    - **`$HOME/.vim/`**

    - `$VIM/vimfiles/`
    - `$VIMRUNTIME/`
    - `$VIM/vimfiles/after/`
    - **`$HOME/.vim/after/`**

- sur Windows

    - **`$HOME/vimfiles/`**
    - `$VIM/vimfiles/`
    - `$VIMRUNTIME/`
    - `$VIM/vimfiles/after/`
    - **`$HOME/vimfiles/after/`**

Parmi les répertoires ci-dessus, installez uniquement les plug-ins dans ceux en gras. Les autres provoqueront l'instabilité sans raison valable. L'installation d'un plugin se résume à placer chacun de ses composants dans le bon répertoire sous `$HOME/.vim/` ou `$HOME/vimfiles/` .

La méthode manuelle

Plugin de fichier unique

Placez le fichier sous `$HOME/.vim/plugin` ou `$HOME/vimfiles/plugin`

Cela source le plugin au démarrage de Vim. Maintenant, l'utilisateur peut utiliser tout ce qui y est défini. Si le plug-in nécessite toutefois une activation, l'utilisateur doit soit exécuter la commande eux-mêmes chaque fois qu'ils le souhaitent, soit ajouter la commande à `.vimrc`

Paquet

Un bundle est une structure de répertoire utilisée par le plugin. Il comprend tous les fichiers du plugin sous les sous-répertoires appropriés.

Pour installer un tel plugin, les sous-répertoires doivent être fusionnés avec leurs homologues dans `$HOME/.vim/plugin` . Cette approche conduit cependant à mélanger les fichiers de différents plug-ins dans les mêmes répertoires et peut éventuellement entraîner des problèmes d'espace de noms.

Une autre approche consiste à copier le répertoire entier dans `$HOME/.vim/bundle` .

Lorsque vous utilisez cette approche, il devrait y avoir au moins un fichier `.vim` sous le répertoire `$HOME/.vim/bundle/autoload` . Ces fichiers seraient fournis par vim au démarrage.

Remarque: Selon le système d'exploitation de l'utilisateur, le préfixe de tous les chemins peut être `$HOME/vimfiles` . Pour plus de détails, voir Comment les plugins fonctionnent

VAM

https://github.com/MarcWeber/vim-addon-manager

Vundle

Vundle est un gestionnaire de plug-ins pour Vim.

Installation de Vundle

(Les détails complets de l'installation peuvent être trouvés dans le démarrage rapide de Vundle)

1. Installez Git et clonez Vundle dans `~/.vim/bundle/Vundle.vim` .
2. Configurez les plug-ins en ajoutant les éléments suivants en haut de votre `.vimrc` , en ajoutant ou en supprimant des plug-ins si nécessaire (les plug-ins de la liste sont simplement `.vimrc` titre d'illustration)

```
set nocompatible              " be iMproved, required
filetype off                  " required

" set the runtime path to include Vundle and initialize
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()
" alternatively, pass a path where Vundle should install plugins
"call vundle#begin('~/some/path/here')

" let Vundle manage Vundle, required
Plugin 'VundleVim/Vundle.vim'

" All of your Plugins must be added before the following line
call vundle#end()            " required
filetype plugin indent on    " required
" To ignore plugin indent changes, instead use:
"filetype plugin on

"place non-Plugin stuff after this line
```

3. Installer les plugins: en lançant Vim et en exécutant `:PluginInstall` .

Formats de plugin pris en charge

Voici des exemples de différents formats pris en charge. Conservez les commandes du plug- `vundle#begin` entre `vundle#begin` et `vundle#end` .

Emplacement du plugin	Usage
plugin sur GitHub	`Plugin 'tpope/vim-fugitive'`
plugin de http://vim-scripts.org/vim/scripts.html	`Plugin 'L9'`
Plugin Git non hébergé sur GitHub	`Plugin 'git://git.wincent.com/command-t.git'`

Emplacement du plugin	Usage
git repos sur votre machine locale (par exemple lorsque vous travaillez sur votre propre plugin)	`Plugin 'file:///home/gmarik/path/to/plugin'`
Le script vim sparkup se trouve dans un sous-répertoire de ce référentiel appelé vim. Passez le chemin pour définir le runtimepath correctement.	`Plugin 'rstacruz/sparkup', {'rtp': 'vim/'}`
Installez L9 et évitez un conflit de nommage si vous avez déjà installé une version différente ailleurs.	`Plugin 'ascenator/L9', {'name': 'newL9'}`

Travailler sur un compte partagé, par exemple, sur un nœud de tête de cluster peut générer des problèmes à partir du point d'utilisation du disque par le répertoire `.vim` . Il y a quelques paquets qui prennent une quantité considérable d'espace disque, par exemple YCM . Alors, choisissez `Vundle` votre `Vundle` plug- `Vundle` , et il est très facile de le faire en définissant `rtp` . Et si vous prévoyez d'installer un plug-in vim, ne faites pas directement `git clone` dans le répertoire `bundle` . Utilisez la méthode Vundle.

L'avenir: les forfaits

Voir `:help packages` .

Agent pathogène

vim-pathogen est un gestionnaire de `runtimepath` créé par Tim Pope pour faciliter l'installation de plugins et de fichiers d'exécution dans leurs propres répertoires privés.

Installation du pathogène

1. Mettez le pathogène dans `~/.vim/bundle` (ici avec Git, mais ce n'est pas obligatoire):

```
git clone https://github.com/tpope/vim-pathogen.git
```

2. Ajoutez les lignes suivantes en haut de votre `.vimrc` :

```
" enable vim-pathogen
runtime bundle/vim-pathogen/autoload/pathogen.vim
execute pathogen#infect()
```

- la directive `runtime` spécifie le chemin d'accès au script d'autoload de `vim-pathogen` ;
- `execute pathogen#infect()` initie.

Une fois lancé, Pathogen lance automatiquement un balayage dans les dossiers de `~/.vim/bundle` et charge le plug-in de chacun d'eux.

Utilisation de l'agent pathogène

1. Placez le répertoire `~/.vim/bundle/` de votre plug-in dans `~/.vim/bundle/` pour le rendre disponible au prochain démarrage de Vim.
2. Exécuter `:Helptags` pour indexer la documentation de votre nouveau plugin.

Avantages

- Chaque plug-in réside dans son propre répertoire sous `~/.vim/bundle/` .
- Votre `.vimrc` reste propre à la configuration nécessaire pour charger les plugins.

L'effort nécessaire pour "gérer" un plugin est ainsi réduit à:

- **mettre** son répertoire de premier niveau sous `~/.vim/bundle/` pour l' *installer* ,
- **remplacer** son répertoire de niveau supérieur pour le *mettre* à *jour* ,
- **supprimer** son répertoire de niveau supérieur pour le *désinstaller* .

La manière dont vous effectuez ces trois actions (manuellement, via un outil d'automatisation, avec Git / Svn / Hg / quel que soit…) dépend entièrement de vous.

Lire Vim extensible en ligne: https://riptutorial.com/fr/vim/topic/3659/vim-extensible

Chapitre 52: Vim Solarisé

Introduction

Passer le plus clair de son temps au terminal peut être très pénible pour les yeux. Choisir judicieusement la palette de couleurs peut être bénéfique pour vos yeux à bien des égards. Récemment, j'ai rencontré `Solarized ColorScheme for Vim` . L'ajout de ce petit plugin peut faire une grande différence sur l'apparence du texte sur le terminal. Un grand merci à Ethan Schoonover pour avoir développé ce package. Les `howtos` sont bien expliqués ici . Prendre plaisir!

Examples

.vimrc

Solarized a deux options - le mode clair et le mode sombre.

Mode lumière :

```
syntax enable
set background=light
colorscheme solarized
```

Mode sombre :

```
syntax enable
set background=dark
colorscheme solarized
```

Lire Vim Solarisé en ligne: https://riptutorial.com/fr/vim/topic/9500/vim-solarise

Chapitre 53: Vimscript

Remarques

Les commandes d'un fichier Vimscript sont exécutées en `command mode` par défaut. Par conséquent, toutes les directives sans `command mode` devraient être préfixées.

Examples

Bonjour le monde

Lorsque vous essayez d'imprimer quelque chose pour le débogage dans vimscript, il est tentant de faire simplement ce qui suit.

```
echo "Hello World!"
```

Cependant, dans le contexte d'un plugin complexe, il y a souvent beaucoup d' autres choses qui se passent juste après que vous tentez d'imprimer votre message, il est donc important d'ajouter le `sleep` après votre message afin que vous puissiez réellement le voir avant qu'il ne disparaisse.

```
echo "Hello World!"
sleep 5
```

Utilisation des commandes en mode normal dans Vimscript

Étant donné qu'un fichier Vimscript est un ensemble d'actions en mode Commande, l'utilisateur doit spécifier que les actions souhaitées doivent être exécutées en mode normal.

Par conséquent, l'exécution d'une commande en mode normal telle que `i` , `a` , `d` etc. dans Vimscript est effectuée en ajoutant la commande `normal` la commande:

En allant au bas du fichier et en sélectionnant les 5 dernières lignes:

```
normal GV5k
```

Ici, le `G` ordonne à vim de changer la position du curseur sur la dernière ligne, le `V` pour passer au mode visuel linéaire et le `5k` à 5 lignes.

Insérez votre nom à la fin de la ligne:

```
normal ABoris
```

où le `A` met l'éditeur en mode d'insertion à la fin de la ligne et le reste est le texte à insérer.

Lire Vimscript en ligne: https://riptutorial.com/fr/vim/topic/5136/vimscript

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec vim	A. Raza, akavel, Ashok, carrdelling, Christian Rondeau, Community, Cows quack, Daniel, Daniel Käfer, Daniel Margosian, Deborah V, depperm, ericdwang, ExistMe, GiftZwergrapper, gmoshkin, HerrSerker, James, Js Lim, KerDam, LittleByBlue, liuyang1, LotoLo, Marek Skiba, Mattias, Miljen Mikic, mnoronha, Nasreddine, Nhan, Nick Weseman, pktangyue, redBit Device, Romain Vincent, romainl, ropata, Rory O'Kane, Sardathrion, sascha, SeekAndDestroy, sjas, sudo bangbang, Sumner Evans, tbodt, Tejus Prasad, TheMole, timss, Tom Gijselinck, Tom Lord, user2314737, user45891, Vin, Vishnu Kumar, vvnraman, Wieland, Wojciech Kazior, zarak, Zaz
2	:global	cmlaverdiere, DJMcMayhem, LittleByBlue, tbodt, Vin
3	Amélioré défaire et refaire avec un undodir	GiftZwergrapper
4	Avantages de vim	gmoshkin, LittleByBlue
5	Bâtiment de vim	grochmal, Josh Petrie, LittleByBlue, Luc Hermitte
6	Code de format automatique	Philip Kirkbride
7	Commandes Automatiques	joeytwiddle, tbodt, Tom Hale
8	Commandes en mode normal	Tom Hale
9	Commandes en mode normal (édition)	A. Raza, rodericktech, romainl, The Nightman
10	Comment compiler Vim	Ingo Karkat, Josh Petrie, romainl
11	Configuration de Vim	Aaron Thoma, bn., Christian Rondeau, Cometsong, Cows quack, Daniel, Johnathan Andersen, KerDam, Luc Hermitte, lwassink, mezzode, nobe4, romainl, SnoringFrog, sudo bangbang, Sumner Evans, timss, Wojciech Kazior, Yosh

12	Configurations utiles pouvant être placées dans .vimrc	Cows quack, maniacmic, Tomh
13	Conversion de fichiers texte de DOS en UNIX avec vi	grochmal, LazyBrush
14	Correcteur orthographique	andipla, Johnathan Andersen, lwassink
15	Défilement	Delapouite, evuez
16	Demandez à créer des répertoires inexistants en enregistrant un nouveau fichier	Tom Hale
17	Différences entre Neovim et Vim	still_dreaming_1, tbodt
18	Échancrure	dallyingllama, Daniel, RamenChef, toto21
19	Économie	abidibo
20	Espace blanc	dallyingllama
21	Expressions régulières	4444, sudo bangbang
22	Expressions régulières en mode Ex	UNagaswamy
23	Fenêtres fendues	beardc, Boysenb3rry, Downgoat, Goluxas, grenangen, HerrSerker, Johnathan Andersen, KerDam, madD7, Sachin Divekar, sudo bangbang, timss, Victor Schröder, zarak
24	Gammes de ligne de commande	RamenChef, romainl
25	Insérer du texte	Batsu, Boysenb3rry, Christopher Bottoms, cmlaverdiere, codefly , DJMcMayhem, Eric Bouchut, GiftZwergrapper, gmoshkin, Johnathan Andersen, Kent, lazysoundsystem, Mahmood, omul, Promarbler, RamenChef, rodrigo, romainl, satyanarayan rao, Scroff, SnoringFrog, sudo bangbang, Sundeep, timss, Tom Lord , UNagaswamy, Xavier Nicollet
26	L'opérateur de points	Js Lim, LittleByBlue

27	Macros	Johan, Johnathan Andersen, lazysoundsystem, LittleByBlue, Oliver Wespi, rjmill, romainl, TheMole, Victor Schröder, vielmetti , Wenzhong
28	Manipulation du texte	Chris Nager, Christopher Bottoms, LittleByBlue, Nikola Geneshki, Philip Kirkbride, romainl, till, Tom Hale, zarak
29	Mappages de touches dans Vim	Christian Rondeau, Ingo Karkat, KerDam, Luc Hermitte, madD7, New Alexandria, Nikola Geneshki, RamenChef, romainl
30	Modes - insérer, normal, visuel, ex	rgoliveira, romainl
31	Motions et objets texte	tbodt
32	Mouvement	Aidan Miles, akavel, Boysenb3rry, Caek, Chris H, Cows quack, depperm, fedorqui, Georgi Dimitrov, gmoshkin, JacobLeach, jamessan, KerDam, Madis Pukkonen, Noam Hacker, rgoliveira, sjas, SnoringFrog, Sundeep, timss, Tyler, Vin, Wazam, zarak
33	Objets texte Vim	UNagaswamy
34	Obtenir: aide (en utilisant le manuel intégré de Vim)	Aidan Miles, Luc Hermitte
35	Œufs de Pâques	Aaron Thoma, Andrea Romagnoli, Christian Rondeau, cmlaverdiere, Daniel Käfer, Gerard Roche, LittleByBlue, Mateusz Piotrowski, Mattias, mcarton, nobe4, NonlinearFruit, sudo bangbang, tbodt, Tejus Prasad
36	Options Vim	dallyingllama, LittleByBlue, mezzode, Wojciech Kazior, Yosh
37	Pliant	Josh Petrie, LittleByBlue, sudo bangbang
38	Plugins	Nick Weseman
39	Plugins de type de fichier	Luc Hermitte, romainl
40	Recherche dans le tampon actuel	Abdelaziz Dabebi, DJMcMayhem, LittleByBlue, romainl
41	Registres Vim	maniacmic, romainl, UNagaswamy
42	Ressources Vim	Nikola Geneshki
43	Sortie de Vim	Arulpandiyan Vadivel, asclepix, AWippler, Nick Weseman, Yosef Nasr

44	Substitution	cmlaverdiere, LittleByBlue, Nikola Geneshki, Timur
45	Tampons	Chris Jones, eli, joeytwiddle, sudo bangbang
46	Trouver et remplacer	DJMcMayhem, Kara, naveen.panwar, ncmathsadist, romainl, sudo bangbang, zzz
47	Trucs et astuces pour augmenter la productivité	Abdelaziz Dabebi, adelarsq, Chris Midgley, depperm, GanitK, gath, gmoshkin, Hotschke, KerDam, LittleByBlue, LotoLo, mash, naveen.panwar, RamenChef, rjmill, romainl, Simone Bronzini, souparno majumder, Stryker, sudo bangbang, tbodt, Tom Hale
48	Utiliser ex de la ligne de commande	DJMcMayhem, Matt Clark
49	Utiliser les scripts Python for Vim	Nikola Geneshki
50	vglobal: exécute les commandes sur les lignes qui ne correspondent pas globalement	UNagaswamy
51	Vim extensible	baptistemm, LittleByBlue, Nikola Geneshki, romainl, satyanarayan rao, Sumner Evans, void
52	Vim Solarisé	Luc Hermitte, Nick Weseman, satyanarayan rao
53	Vimscript	merlin2011, Nikola Geneshki