

VI Einführung

Jan Theofel

13. Oktober 2002

Diese Zusammenfassung stellt eine Einführung in den Editor *vi*, insbesondere den von uns verwendeten Klone *vim*, dar.

Inhaltsverzeichnis

1	Warum VI?	3
2	Hilfen rund um vi	4
2.1	Bücher	4
2.2	Tasse	4
3	Grundlagen	4
3.1	ex	4
3.2	Modi	5
3.3	Hilfe	5
4	Starten, Speichern, Beenden	5
4.1	Aufrufmöglichkeiten des VI	5
4.2	Speichern und Beenden	5
5	Bewegung im Text	6
6	Editier-Befehle	6
6.1	Einfügen-Befehle	6
6.2	Ersetzen	6
6.3	Löschen	7
7	Kopieren / Einfügen	7
7.1	Text kopieren	7
7.2	Text einfügen	7

8	Suchen / Ersetzen	7
8.1	Suchbefehle	7
8.2	Ersetzen-Befehle	8
9	Sonstige Befehle	9
10	Erweiterte Möglichkeiten	9
10.1	Mehrfaches Ausführen	9
10.2	Abkürzungen	9
10.3	Makros	9

1 Warum VI?

vi ist ein Textbildschirm-orientierter Editor. Trotz der modernen GUI-Editoren, gibt es eine Reihe von guten Gründen trotzdem den *vi* vorzuziehen. (Ich selbst verwende ihn ausschließlich. Alle Folien und Texte von mir sind mit ihm erstellt worden, aber auch sämtlicher HTML-Code und alle Programme.)

Freie Software *vi* ist als freie Software erhältlich. Bei *VIM* handelt es sich z. B. um „Charityware“, was bedeutet, daß die Benutzer gebeten werden für einen bestimmten Zweck zu spenden falls ihnen das Programm gefällt. Allerdings gibt es auch eine ganze Reihe kommerzieller Versionen.

klein, schnell, flexibel Das ausführbare Programm hat etwa 600kB - beachtlich klein für einen Editor der so mächtig wie der *vi* ist. Selbst die (inzwischen verfügbare) GUI-Version für Windows ist nicht viel größer.

Durch eine Konfigurationsdatei kann der Editor stark an individuelle Bedürfnisse angepaßt werden.

„Terminal-Tauglichkeit“ Als einziger mir bekannter Editor läßt sich *vi* ausschließlich über die alphanumerische Tastatur ohne Sondertasten und Strg oder Alt bedienen. Dies ist insbesondere bei Terminals wichtig, die diese Tasten einfach nicht besitzen oder bei solchen, die die nicht richtig übertragen.

UNIX-Systeme Auf (fast?) jedem UNIX-System ist mindestens ein *vi* installiert. Damit stellt er den „De-facto-Standard“ für diese Systeme dar. Dabei kommt entweder eine vom UNIX-Hersteller geschriebene, kommerzielle Version zum Einsatz oder aber eine Portierung der freien Versionen.

Verfügbarkeit Darüber hinaus ist *vi* für (fast?) jedes Betriebssystem verfügbar. Hier eine kleine Liste (die auch fast vollständig der von VIM entspricht): AmigaOS, AtariaMiNT, BeOS, DOS, MacOS, OS/2, RiscOS, VMS, Windows (3.1,3.11,CE/95/98/NT4/NT5) und natürlich UNIX (A/UX, AIX, BSDI, Convex, DYNIX/ptx, DG/UX, Dec UNIX, FreeBSD, HPUX, Irix, Linux [Debin, RedHat, SuSE], MacOSX, NetBSD, NEXTSTEP, OpenBSD, OSF, QNX, SCO, Sinix, Solaris, SunOS, SUPER-UX, Ultrix, Unixware, Unisys)

Der einzige mir bekannte Nachteil liegt in der etwas gewöhnungsbedürftige Handhabung. Dies gilt insbesondere, wenn man sich als Benutzer erst mal an „klicki-bunti“ gewöhnt hat.

Die Unix-/Linux-Welt splittet sich daher eigentlich in zwei Lager: Die *vi*-Benutzer und die *vi*-Hasser. (Dabei gehöre ich der ersteren an.)

2 Hilfen rund um vi

2.1 Bücher

Zu *vi* gibt es zwei gute Bücher. Das erste ist eines aus der *pocket-reference*-Serie von O'Reilly. Es eignet sich eher um mal eben ein Kommando nachzuschlagen, denn zum Lernen des Editors. Hierzu bietet sich eher das zweite, umfassendere Werk an, von dem ich hier die deutsche Übersetzung aufgenommen habe.

Arnold Robbins:

vi Editor — Pocket Reference

O'Reilly (ISBN: 1 - 56592 - 497 - 5)

Linda Lamb & Arnold Robbins:

Textbearbeitung mit dem vi Editor

O'Reilly (ISBN: 3 - 89721 - 126 - 2)

2.2 Tasse

Und natürlich hat das *Merchandising* auch den *vi*-Fan nicht verschont. Für etwa 12 EUR gibt es „Die VI Referenz Tasse“. Bestellen kann man sie z. B. bei Lehmann's Online-Buchhandlung (www.lob.de).

3 Grundlagen

3.1 ex

vi baut noch auf dem alten zeilenbasierten Editor *ex* auf. Diese Tradition lebt bis heute in den Befehlen weiter. Oftmals erscheinen diese daher sehr umständlich. Auf der anderen Seite sichern sie auch die Mächtigkeit des *vi*. So kann z. B. das AI-Spiel „Conway's Game of Life“ oder die „Türme von Hanoi“ mit wenigen Zeilen *vi*-Befehlen programmiert werden.

vi Klone Wie bereits beiläufig erwähnt gibt es viele verschieden VI Klone, von denen einige frei, andere dagegen kommerziell sind. Im Rahmen dieser Schulung verwenden wir dabei den **VIM** (**VI IM**proved). Einer der vorrangigen Gründe ist der, das *VIM* inzwischen auch unter Windows läuft und damit alles auch auf einem solchen System nachvollziehbar ist.

Die neusten *VIM*-Versionen kann man am besten über die Homepage von *VIM* beziehen¹. Dort finden sich auch weitere Dokumentationen und die neusten Developer-Versionen sowie Verweise auch Mailinglisten zu *vi*.

¹<http://www.vim.org/>

3.2 Modi

Der wichtigste Unterschied zu anderen Editoren ist der, daß man in *vi* nicht einfach „drauf los tippen“ kann. Dies ist nötig um sowohl die Kommandos als auch die Eingabe über die alphanumerische Tastatur zu ermöglichen (und hat auch noch andere historische Gründe).

Daher muß zur Texteingabe zunächst in den *insert-mode* umgeschaltet werden. Dies geschieht meistens durch Drücken der Taste `i`. In Abschnitt 6.1 werden auch noch einige anderen Befehle beschrieben, die auch in diesen *mode* umschalten.

Zum Verlassen des *insert-mode* muß nur die Taste `ESC` gedrückt werden. Danach befindet man sich – wie beim Programmstart – im Befehlsmodus.

3.3 Hilfe

Neben der Hilfe durch die bekannten *man-pages* (Eingabe von `man vi` in der *shell*.) steht noch ein internes und wesentlich umfangreicheres Hilfesystem zur Verfügung. Dieses kann durch die Eingabe des Befehls `:help` im Kommandomodus angezeigt werden. Soll ein bestimmtes Hilfethema angezeigt werden, kann dies durch `:help topic` geschehen.

4 Starten, Speichern, Beenden

4.1 Aufrufmöglichkeiten des VI

VI kann einfach durch die Eingabe von `vi` in der Kommandozeile gestartet werden. Allerdings können noch einige Parameter angegeben werden, von denen die wichtigsten in der folgenden Tabelle dargestellt sind.

<code>vi file</code>	Ruft VI mit <i>file</i> auf
<code>vi -R file</code>	Öffnet <i>file</i> schreibgeschützt
<code>vi -r file</code>	Stellt <i>file</i> nach einem Absturz wieder her
<code>vi + file</code>	Öffnet <i>file</i> an der letzten Zeile
<code>vi +n file</code>	Öffnet <i>file</i> an der Zeile <i>n</i>

4.2 Speichern und Beenden

Die Befehle zum Speichern und zum Beenden sind nur im Kommandomodus verfügbar. Da sie zum Teil zusammengehören sind sie in einer Tabelle zusammengefaßt.

<code>:w</code>	Datei speichern
<code>:w!</code>	Speichern (auch bei Schreibschutz)
<code>:q</code>	VI beenden
<code>:q!</code>	Beenden ohne Änderungen zu speichern
<code>:wq</code>	Schreiben des Puffers und danach beenden

5 Bewegung im Text

Neben den Pfeil- und Sondertasten die auch unter VI ihre gewohnten Funktionen haben, stehen noch folgende Tasten im Kommandomodus zur Verfügung:

h, j, k, l	Ersatz für die Pfeiltasten
0, \$	Zeilenanfang / Zeilenende
^	Erstes Zeichen in der Zeile (keine blanks)
n	Spalte <i>n</i> in der aktuellen Zeile
H, M, L	Erste, middle, letzte Zeile des Bildschirms
nG	Gehe zu Zeile <i>n</i>
G	Gehe zur letzten Zeile der Datei

6 Editier-Befehle

6.1 Einfügen-Befehle

Die folgenden Befehle schalten alle in den *insert-mode* um. Lediglich die Stelle, an der dann Text eingefügt wird, unterscheidet sich.

i, a	Einfügen vor / nach dem Cursor
I, A	Einfügen am Zeilenanfang / -ende
O, o	Neue Zeile über / unter dem Cursor einfügen

6.2 Ersetzen

Auch die Befehle zum Ersetzen schalten in den *insert-mode*. Dabei löschen sie aber vorher den zu ersetzenden Teil des Textes. (Eine Ausnahme stellt dabei nur das Ersetzen eines einzelnen Zeichens dar.)

rx	Zeichen unter dem Cursor durch <i>x</i> ersetzen
cw	Bis zum Wortende ersetzten (Wechselt in Insert-Mode)
cc	Aktuelle Zeile ersetzen (Wechselt in Insert-Mode)
C	Bis zum Zeileneende ersetzten (Wechselt in Insert-Mode)

6.3 Löschen

Neben den gewohnten Tasten `Entf` und `←` stehen noch weitere Befehle zum Löschen zur Verfügung.

`x,X` Löscht Zeichen unter / vor dem Cursor
`dw` Bis zum Wortende löschen (mit space)
`de` dito, aber ohne space
`dd` Aktuelle Zeile löschen
`D,d$` Bis zum Zeileneende löschen

Die Befehle zum Einfügen von gelöschtem Text werden in Abschnitt 7.2 angegeben.

7 Kopieren / Einfügen

In VI gibt es verschiedene Textpuffer. Diese werden verwendet um bestimmte Textstellen zu kopieren bzw. gelöschte wiederherzustellen.

`1 - 9` Text aus den letzten Löschvorgängen
`a - z` Benutzerdefinierte Puffer
`A - Z` Hängt Text an die Puffer `a - z` an.

7.1 Text kopieren

Um Text zu kopieren werden die Befehlskombinationen beginnend mit `y` (für *yank*) verwendet:

`yw` Aktuelles Wort kopieren
`yy` Aktuelle Zeile kopieren
`\{a}yy` Aktuelle Zeile in Puffer `a` kopieren

7.2 Text einfügen

Kopierte bzw. gelöschte Texte können wie folgt wieder eingefügt werden.

`P,p` Letzen kopierten / gelöschten Text vor / nach dem Cursor einfügen.
`\{a}P \{a}p` dito für den Puffer `a`

8 Suchen / Ersetzen

8.1 Suchbefehle

VI kennt verschiedene Befehle zum Suchen innerhalb des Textes. Diese sind in der folgenden Tabelle zusammengefaßt.

Dabei werden bestimmte Zeichen im Suchmuster besonders bewertet. Sie sind in der zweiten Tabelle dargestellt.

<code>/muster</code>	Sucht nach unten nach <i>muster</i>
<code>?muster</code>	Sucht nach oben nach <i>muster</i>
<code>n,N</code>	Nächster Treffer in gleicher / andere Richtung
<code>/,?</code>	Letzte Suche nach oben / unten wiederholen
<code>%</code>	Ermittelt die passende Klammer

Beim letzten Befehl springt VI mit dem Cursor zu der passenden Klammer. Befindet sich der Cursor bei der Eingabe nicht auf einer Klammer, springt er zu der Klammer, die vor ihm aufgeht, falls er sich zwischen zwei Klammern befindet.

<code>.</code>	Jedes beliebige Zeichen (außer Zeilenende)
<code>*</code>	Vorheriges Zeichen beliebig oft (auch 0 mal)
<code>^</code>	Zeilenanfang
<code>\$</code>	Zeilenende
<code>[...]</code>	Beliebiges Zeichen aus der Menge zwischen den Klammern
<code>[^...]</code>	Jedes beliebige Zeichen, das nicht in aus dieser Menge stammt
<code>\(... \)</code>	Sichert das Suchmuster zur Wiederholung in der Suche
<code>\1 bis \9</code>	Wiederholt eines der verfügbaren neun Suchmuster
<code>\<</code>	Wortbeginn
<code>\></code>	Wortende
<code>~</code>	Suchmuster der letzten Suche
<code>\</code>	Entwertet jedes beliebige Sonderzeichen

8.2 Ersetzen-Befehle

Zum Ersetzen innerhalb der aktuellen Zeile kann folgender Befehl verwendet werden:
`:s/alt/neu/gc`

Dabei sind `g` und `c` optionale Parameter. `g` bedeutet, daß alle Vorkommen ersetzt werden, `c` ersetzt mit Rückfrage. Dabei können für das Suchmuster auch die oben erläuterten Sonderzeichen verwendet werden.

Soll von Zeile `x` bis `y` ersetzt werden, so ist folgender Befehl zu verwenden (`g` und `c` sind wieder optional): `:x,ys/alt/neu/gc`

Um in der ganzen Datei zu ersetzen wird `:1,$s/alt/neu/gc` oder `:%s/alt/neu/gc` verwendet.

9 Sonstige Befehle

Hier noch eine but gemischte Liste mit verschiedenen Befehlen:

.	Letzten Editierbefehl wiederholen
u	Letzten Befehl rückgängig machen
U	Aktuelle Zeile wiederherstellen
J	Zwei Zeilen verbinden (nächste wird angehängt)
:\$d	Löscht bis zum Dateiende
:x,ya	Schiebt die Zeilen $x - y$ nach a
:x,yw file	Schreibt Zeilen $x - y$ als Datei $file$
:x,yw>> file	Hängt die Zeilen $x - y$ an $file$ an
:w! file	Schreibe Datei als $file$
:w %.neu	Schreibe Datei mit Endung neu
:e file	Öffnet zusätzlich Datei $file$
:r file	Datei $file$ einfügen
:n	Gehe zur nächsten Datei
:!befehl	Führt den Befehl $befehl$ aus

10 Erweiterte Möglichkeiten

10.1 Mehrfaches Ausführen

Fast jeder Befehl kann durch Voranstellen einer Zahl **mehrfach** ausgeführt werden.

Beispiele:

```
10dd      löscht die nächsten 10 Zeilen
10i. erzeugt: „.....“
5\"{a}yy  fügt die nächsten 5 Zeilen in  $a$  ein
```

10.2 Abkürzungen

Häufig verwendete Textstellen können als **Abkürzung** definiert werden.

```
:ab kurz Lange_Form
```

Abkürzungen wieder löschen:

```
:unab kurz
```

10.3 Makros

Auf eine ähnliche Art können **Makros** definiert werden:

```
:map x befehlsfolge
```

...und wieder auflösen:

```
:unmap x
```