

SYN

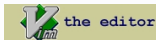
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vim - vi improved

Stefan Krist

H Furtwangen U

17. November 2005



Agenda

- 1 Warum will man vim benutzen?
- 2 vi vs. vim
- 3 vi(m)
- 4 vim
- 5 vim && C
- 6 nützliches und fragliches



SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

Andere Editoren?

- nano
- emacs
- ed
- ...



└─ Warum will man vim benutzen?

└─ Andere Editoren?

- ▼ nano
- ▼ emacs
- ▼ ed
- ▼ ...

Natuerlich gibt es auch andere Editoren die man verwenden koennte. Das Problem ist, das diese Editoren sich viel ueber STRG + \$KEY steuern lassen. Dies mag fuer den neueinsteiger ganz in Ordnung sein. Wer dann aber anfaengt sich per ssh auf anderen rechnern anzumelden und dort Produktiv werden will faellt unter Umstaenden sehr einfach auf die Schnauze, da nicht immer die richtigen Werte ankommen. Das liegt zum Teil an der \$TERM Variable kann aber auch noch andere Ursachen haben. Wie vi(m) dieses Problem loest sehen wir im folgenden. Hinzu kommt, das vim bzw. vi schon zu den Standardeditoren auf jedem linux und unix os gehoert. Man lernt also mit einem allgemein verfügbaren tool zu arbeiten.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

Modis in vi(m)

- Kommandomodus (Normalmodus)
- Einfügemodus
- Visuellermodus
- Kommandozeilenmodus
- Exmodus
- ... :help vim-modes



vim - vi improved

└─ Warum will man vim benutzen?

└─ Modis in vi(m)

Modis in vi(m)

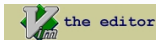
- Kommandomodus (Normalmodus)
- Einfügemodus
- Visuellermodus
- Kommandozeilenmodus
- Exmodus
- ... :help vim-modes

vim arbeitet in verschiedenen Modi die vim aus vi geerbt hat. Nahezu jede Taste reagiert anders in einem anderen Modus. Die hier aufgeführten Modis sind nur ein Teil der grossen Masse. Diese Eigenschaft macht vim gerade zu so einem starken und mächtigen Editor.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

Die Wichtigsten Unterschiede

- 1 Hilfe
- 2 undo stack
- 3 syntax highlighting
- 4 split windows
- 5 visualmode
- 6 gvim / evim *puke*
- 7 ... :help vi-diff



- Hilfe
- undo stack
- syntax highlighting
- split windows
- visualmode
- gvim / evim *pöde*
- ... help vi-diff

Neben den zahlreichen anderen neuerungen sind diese meines Erachtens nach die wichtigsten. Hier sei gesagt, dass `:set compatible vim` kompatibler zu vi macht. Wer gerade 2 Wochen Urlaub hat und nicht weiss was tun, ist hiermit gerne angeregt sich die ganzen von mir aufgeführten „:helps“ wie hier oder oben reinzuziehen.

vi hat einen undo-buffer der für einen Befehl reicht. Vim hat seinen default auf 1000 undolevels.

Zusätzlich hat vim syntax highlighting für über 180 Sprachen.

Gesplittete Fenster mit allerlei sinn(voll—frei)er Funktionen.

Einen visualmode, mit dem sich für fast jeden Befehl Wirkungsbereiche abstecken lassen was das Arbeiten sehr erleichtert.

Die Programme gvim und evim sind für GUI-lovers, wobei dennoch ein grosser Unterschied zwischen den beiden herrscht. Während gvim vim mit ein paar Icons darstellt ist evim schon ein Notepad-klon der sogar default-mässig in den Insert-Mode geht.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Die Kommandos

- 1 vi(m)
- 2 vim
- 3 vim && C



vim - vi improved

└─vi(m)

└─Die Kommandos

- vi(m)
- vim
- vim && C

Hier möchte ich eine kurze Einführung geben, über das, was der gute alte vi schon konnte und was in vim noch verfeinert und erweitert wurde.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Wechsel in den Normalmodus

- ESC, aus fast allen Modis
- STRG + c, aus fast allen Modis
- q, Recording beenden



vim - vi improved

└─vi(m)

└─vi(m)

└─Wechsel in den Normalmodus

- ESC, aus fast allen Modis
- STRG + c, aus fast allen Modis
- q, Recording beenden

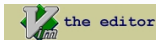
Da sich vim nur aus dem Normalmodus (eigentlich ja exmodus) beenden lässt wird erstmal gezeigt wie man den in den Normalmodus kommt, sollte man ihn den mal verlassen haben.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

vim beenden

- `:q`, beenden
- `:q!`, beenden ohne zu speichern
- `:wq`, speichern und beenden
- `ZZ`, speichern und beenden
- `:x`, speichern und beenden



vim - vi improved
└─ vi(m)
 └─ vi(m)
 └─ vim beenden

- :q, beenden
- :q!, beenden ohne zu speichern
- :wq, speichern und beenden
- ZZ, speichern und beenden
- :x, speichern und beenden

Aus dem Normalmodus heraus lässt sich vim wie aufgeführt beenden. Allerdings sei angemerkt, dass der `: vim` veranlasst in den Exmodus zugehen und diese Bezeichnung also nicht 100% richtig ist.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützlich und fraglich
FIN

vi(m)
vim

Wechsel in den Einfügemodus

- *a*, rechts vom cursor
- *A*, am Ende der Zeile
- *i*, auf dem cursor
- *I*, zu Beginn der Zeile
- *o*, Füge eine Leerzeile unter dem cursor ein
- *O*, Füge eine Leerzeile über dem cursor ein



vim - vi improved

└─vi(m)

└─vi(m)

└─Wechsel in den Einfügemodus

- a, rechts vom cursor
- A, am Ende der Zeile
- i, auf dem cursor
- I, zu Beginn der Zeile
- o, Füge eine Leerzeile unter dem cursor ein
- O, Füge eine Leerzeile über dem cursor ein

Es gibt sicher noch weitere Arten in den Einfügemodus zukommen. Ich benutze nur die angeführten. Ein weiterer Weg wäre zum Beispiel die Einfg-Taste, wobei diese bei erneutem Drücken auch in den Überschreibmodus wechselt. Von dieser Art kann ich allerdings nur abraten.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützlich und fraglich
FIN

vi(m)
vim

löschen

[“registername][zähler]d[positionierbefehl]

- *dd*, lösche die aktuelle Zeile
- *dG*, löscht alles bis zum Ende des Dokument
- *xdd*, löscht x mal die aktuelle Zeile
- *dap*, löscht den aktuellen Absatz
- *d/foo*, löscht den Text zwischen dem cursor und dem folgenden Auftreten von foo
- *“fd\$*, löscht alles bis zum Ende der Zeile und speichert den Text in Register f



vim - vi improved
└─ vi(m)
 └─ vi(m)
 └─ löschen

- ```
[*registername][zähler][positionierbefehl]
```
- dd löscht die aktuelle Zeile
  - dG löscht alles bis zum Ende des Dokument
  - xdd löscht x mal die aktuelle Zeile
  - dap löscht den aktuellen Absatz
  - d/fos löscht den Text zwischen dem cursor und dem folgenden Auftreten von foo
  - "f\$ löscht alles bis zum Ende der Zeile und speichert den Text in Register f

Das löschen ist ein etwas komplexeres Kommando. So kann hier einfach gelöscht werden und der gelöschte Text wandert in das Namenlose-Register, oder es kann ein Register in welches gespeichert werden soll mit angegeben werden. Über den, in fast jedem Befehl implementierten Zähler wird vi(m) zu einem effektivem Editor. Auch das löschen nach REGEX ist hier kein Problem. Wer viel mit den Registern arbeitet wird das Löschkommando auch mal zum ausschneiden von Texten verwenden.

SYN  
Warum will man vim benutzen?  
vi vs. vim  
**vi(m)**  
vim && C  
nützliches und fragliches  
FIN

vi(m)  
vim

## Wechsel in den Einfügemodus mit löschen

[“registername”][zähler]c[positionierbefehl]

- C, löscht den Text ab dem cursor bis zum EOL
- cc, löscht den Text der Zeile
- c0, löscht den Text links vom cursor bis zum BOL
- “bcgg, löscht die Zeile auf der der cursor steht bis zum Beginn des Dokument und speichert das in Register b.



vim - vi improved

└─vi(m)

└─vi(m)

└─Wechsel in den Einfügemodus mit löschen

- ```
[registername][zähler]:[positionierbefehl]
```
- C, löscht den Text ab dem cursor bis zum EOL
 - cc, löscht den Text der Zeile
 - c0, löscht den Text links vom cursor bis zum BOL
 - *bgy, löscht die Zeile auf der der cursor steht bis zum Begin des Dokuments und speichert das in Register b.

Das angezeigte ist nur die halbe Wahrheit. Es gibt Befehle in vim, wie zum Beispiel C, nach dem kein Positionierbefehl mehr eingegeben werden kann, da C ja schon einen Positionierbefehl beinhaltet. Ansonsten sind das nette Werkzeuge um gemütlich zu arbeiten.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützlich und fragliches
FIN

vi(m)
vim

Suche

/muster

- /foo, springt an das nächste Auftreten von foo
- ?foo, springt an das vorherige Auftreten von foo
- *, springt an das nächste Wort unter dem cursor
- #, springt an das vorherige Wort unter dem cursor
- n, sucht das nächste Auftreten der letzten Suche
- N, sucht das vorherige Auftreten der letzten Suche



2005-11-17

vim - vi improved

└─ vi(m)

└─ vi(m)

└─ Suche

Suche

/master

- /foo, springt an das nächste Auftreten von foo
- ?foo, springt an das vorherige Auftreten von foo
- *, springt an das nächste Wort unter dem cursor
- #, springt an das vorherige Wort unter dem cursor
- n, sucht das nächste Auftreten der letzten Suche
- N, sucht das vorherige Auftreten der letzten Suche

Die Suche ist sehr angenehm, da sie REGEX unterstützt.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützlich und fraglich
FIN

vi(m)
vim

Ergänzung zur Suche - 1

- `z.`, setzt die aktuelle Zeile in die Bildschirmmitte
- `z-`, setzt die aktuelle Zeile an den unteren Rand
- `z[ENTER]`, setzt die aktuelle Zeile an den oberen Rand
- `:set hls`, highlight search ist in den meisten Fällen erwünscht
- `:set ic`, ignore case
- `:set is`, incremental search, such die Zeichenkette schon beim eintippen



vim - vi improved

└─vi(m)

└─vi(m)

└─Ergänzung zur Suche - 1

- z, setzt die aktuelle Zeile in die Bildschirmmitte
- j, setzt die aktuelle Zeile an den unteren Rand
- J[ENTER], setzt die aktuelle Zeile an den oberen Rand
- :set hl, highlight search ist in den meisten Fällen erwünscht
- :set ic, ignore case
- :set is, incremental search, such die Zeichenkette schon beim eintippen

Besonders schmackhaft ist, dass sich die Suche noch über Optionen verfeinern lässt.

Ersetzen

:**[Bereich]**s/**Suchmuster**/**Ersetzung**/**[Flags]**

- `:s/foo/bar`, ersetzt das erste foo durch ein bar in der aktuellen Zeile
- `:s/foo/bar/g`, ersetzt jedes foo durch ein bar in der aktuellen Zeile
- `:%s/foo/bar/gc`, ersetzt jedes foo durch bar im Dokument und fragt vorher
- `:1,4 s/foo/bar/c`, ersetzt das erste foo durch bar von Zeile von 1 bis 4 und fragt vorher
- `:1,$ s/foo/bar/gc`, ersetzt jedes foo durch bar von Zeile von 1 bis zur letzten und fragt vorher
- `:help substitute`



vim - vi improved
└─ vi(m)
 └─ vi(m)
 └─ Ersetzen

Ersetzen

[Bereich]s/Suchmuster/Ersetzung/[Flags]

- `s/foo/bar`, ersetzt das erste foo durch ein bar in der aktuellen Zeile
- `s/foo/bar/g`, ersetzt jedes foo durch ein bar in der aktuellen Zeile
- `%s/foo/bar/gc`, ersetzt jedes foo durch bar im Dokument und fragt vorher
- `:1,4 s/foo/bar/c`, ersetzt das erste foo durch bar von Zeile von 1 bis 4 und fragt vorher
- `:1,5 s/foo/bar/gc`, ersetzt jedes foo durch bar von Zeile von 1 bis zur letzten und fragt vorher
- `:help substitute`

Das Ersetzen funktioniert ähnlich wie die Suche. Eben auch REGEX. Wobei REGEX nur Write-Only sind, soll heißen, dass man sie zwar schreiben kann aber fremde REGEX zu lesen und zu verstehen ist sehr viel aufwendiger.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Wechsel in den Überschreibmodus

- `rx`, Überschreibt das Zeichen unter dem cursor mit `x` und wechselt in den Normalmodus
- `R`, Geht wirklich in den Überschreibmodus bis er mit Eingfg oder Esc wieder verlassen wird.



Bewegung

:[Zähler][Befehl]

- *j*, setzt den cursor eine Zeile tiefer
- *h*, setzt den cursor weiter nach links
- *k*, setzt den cursor eine Zeile höher
- *x|*, setzt den cursor um x Zeichen weiter nach rechts
- *fy*, setzt den cursor auf das nächste y in der Zeile
- *Fy*, setzt den cursor auf das vorherige y in der Zeile
- *:x*, springt zu Zeile x an die erste Zeile
- *gg*, springt an die erste Zeile
- *G*, springt an die letzte Zeile



vim - vi improved
└─ vi(m)
 └─ vi(m)
 └─ Bewegung

[Zähler][Befehl]

- j, setzt den cursor eine Zeile tiefer
- h, setzt den cursor weiter nach links
- k, setzt den cursor eine Zeile höher
- l, setzt den cursor um x Zeichen weiter nach rechts
- fy, setzt den cursor auf das nächste y in der Zeile
- Fy, setzt den cursor auf das vorherige y in der Zeile
- :x, springt zu Zeile x an die erste Zeile
- gg, springt an die erste Zeile
- G, springt an die letzte Zeile

Wer auch nur einen Bruchteil dieser Bewegungs shortcuts drauf hat kann Texte editieren wie in keinem anderen Editor. Die Geschwindigkeit wird sich unter Verwendung dieser einfachen und einprägsamen Kommandos stark verbessern.

Bewegung

:[Zähler][Befehl]

-), setzt den cursor an den Anfang des Wortes nach dem folgenden Punkt
- (, setzt den cursor weiter nach links
- {, setzt den cursor eine über den vorherigen Absatz
- }, setzt den cursor unter den folgenden Absatz
- e, setzt den cursor ans Ende des folgenden smallwords
- E, setzt den cursor ans Ende des folgenden bigwords
- b, setzt den cursor an den Anfang des vorherigen smallwords
- B, setzt den cursor an den Anfang des vorherigen bigwords
- w, springt an den Anfang des folgenden smallwords
- W, springt an den Anfang des folgenden bigwords



SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

wiederholen und rückgängig

- `.`, wiederholt das letzte Kommando
- `u`, macht den letzten Befehl rückgängig
- `STRG + R`, macht den letzten undo-Befehl rückgängig

vi(m) ist fertig
Durchatmen!



vim - vi improved

└─ vi(m)

└─ vi(m)

└─ wiederholen und rückgängig

- . wiederholt das letzte Kommando
- u macht den letzten Befehl rückgängig
- STRG + R macht den letzten undo-Befehl rückgängig

vi(m) ist fertig
Durchsetzen

Auf jeden Fall überlebensnotwendig. Zumindest undo und redo. Der Punkt macht das arbeiten zwar leichter ist aber nicht von solcher Relevanz wie die beiden anderen.

vim - vi improved

└─vi(m)

└─vim

└─wiederholen und rückgängig

- . wiederholt das letzte Kommando
- x macht den letzten Befehl rückgängig
- STRG + R macht den letzten undo-Befehl rückgängig

vi(m) ist fertig
Durchatmen!

Die vim in vim eingebaute Hilfe ist riesig gross. In einem Buch von vor 2 Jahren habe ich gelesen dass sie 3,8 MB hat. Das kann allerdings nicht ganz stimmen, da sogar das doc - Verzeichnis, das nur einen ausschnitt der Hilfe enthält 4,2 MB gross ist.

Aber egal, die Hilfe ist der Hammer, man will sie einfach benutzen.

Verlinkte Textdateien die in einem gesplittetem Fenster angezeigt werden.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Hilfe

- `:h[elp]` *help*, zeigt die Hilfe zur Hilfe
- `:h[elp]` *foo*, zeigt die Hilfe zu *foo* (TAB-completion)
- `:h[elp]` *foo* **STRG+D**, zeigt jedes vorkommen von *foo* in der Hilfe
- **STRG +]**, folgt einem Hyperlink der Hilfe
- **STRG + o**,
STRG + t, springt zurück zum letzten Hilfethema

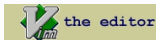


SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Visual

- *v*, Zeichenweise markieren
- *V*, Zeilenweise markieren
- *STRG+v*, Blockweise markieren
- *:h visual-use*, lesen lesen lesen...



vim - vi improved
└─ vi(m)
 └─ vim
 └─ Visual

- v Zeichenweise markieren
- V Zeilenweise markieren
- STRG+V Blockweise markieren
- :h visual-ore, lesen lesen lesen...

Der Visualmode ist nicht zu verachten. Gerade wer mit vim programmieren will kann hier sehr einfach zum einen seinen Code umbauen, was natürlich auch für nichtprogrammierer sinnvoll sein kann. Es lassen sich einfach Bereiche zum kopieren und einfügen markieren, aber auch Bereich für die Suche definieren.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

kopieren & einfügen

- “*ty*, Yankt den markierten Bereich in das Register t
- “*tx*, Schneidet den markierten Bereich aus und packt ihn in das Register t
- “*ap*, Pastet den Registerinhalt aus a nach dem cursor
- “*tP*, Pastet den Registerinhalt aus t vor dem cursor



SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Geteilte Fenster

- `:sp [Datei]`, teilt das Fenster horizontal
- `:vsp [Datei]`, teilt das Fenster vertikal
- `STRG+WW`, wechselt den Focus zwischen den Fenster
- `STRG+W+=`, stellt alle Fenster ca. gleich gross dar
- `STRG+W+R`,
`STRG+W+r`, verändert die Anordnung der Fenster
- `:qa`, schliesst alle Fenster
- `:h windows`, zeigt die 1123 Zeilen lange Hilfe zu den Fenstern



vim - vi improved
└─ vi(m)
 └─ vim
 └─ Geteilte Fenster

- `sp [Datei]`, teilt das Fenster horizontal
- `vsp [Datei]`, teilt das Fenster vertikal
- `STRG+W`, wechselt den Focus zwischen den Fenstern
- `STRG+W+`, stellt alle Fenster ca. gleich gross dar
- `STRG+W-`, verändert die Anordnung der Fenster
- `qa`, schliesst alle Fenster
- `:h windows`, zeigt die 1123 Zeilen lange Hilfe zu den Fenstern

Ebenfalls ein sehr nettes Feature. Was ich nicht in der Aufzählung habe aber unbedingt erzählen muss ist, dass `STRG+W_` das Fenster auf die Maximalgrösse setzt. So lässt sich ein im Moment zu kleines Fenster kurz auf einen Schlag vergrössern und über `STRG+W+=` wieder alle Fenster gleichermassen anzeigen. Sehr Produktiv, da man von einem Dokument kurz ins andere kann und Textteile austauschen.

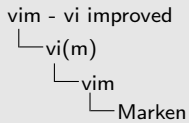
SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Marken

- *mb*, setzt die Marke b auf die aktuelle Zeile
- *'b*, springt an die Marke b
- *:marks*, zeigt alle gesetzten Marken an.





- mb, setzt die Marke b auf die aktuelle Zeile
- %b, springt an die Marke b
- :marks, zeigt alle gesetzten Marken an.

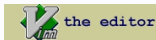
Diese Marken gab es auch schon in vi. Neu dazu gekommen ist die Auflistung über :marks.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

intentionation

- `:set ai`, Automatische Einrückung (autoindent)
- `:set cindent`, C - Spezifische Einrückung
- `:set paste`, indent toggle feature



vim - vi improved
└─ vi(m)
 └─ vim
 └─ intendation

- `:set ai`, Automatische Einrückung (autoindent)
- `:set cindent, C` - Spezifische Einrückung
- `:set paste`, indent toggle feature

intendation ist ebenfalls ein nettes feature welches sich die vim-leute einfallen lassen haben. So gibt es nicht nur verschiedene vorgefertigte intendations sondern auch wieder mal unzählige Optionen diese anzupassen. Vorallem paste ist ein nettes feature, was einfach zwischen garkeiner intendation und der zuvor eingestellten togglet um so Text zu pasten, welcher untereinander und nicht nach rechts flüchtend angezeigt wird.

Mapping

befehl “Zeichenkette“ “Neue Zeichenkette“

- *map xy /hotzenplotz <CR>*, setzt im Kommandomodus den String *xy*
- *imap if if() { <Enter><Enter>}*, setzt im Einfügemodus die Zeichenkette *if*
- *imap <F12>class { <Enter><Tab>public static void main(String args[]) { <Enter><Enter> } <Enter>}*, setzt F12 im Einfügemodus
- *:help map-overview*, zeigt verfügbare mappings an
- *:help keycodes*, zeigt verfügbare keycodes an



vim - vi improved
└─ vi(m)
 └─ vim
 └─ Mapping

Mapping

befehl "Zeichenkette" "Neue Zeichenkette"

- `map xy /hotzenplotz <CR>`, setzt im Kommandomodus den String `xy`
- `imap if if() { <Enter><Enter>`, setzt im Einfügemodus die Zeichenkette `if`
- `imap <F12>class { <Enter><Tab>public static void main(String args[]) { <Enter><Enter> } <Enter>`, setzt `F12` im Einfügemodus
- `:help map-overview`, zeigt verfügbare mappings an
- `:help keycodes`, zeigt verfügbare keycodes an

In vim gibt es `nmap`, `vmap`, `omap`, `cmap`, `lmap` und `imap` um Mappings zu definieren. Ich werde nicht alle ansprechen. Ich werde es bei `map`, `imap` und je nach Publikum `vmap` belassen. Wer mehr wissen möchte sei wie an jeder Stelle auf die Hilfe verwiesen.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

abbreviations

- *:ab tihs this*, für die chronischen Tippfheerl ;-)
- *:ab mfg Mit freundlichen Grüsen*, auch für die Faulen
- *:h abbreviations*, `while(true) { read(); }`



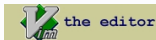
SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

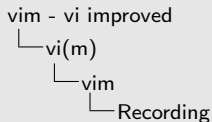
vi(m)
vim

Recording

q register

- `qa`, startet die Aufzeichnung in Register a
- `q`, beendet den Recordmode
- `@a`, „pastet“ Register a
- `:registers`, zeigt den Inhalt aller Register





- q register
- qa, startet die Aufzeichnung in Register a
 - q, beendet den Recordmode
 - @a, „pastet“ Register a
 - :registers, zeigt den Inhalt aller Register

Dieses Feature lässt den Editor zu einem Powertool der Extraklasse werden. Ich kann Kommandos während dem eintippen aufnehmen, die sogar schon aufgezeichnete Kommandos benutzen und hinterher eins-zu-eins wieder ausführen lassen. Verknüpft mit mappings geht eine Tür auf die die Produktivität ins unermässliche steigert.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Faltungen

- `:zfap`, Faltet den aktuellen Absatz
- `:zd`, löscht die Faltung unter dem cursor
- `:zE`, löscht alle Faltungen im Dokument



vim - vi improved
└─ vi(m)
 └─ vim
 └─ Faltungen

- :zfp, Faltet den aktuellen Absatz
- :zf, löscht die Faltung unter dem cursor
- :E, löscht alle Faltungen im Dokument

Seit ich dieses Feature kenne ist mein ganzes Dokument gefaltet. Ein sehr nettes Spielzeug für grosse Dokumente. So wurde zum Beispiel dieser Vortrag mit 46 gefalteten Absätzen für mich nicht nur überschaubar sondern auch noch editierbar gehalten. Sich zfp auf eine Taste zu mappen ist sicher nicht das verkehrteste.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

options

- `:set foo`, setzt boolsche Optionen auf true
- `:set nofoo`, setzt boolsche Optionen auf false
- `:set foo?`, zeigt den Wert von Optionen an
- `:set foo=text`, setzt Textoptionen auf Text
- `:h option-list, options, option-summary`, Lesestoff



2005-11-17

vim - vi improved
├── vi(m)
│ ├── vim
│ └── options

options

- `set foo`, setzt boolische Optionen auf true
- `set nofoo`, setzt boolische Optionen auf false
- `set foo?`, zeigt den Wert von Optionen an
- `set foo=text`, setzt Textoptionen auf Text
- `h option-list, options, option-summary`, Lesestoff

Die vim-options in Verbindung mit der `.vimrc` machen ein sehr stressfreies und komfortables arbeiten möglich.

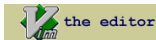
SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

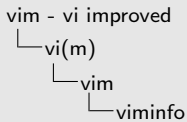
vi(m)
vim

viminfo

Optionsabhängig wird folgendes gespeichert:

- *Liste der Puffer,*
- *Histories,*
- *Register,*
- *Marken,*
- *globale Variablen,*





Optionsabhängig wird folgendes gespeichert:

- Liste der Puffer,
- Historie,
- Register,
- Markierungen,
- globale Variablen.

Hier kann auch sehr viel „UnFUG“ getrieben werden. Wobei die Arbeit mit diesem Feature sehr viel verbrachte Zeit in vim voraussetzt. Man setzt sich nicht hin und fängt an sich seine verschiedenen viminfos zu backuen. Der Sinn dahinter kommt erst zutage, wenn man sehr viel mit vim und auch sehr viel zeitgleich mit vim arbeitet.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

viminfo - 2

- `:wv[iminfo][!][Datei]`, schreibt die viminfo, ggf in Datei
- `:rv[iminfo][!][Datei]`,
`vim [-i Datei]`, viminfo einlesen
- `:help viminfo`



SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

vimrc

Was speichert die vimrc alles?

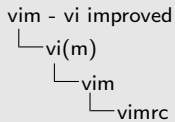
- *settings*
- *mappings*

Nützlich

- `:mkvimrc[!][Datei]`, Erstellt eine vimrc, ggf in Datei, mit den aktuellen Einstellungen



2005-11-17



vimrc

Was speichert die vimrc alles?

- settings
- mappings

Nützlich

- `mkvimrc![[Datei]]` Erstellt eine vimrc, ggf in Datei, mit den aktuellen Einstellungen

Besonders das nette Extra, mkvimrc macht es besonders einfach seine eigene vimrc zu bauen und Problemlos zu erweitern. Ansonsten ist denke ich jedem die Relevanz dieser Datei klar. Jeder sollte eine Eigene haben.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Views

Was speichert ein View-Datei?

- *Cursorposition*
- *Faltungen*
- *lokale Optionen*
- *Mappings*
- *:help viewoptions*



2005-11-17

vim - vi improved
├── vi(m)
│ ├── vim
│ └── Views

Views

Was speichert ein View-Datei?

- Cursorposition
- Faltungen
- lokale Optionen
- Mappings
- `.help viewoptions`

Views sind nicht schlecht für lokale Mappings und -Wofür sind views gut?

```
SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN
```

```
vi(m)
vim
```

Views

- `:mkview`, legt eine Datei in `/.vim/views` an (`/.vim` muss existieren)
- `:loadview`, liest die view aus der passenden Datei
- `:help viewoptions`



SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Sitzungen

Was speichert ein Sitzungs-Datei?

- *Verzeichnis*
- *globale Einstellungen*
- *Fensterparameter*
- *:help 'sessionoptions'*



vim - vi improved
└─ vi(m)
 └─ vim
 └─ Sitzungen

Was speichert ein Sitzungs-Data?

- Verzeichnis
- globale Einstellungen
- Fensterparameter
- .help 'sessionoptions'

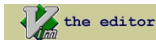
In Verbindung mit einer viminfo und einem View ist eine Session als hätte man seinen vim nie beendet. Ich steige an der selben Stelle mit den selben Registerinhalten, den selben gesplitteten Fenstern den selben Faltungen, den selben Mappings... wieder ein.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Sitzungen

- `:mks[ession] [!][Datei]`, speichert eine Sitzung in eine Datei
- `vim -S Session.vim`, öffnet eine Sitzung aus der Datei



SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützlich und fragliches
FIN

vi(m)
vim

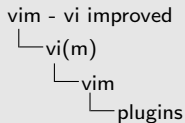
plugins

- plugins == skripte die beim start geladen werden

Wir unterscheiden:

- *ftugins*,
- *lokale plugins*,
- *globale plugins*,
- *:h plugin STRG + D*, read, read, read...





- plugins — skripte die beim start geladen werden

Wir unterscheiden:

- lokale plugins,
- globale plugins,
- :h plugin STRG + D, read, read...

vim-skripte, welche in einem der plugin-Verzeichnisse werden als Plugin geladen. Diese findet man schön Übersichtlich bei www.vim.org. Die Anzahl ist riesig und wird noch wachsen. Momentan gibt es 1319 Skripte die alle mehr oder weniger nützlich sind. Kurze Anleitungen sowie eine interne Bewertung machen das Leben einfach.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

plugins

Vim hat 118 eigene Methoden

- :help function-list, functions



vim - vi improved
├── vi(m)
│ ├── vim
│ └── plugins

natürlich hat vim auch schon fertige Funktionen mit denen das arbeiten einfach gemacht wird. Wer sich also selbst ein eigenes Skript bzw. Plugin schreiben möchte kann sich mit diesen Funktionen eine Menge arbeit ersparen.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

vi(m)
vim

Debugging

Was kann der Debugger

- *cont*, Fortsetzen bis zum nächsten Breakpoint
- *step*, Schrittweises ausführen
- *finish*, Debug-Mode für die aktuelle Funktion beenden
- *:h debug-scripts*,



2005-11-17

vim - vi improved
└─ vi(m)
 └─ vim
 └─ Debugging

Debugging

Was kann der Debugger

- `cont`, Fortsetzen bis zum nächsten Breakpoint
- `step`, Schrittweises ausführen
- `finish`, Debug-Mode für die aktuelle Funktion beenden
- `!h debug-scripts`

Wer selbst skriptet und nicht weiter weiß kann auf vims hauseigenen debugger zurückgreifen. Mit den folgenden Funktionen ist das debuggen kein Problem.

nützliches

- *K*, ruft die man-page für den Befehl unter dem cursor
- *STRG + P*, autovervollständigung für das begonnene Wort
- *STRG + N*, autovervollständigung für das begonnene Wort
- *STRG + V*, Kommentare einfügen + löschen
- *:set cindent*, einrückung im C-Stil
- *:set tabstop=3*, setzt Tab auf 3 Zeichen
- *:set shiftwidth=3*, einrückung um 3 Zeichen



vim - vi improved

└ vim && C

└ nützliches

nützliches

- K, ruft die man-page für den Befehl unter dem cursor
- STRG + P, autovervollständigung für das begonnene Wort
- STRG + N, autovervollständigung für das begonnene Wort
- STRG + V, Kommentare einfügen + löschen
- :set cinindent, einrückung im C-Stil
- :set tabstop=3, setzt Tab auf 3 Zeichen
- :set shiftwidth=3, einrückung um 3 Zeichen

Für C-Programmierer ein muss! Es erleichtert einfach die Arbeit und macht den Code ohne eigenes zutun wesentlich übersichtlicher.

nützliches

- `:make`, braucht ein Makefile, springt zum ersten Aufgetretenem Fehler
- `:cl`, listet alle errors und warnings die bei `:make` aufgetreten sind
- `:cn`, springt zum nächsten Fehler aus `:cl`
- `:cp`, springt zum vorherigen Fehler aus `:cl`



SYN

Warum will man vim benutzen?

vi vs. vim

vi(m)

vim && C

nützliches und fragliches

FIN

nützliches

- `:gf`, goto file



2005-11-17

vim - vi improved

└ vim && C

└ nützliches

nützliches

▼ gf goto file

goto file springt zur der unter dem cursor angegebenen Datei. Braucht das netrw plugin mit dem dann beinahe jede Art von Netzverbindung möglich ist.

nützliches und fragliches

- *STRG + A*, inkrementiert die Zahl unter dem cursor (d,h,o)
- *STRG + X*, dekrementiert die Zahl unter dem cursor (d,h,o)
- *STRG + G*, zeigt den Dateinamen, die Zeilennummer und den prozentualen Stand
- *~*, der markierte Bereich tooglet von Gross- auf Kleinbuchstaben
- *g?*, verschlüsselt den markierten Text Rot13
- *gv*, stellt nach einem abgegebenen Befehl die Markierung wieder her.



└─ nützliches und fragliches

└─ nützliches und fragliches

- `STRG + A`, inkrementiert die Zahl unter dem cursor (d,h,o)
- `STRG + X`, dekrementiert die Zahl unter dem cursor (d,h,o)
- `STRG + G`, zeigt den Dateinamen, die Zeilennummer und den prozentualen Stand
- `T`, der markierte Bereich togglet von Gross- auf Kleinschreibstaben
- `g?`, verschmälert den markierten Text Rot13
- `gx`, stellt nach einem abgegeben Befehl die Markierung wieder her.

Die Taste `g` hat insgesamt in vim 6 ungefähr 50 Belegungen, da alle anderen Tasten schon belegt sind. Daher kommen die Doppelbelegungen wie `gv`, `g?`... Man sollte sich aber von der Vielfalt der Tastenkombinationen nicht abschrecken lassen. Wer es bis hier geschafft hat, hat das schlimmste Überstanden und weiss, das es nur auf die, die das Arbeiten auch wirklich produktiver machen ankommt.

SYN
Warum will man vim benutzen?
vi vs. vim
vi(m)
vim && C
nützliches und fragliches
FIN

Quellen und Hinweise

- vim GE-PACKT, Reinhard Wobst
- vi lernen, <http://www.gentoo.org/doc/de/vi-guide.xml>
- vintutor, kleiner Einstiegstest
- for i in 'possible_words'; do :help \$i; done



SYN

Warum will man vim benutzen?

vi vs. vim

vi(m)

vim && C

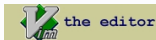
nützliches und fragliches

FIN

FIN

Weitere Fragen???

ERSCHLAGEN!



Sollte aus dem Publikum noch irgendjemand wach sein und die Gefahr einer Steinigung durch folgenden Satz aufsichziehen „das war doch noch lange nicht alles...“ kann man ihm getrost die bislang unerwähnten Dinge präsentieren.

Autokommandos vorlegen. Dies funktioniert über Ereignisse die in vim auftreten können wie zum Beispiel das öffnen einer Datei. Auf solch ein Ereignis kann reagiert werden oder man ignoriert es einfach. Mehr dazu in :h au

Auch *modelines* wurden noch nicht erwähnt. Das sind vim-kommandos die in der Datei stehen und beim laden ausgeführt werden. :h modeline