

Editieren mit Vi

Jonathan Kleinhellefort
jk@molb.org

LUG Burghausen

Burghauser Linux-Informationstage 2005

Beschreibung

Vi ist ein sogenannter Bildschirm-orientierter Editor für Unix.

Er basiert auf dem zeilenorientierten Editor Ex.

Der Name ist von der Abkürzung für das Ex-Kommando `visual` abgeleitet, welches Ex in den „graphischen“ Modus schaltet.

Geschichte

Vi wurde ursprünglich von [Bill Joy](#) für [BSD](#) im Jahre 1976 geschrieben.

Er war einer der ersten „graphischen“ Editoren und wurde bald zum De-Facto-Standard unter Unix.

Ab 1984 wurde ihm dieser Status allerdings durch [Emacs](#) wieder streitig gemacht.

Vor- und Nachteile

Vorteile

- ▶ klein und schnell
- ▶ komplizierte Operationen erfordern nur wenige Tastenanschläge
- ▶ auf (beinahe) jedem Unix-System installiert

Nachteile

- ▶ lange Lernphase

Vi-Varianten

Auf modernen Unix-Systemen findet man nicht mehr den original Vi, sondern einen von mehreren Vi-Klonen:

NVi Eine möglichst Feature-Treue Kopie von Vi

Vim Vi-kompatibler Editor mit vielen neuen Funktionen

Vile Vi-Variante mit integriertem Perl-Interpreter.

Elvis Ebenfalls eine Variante mit zusätzlichen Features.

Viper Ein Vi-ähnlicher Modus für Emacs.

Vi Starten

Vi wird unter Unix mit dem Kommando `vi` gestartet.

```
vi [options] [file ...]
```

Gibt man einen Dateinamen *file* als Argument, wird die Datei geöffnet, falls sie existiert.

Die Optionen werden selten gebraucht. Bei Bedarf können sie in der Manpage `vi(1)` nachgeschlagen werden.

Terminal-Grundlagen

Als Terminal bezeichnet man ein **Ein-/Ausgabegerät**, das über ein Kabel oder Netzwerk mit einem Computer verbunden ist.

Die ersten Terminals waren so genannte zeichenbasierte **Teletypes (TTYs)**, die aus einem **Drucker** (später **Monitor**) und einer **Tastatur** bestanden.

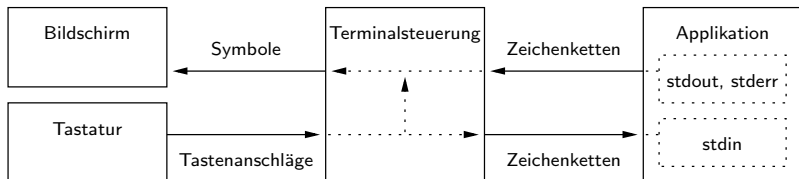
Da viele Unix-Programme zur Ein- und Ausgabe ein Terminal erwarten, gibt es auf heutigen Unix-Systemen **Terminal-Emulatoren**.

Funktionsweise von Terminals

Tastatureingaben werden an die **Standardeingabe** der darin laufenden Applikation gesendet. Die **Standardausgabe** sieht man auf dem Schirm.

Einige Ein- und Ausgaben werden vom Terminal selbst speziell interpretiert.

Abbildung: Terminal Ein-/Ausgabe



Kontrollzeichen

Kontrollzeichen (Steuerzeichen) sind nicht-druckbare Zeichen (ASCII-Werte 0–31).

Sie werden beim Betätigen manchen Tasten erzeugt, oder wenn man eine Taste zusammen mit **Control** (Steuerung) drückt.

In Vi werden sie durch ein vorangestelltes Dach dargestellt (z.B. Newline: `^M`; Escape: `^[]`).

Kontrollzeichen

Kontrollzeichen (Steuerzeichen) sind **nicht-druckbare** Zeichen (ASCII-Werte 0–31).

Sie werden beim Betätigen manchen Tasten erzeugt, oder wenn man eine Taste zusammen mit **Control** (Steuerung) drückt.

In Vi werden sie durch ein vorangestelltes Dach dargestellt (z.B. Newline: `^M`; Escape: `^[]`).

Eingabe

Will man ein Steuerzeichen in Vi eingeben, muss man verhindern, dass es als Befehl interpretiert wird, indem man ihm ein `^V` voranstellt.

Anzeige

In allen Zeilen, bis auf der letzten, wird der Inhalt des aktuellen **Puffers** gezeigt.

Nicht-existierende Zeilen werden durch ein Tilde (~) gekennzeichnet.

In der untersten Zeile werden Statusinformationen angezeigt. Bei manchen Befehlen erscheint hier auch ein Eingabefeld.

Arbeitsmodi

Im Gegensatz zu den meisten anderen Editoren wird in Vi im Grundzustand jede Eingabe als Befehl verstanden (**Befehlsmodus**).

Durch einen Einfügebefehl gelangt man in den **Eingabemodus**, in dem Eingaben in den Editorpuffer **eingefügt** werden.

Eingaben beendet man mit **^[]** (Escape) oder **^C**.

Arbeitsmodi

Im Gegensatz zu den meisten anderen Editoren wird in Vi im Grundzustand jede Eingabe als Befehl verstanden (**Befehlsmodus**).

Durch einen Einfügebefehl gelangt man in den **Eingabemodus**, in dem Eingaben in den Editorpuffer **eingefügt** werden.

Eingaben beendet man mit `^[` (Escape) oder `^C`.

Beispiel

```
iDieser Text wird eingefügt.^[
```

Befehlsstruktur

Vi Befehle bestehen meistens aus **einem Zeichen** und erwarten manchmal noch ein Zeichen als **Argument**, das folgen muss.

Manche Befehle erwarten ein Endzeichen (z.B. **^M** bei der Suche).

Durch Voranstellen einer Zahl führt man Befehle mehrmals aus.

Befehlsstruktur

Vi Befehle bestehen meistens aus **einem Zeichen** und erwarten manchmal noch ein Zeichen als **Argument**, das folgen muss.

Manche Befehle erwarten ein Endzeichen (z.B. `^M` bei der Suche).

Durch Voranstellen einer Zahl führt man Befehle mehrmals aus.

Beispiele

- ▶ Gehe zum nächsten Wort: `w`
- ▶ Lösche 3 Wörter: `d3w`, `3dw` oder `dwdwdw`
- ▶ Suche „foo“: `/foo^M`
- ▶ Hänge „barbar“ an das Ende der Zeile: `2Abar^[`

Ex-Kommandos

Die Befehle des Editors Ex sowie weitere komplexe Befehle lassen sich auch von Vi aus benutzen:

:*<excmd>*^M

Ex-Kommandos werden immer mit Enter abgeschlossen.

Ex-Kommandos

Die Befehle des Editors Ex sowie weitere komplexe Befehle lassen sich auch von Vi aus benutzen:

:*<excmd>*^M

Ex-Kommandos werden immer mit Enter abgeschlossen.

Beispiele

- ▶ Lösche eine Zeile: :d^M (Vi-Kommando: dd)
- ▶ Beende Vi: :q^M
- ▶ Beende trotz Änderungen: :q!^M
- ▶ Speichern und Beenden: :x^M (Vi: ZZ)

Bereiche

...

Bewegungskommandos

Die häufigsten Befehle sind diejenigen zum Bewegen des Cursors:

`h`, `j`, `k`, `l` links, runter, rauf, rechts

`w`, `b` nächstes/vorheriges Wort

`(`, `)` nächster/vorheriger Satz

`{`, `}` nächster/vorheriger Absatz

`0`, `$` Anfang/Ende der Zeile

Eingabe

Diese Kommandos wechseln in den **Eingabemodus**, aus dem man mit `^[]` (Escape) oder `^C` wieder herauskommt:

i, a füge vor/nach dem Cursor ein

I, A füge am Anfang/Ende der Zeile ein

o, O füge in einer neuen Zeile ein (unten/oben)

R überschreibe

c*<motion>* ersetze Zeichen, die *<motion>* überstreicht

cc, C ersetze Zeile/bis zum Zeilenende

Löschen und Kopieren

x, X lösche Zeichen unter/vor dem Cursor

d<motion> lösche Zeichen, die <motion> überspringt

dd, D lösche Zeile/bis zum Zeilenende

y<motion> kopiere Zeichen, die <motion> überspringt

yy, Y kopiere Zeile

p, P füge hinter/vor dem Cursor ein

Textpuffer

Man kann den Puffer (**Register**), den ein **Kopier-** oder **Einfügebefehl** benutzt, bestimmen, indem man ihm ein "*letter*" voranstellt.

Textpuffer

Man kann den Puffer (**Register**), den ein **Kopier-** oder **Einfügebefehl** benutzt, bestimmen, indem man ihm ein "*letter*" voranstellt.

Beispiele

- ▶ Ein Wort im Puffer **a** speichern: "ayw
- ▶ Das Wort hinter dem Cursor einfügen: "ap

Suchen

Suche nach einem **regulären Ausdruck** vorwärts:

```
/⟨pattern⟩[/]^M
```

... und rückwärts:

```
?⟨pattern⟩[?]^M
```

Zur Wiederholung der letzten Suche benutzt man `n` und `N`.

Suchen

Suche nach einem **regulären Ausdruck** vorwärts:

```
/⟨pattern⟩[/]^M
```

... und rückwärts:

```
?⟨pattern⟩[?]^M
```

Zur Wiederholung der letzten Suche benutzt man `n` und `N`.

Beispiel

```
/Wie.*ng^M
```

Ersetzen

```
:<range>s/<pattern>/<string>/<flags>^M
```

Das ersetzt jedes Vorkommen des regulären Ausdrucks *<pattern>* im Bereich *<range>* durch *<string>*.

Die Flags sind z.B. *i* (Groß-/Kleinschreibung ignorieren) oder *g* (mehrere Ersetzungen).

Beispiele

```
:s/Beispiel/Example/g
```

Rückgängig machen

In Vi kann man mit `u` (für **U**ndo) die letzte Änderung rückgängig machen. `U` macht die letzte Änderung in der aktuellen Zeile rückgängig.

Vim

In Vim gibt es unendliches Undo, und daher auch Redo: `^R`.

Dateioperationen

- :w[*file*]^M speichere Puffer in Datei
- :e *file* ^M lade Datei in den Editorpuffer
- :n^M, :N^M lade Datei nächste/vorherige Datei aus der Argumentliste
- :x[*file*]^M speicher Puffer und beende Vi

Einstellungen

Mit Einstellungen kann man die Anzeige oder das Verhalten von Vi beeinflussen.

Eine Liste aller Einstellungen kann man sich mit `:set allM` anzeigen lassen.

Den Wert einer Option erfährt man durch `:set option?M`.

Booleans `:set [no]optionM`

Strings `:set option=valueM`

Einige Einstellungen

Option	Default	Beschreibung
autowrite	off	Speichern beim Dateiwchsel
number	off	Zeilennummerierung anzeigen
autoindent	off	automatisches Einrücken neuer Zeilen
shiftwidth	8	Länge einer Einrückung
tabstop	8	angezeigte Länge eines Tabs
expandtab	off	füge Leerzeichen statt Tabs ein (Vim)
filetype	-	Dateityp (Vim)

Shell-Befehle ausführen

```
:!shcmd^M
```

Dieses Kommando führt den Befehl *shcmd* in einer Shell aus.

Dies ist vor allem nützlich, um Dateioperationen zu tätigen, ohne den Editor verlassen zu müssen.

Beispiele

```
▶ :!chmod 755 script^M
```

```
▶ :!ls -l^M
```

```
▶ :!./script^M
```

Filter

Es lassen sich außerdem **Zeilen** des Editorpuffers durch Shell-Befehle filtern, d.h. einige Zeilen werden auf die **Standardeingabe** eines Programmes gegeben und durch dessen **Ausgabe** ersetzt:

```
!<motion><shcmd>^M
```

<motion> ist dabei ein **Bewegungskommando** und *<shcmd>* der **Filter-Befehl**.

Beispiele

- ▶ Einen Absatz „formatieren“: `{!}fmt^M`
- ▶ Alle Zeilen alphabetisch sortieren: `gg!Gsort^M`

Makros

Definition

Ein Makro ist eine **Befehlssequenz** (ein Skript), die einen bestimmten Ablauf in einer Anwendung **automatisiert**.

In Vi gibt es zwei Typen von Makros:

Abkürzungen sollen helfen, häufige benutzte Ausdrücke schneller einzugeben.

Mappings sollen gebräuchliche Befehlsfolgen abkürzen.

Abkürzungen

:ab $\langle lhs \rangle \langle rhs \rangle ^M$

Danach wird jede Eingabe von $\langle lhs \rangle$ im Eingabemodus durch $\langle rhs \rangle$ ersetzt, solange $\langle lhs \rangle$ nicht Teil eines Wortes ist.

$\langle rhs \rangle$ kann auch Steuerzeichen (insbesondere auch Escape) enthalten.

Remapping von Tastatureingaben

```
:map <lhs> <rhs> ^M
```

Tippt man nun im Befehlsmodus *<lhs>*, wird stattdessen *<rhs>* ausgeführt.

Für Eingabemodus-Mappings benutzt man `:map!`.

Achtung: Kontrollzeichen müssen mit `^V` maskiert werden!

Remapping von Tastatureingaben

```
:map <lhs> <rhs>^M
```

Tippt man nun im Befehlsmodus $\langle lhs \rangle$, wird stattdessen $\langle rhs \rangle$ ausgeführt.

Für Eingabemodus-Mappings benutzt man `:map!`.

Achtung: Kontrollzeichen müssen mit $\^V$ maskiert werden!

Beispiele

In \LaTeX eine Betonung setzen:

```
Eingabe :map! EM \emph{ }^V^[i^M
```

```
Befehl :map EM lbi\emph{ }^V^[ea]^V^[^M
```

Textpuffer-Makros

Eine weitere Möglichkeit in einen Textpuffer zu schreiben zu schreiben, ist, die Eingaben in ihm „aufzunehmen“ (nur in Vim):

q<*letter*><*seq*>q

Mit @<*letter*> lassen sie sich dann ausführen.

Textpuffer-Makros

Eine weitere Möglichkeit in einen Textpuffer zu schreiben zu schreiben, ist, die Eingaben in ihm „aufzunehmen“ (nur in Vim):

q<*letter*><*seq*>q

Mit @<*letter*> lassen sie sich dann ausführen.

Beispiele

Aufnehmen Löschen des letzten Zeichens der Zeile: qq\$*x*jq

Ausführen Für die nächsten 20 Zeilen wiederholen: 20@q

Speichern von Makros und Einstellungen

Bei jedem Start von Vi werden die Ex-Kommandos aus der Datei `~/.exrc` ausgeführt.

In diese Datei kann man einfach `map`-Befehle eintragen.

Vim-Erweiterungen

Vim führt außer der `~/.exrc` auch die `~/.vimrc` aus.

Hat man die Dateityp-Erkennung aktiviert, so wird zusätzlich `~/.vim/ftplugin/<type>.vim` geladen.

Online-Hilfe

In Vim kann man mit `:help^M` die Hypertext-Hilfe aufrufen. Navigieren kann man mit `^]` (Link folgen) und `^T` (in der History zurück).

Zum Einstieg bietet sich das Programm `vimtutor` an, welches Vim in einem Tutorialmodus startet.

Scripting

Der Ursprüngliche Vi hat keine weiteren Scripting-Möglichkeiten.

Vim besitzt jedoch eine eigenen Skriptsprache und Bindings für die gängigsten Skriptsprachen.

Andere Vi-Varianten haben teilweise auch Unterstützung fürs Scripting.

Weblinks

- ▶ Vi Lovers Homepage, Linksammlung,
<http://thomer.com/vi/vi.html>
- ▶ Vim Homepage, mit vielen Tipps und Scripts,
<http://www.vim.org/>
- ▶ Wikipedia-Einträge,
<http://en.wikipedia.org/wiki/Vi>
<http://de.wikipedia.org/wiki/Vi>

Literatur

- ▶ Fred Buck: Vi Macros, Abbreviations and Buffers,
<http://soma.npa.uiuc.edu/docs/vi.macros>
- ▶ Vim Online Hilfe, in Vim :help
- ▶ NVi Manpage: `nvi(1)`
- ▶ Wikipedia Eintrag zu Terminals,
http://en.wikipedia.org/wiki/Computer_terminal
- ▶ Deutscher Wikipedia Eintrag zu Vi,
<http://de.wikipedia.org/wiki/Vi>

Escape-Sequenzen

- ▶ Um spezielle **Steuerbefehle** an ein Terminal zu senden, werden so genannte Escape-Sequenzen verwendet.
- ▶ Escape-Sequenzen werden auch beim betätigen **mancher Tasten** vom Terminal an die Anwendung gesendet.
- ▶ Escape-Sequenzen beginnen mit einem Escape-Zeichen (`^[]`).

Beispiele

- ▶ Linke Cursor-Taste: `^[[D`
- ▶ Bildschirm löschen: `^[[2J`

Ausführen von Make

In **Vim** gibt es extra einen Befehl, um `make` aufzurufen:

```
:make[ <target>]^M
```

Mit Hilfe von `:cc`, `:cn` und `:cp` kann man sich dann einen Fehler anzeigen lassen und zum nächsten oder vorherigen springen.

Das Ausgabeformat lässt sich je nach Compiler einstellen.