Kurzeinführung VIM

Index

- Syntax Highlighting

Vorbemerkung

Das vorliegende Dokument ist als Einführung in die Nutzung des hauptsächlich von Bram Moolenaar entwickelten, bildschirmorientierten Text-Editors **VIM** (**V**i **IM**proved) gedacht, der zur Familie der dem Unix-Standard-Editor **Vi** ähnlichen Editoren gehört.

Die meisten Vi-Kommandos funktionieren beim VIM identisch. Er bietet allerdings gegenüber dem alten Vi eine große Zahl von Erweiterungen und Verbesserungen, die die Arbeit deutlich vereinfachen und effektivieren.

Die nachfolgenden Darstellungen gelten größtenteils ab VIM 4.x, teilweise aber erst ab Vim 5.x bzw. Vim 6.x. Es empfiehlt sich generell, eine aktuelle VIM-Version zu nutzen, um von neuen Möglichkeiten profitieren zu können. Auf Grund der hohen Kompatibilität zu älteren Versionen ist ein Wechsel zu einer höheren VIM-Version erfahrungsgemäß problemlos möglich.

VIM steht für eine breite Palette von Betriebssystem- und Rechner-Plattformen zur Verfügung und wird sehr stark genutzt, auch als Ersatz für den Original-Vi. Bei heutigen Linux-Distributionen versteckt sich hinter dem Kommando **vi** bereits der VIM und nicht mehr wie früher der Vi-Clone Elvis.

Aufruf des Editors

```
vim [Optionen] [Dateinamen]
```

Die Wahl des Namens, unter dem man den Editor aufruft, kann dessen Zustand nach dem Start beeinflussen. Dieselbe Wirkung lässt sich durch entsprechende Optionen des Kommandos **vim** erreichen. Die folgende Tabelle zeigt einige typische Beispiele:

Kommandoname	äquivalente	Wirkung

	Option bei vim	
view	**-R**	Start mit gesetzter Option **readonly** Dadurch wird das unbeabsichtigte Überschreiben der editierten Dateien verhindert.
gvim	**-g**	Start der graphischen Oberfläche (GUI) Im Standardfall nutzt der VIM einen Text-Bildschirm und kann somit z.B. auf einer Linux-Text-Konsole sowie einem Terminal-Emulator wie **xterm** unter X-Windows eingesetzt werden.
ex	**-e**	Start im Ex-Modus, d.h. als zeilenorientierter Editor

Hinter dem Kommandonamen können Optionen und/oder Dateinamen angegeben werden. Optionen spielen in der Mehrzahl der praktischen Anwendungsfälle keine bzw. eine geringe Rolle. Wir wollen uns daher hier auf zwei Beispiele beschränken:

Kommandozeilen-Option	Bedeutung
-b	Option **binary** setzen Dieser Modus gestattet das Editieren beliebiger Binärdateien, allerdings steht kein Hexadezimal-Editor zur Verfügung.
-r	Recovery durchführen

Verzichtet der Nutzer auf die Angabe von Argumenten, dann startet VIM mit einem leeren **Puffer**, d.h., er lädt keine Datei zum Editieren. Ist die Dateiliste dagegen nicht leer, dann wird ihr erstes Element (das nicht selten das einzige überhaupt ist) nach dem Start des Editors automatisch zum aktuellen Dateinamen.

Sofern diese Datei im Dateisystem existiert, wird ihr Inhalt in einen Puffer geladen und in einem **Fenster** zum Editieren bereitgestellt. Eine noch nicht existente Datei legt der Editor allerdings nicht automatisch an. Die Erzeugung neuer Dateien erfolgt erst beim durch ein Nutzerkommando veranlassten Abspeichern des Inhalts eines Editor-Puffers.

Der Anwender hat die Möglichkeit, die Dateiliste in beiden Richtungen zu durchlaufen und die angegebenen Dateien nacheinander oder gleichzeitig in Fenstern darzustellen und so zu bearbeiten.

Sofern eine Initialisierungsdatei (z.B. **$HOME/.vimrc** bei Unix oder **$VIM_vimrc** bei Windows) existiert, wird sie während des Starts vom Editor gelesen und als Folge von Ex-Kommandos interpretiert, wobei auf den sonst erforderlichen Doppelpunkt vor dem Kommando verzichtet werden kann. Durch eine Initialisierungsdatei lassen sich ohne manuelle Eingriffe nutzerspezifische Einstellungen vornehmen.

Beispielsweise kann man durch

```
set number
```

festlegen, dass vor jeder Zeile des editierten Textes die Zeilennummer angezeigt werden soll.

Grundlegende Konzepte

Der VIM liest jede zu bearbeitende Datei in einen von ihm verwalteten **Puffer**, der sich entweder komplett im Hauptspeicher oder auch teilweise in einer Swap-Datei befindet. Alle Änderungen werden nur in diesem Puffer vollzogen. Die zu bearbeitende Datei wird so lange nicht verändert, bis der Nutzer das Abspeichern des Inhalts explizit anordnet.

Der Name der Swap-Datei entspricht unter Unix dem Muster **.*.sw?**. Anstelle des Sterns wird der Name der editierten Datei eingesetzt. Die erste und meist einzige Swap-Datei hat die Endung **.swp**. Für alle weiteren

Swap-Dateien wird der letzte Buchstabe der Endung schrittweise dekrementiert, also **.swo**, **.swn** bis **.swa**. Sofern letztere existiert, wird keine weitere Swap-Datei angelegt.

VIM ist in der Lage, mehrere Puffer gleichzeitig zu verwalten und diese in mehreren Fenstern auf dem Bildschirm darzustellen. Im Standardfall verhält sich der VIM allerdings wie der Original-Vi und verwaltet nur einen Puffer und genau ein Fenster, das mit Ausnahme der untersten Zeile den gesamten Bildschirm überdeckt.

Der untere, standardmäßig genau eine Zeile umfassende Bereich des Editor-Bildschirms dient der Anzeige von Status-Informationen (z.B. Warnungen und Fehlermeldungen, Nummer der aktuellen Zeile und Spalte, aktueller Modus) sowie der Eingabe von Kommandos im Kommandozeilen-Modus.

Sind mehrere Fenster aktiv, so wird jeweils an deren unterem Rand eine fensterbezogene Statuszeile dargestellt, die für das unterste Fenster allerdings optional unterdrückt werden kann.

Eine der Stärken des VIM ist seine **Recovery**-Fähigkeit. Konkret bedeutet das, dass auch bei Crash der Maschine oder des Editors der letzte Stand der Editor-Puffer weitestgehend bzw. vollständig wiederhergestellt werden kann, sofern nicht explizit die Pflege der bereits erwähnten Swap-Datei unterdrückt wurde. Um das Recovery einzuleiten, ist der VIM mit der Option **-r** aufzurufen, z.B.

```
vim -r intro.html
```

zur Wiederherstellung der Datei **intro.html**. Das Recovery stützt sich auf die Swap-Datei.

Alle Vi-Versionen unterscheiden verschiedene **Modi**. Der jeweils aktive Modus legt fest, wie die vom Anwender eingegebenen Zeichen interpretiert werden. Dieses Modus-Konzept, das den Vi sehr deutlich von den meisten anderen Editoren unterscheidet, ist für Anfänger meist gewöhnungsbedürftig, gestattet aber dem erfahrenen Bediener ein sehr schnelles und effizientes Arbeiten.

Der VIM kennt sechs Basis-Modi:

- **Kommando-Modus** (*Command mode*), auch **Normal-Modus** (*Normal mode*) genannt

  Alle Eingaben werden als Editor-Kommandos betrachtet. Es erfolgt also kein Einfügen der

eingegebenen Zeichen in den Editor-Puffer, wie das bei den meisten anderen Editoren der Fall ist. Nach dem Start befindet man sich in der Regel im Kommando-Modus.

- **Visueller Modus** (*Visual mode*)

  Dieser Modus dient der Markierung eines zusammenhängenden Teils des Textes, der dabei optisch hervorgehoben wird. Kommandos zur Cursor-Positionierung verändern die Grenzen der Markierung. Der hervorgehobene Bereich lässt sich auf verschiedene Weise bearbeiten, z.B. löschen, kopieren oder ersetzen.

- **Select-Modus** (*Select mode*)

  Er ermöglicht die beispielsweise von MS Windows her bekannte Arbeitsweise, einen zu verändernden Textabschnitt zu markieren und anschließend durch Eingabe des neuen Textes zu ersetzen.

  Anmerkung: Derselbe Effekt lässt sich auch sehr bequem mit Hilfe des visuellen Modus erzielen.

- **Einfüge-Modus** (*Insert mode*)

  Bis auf wenige Ausnahmen werden alle eingegebenen Zeichen in den Puffer eingefügt. Mit **Esc** verlässt man den Einfüge-Modus wieder.

- **Kommandozeilen-Modus** (*Command-line mode*)

  Er gestattet die Eingabe einzeiliger Kommandos am unteren Rand des Editor-Bildschirms. Dazu gehören:

Ex-Kommandos	**:**
Such-Kommandos	**/** und **?**
Filter-Kommandos	**!**

  Nach der Ausführung des Kommandos verlässt der Editor den Kommandozeilen-Modus wieder.

- **Ex-Modus** (*Ex mode*)

  Hier arbeitet der Editor nicht mehr bildschirmorientiert, sondern zeilenorientiert. Das hat u.a. zur Folge, dass man sich nicht mehr mit Hilfe eines Cursors durch den Text bewegen und so die aktuelle Position im Puffer festlegen kann. Außerdem wird auf dem Bildschirm nicht mehr automatisch der aktuelle Zustand eines Textausschnitts dargestellt.

  Dem Anwender stehen im wesentlichen die Kommandos des Kommandozeilen-Modus zur Verfügung. Der Editor verlässt den Ex-Modus allerdings nicht automatisch nach der Ausführung eines Kommandos, sondern erst dann, wenn dies der Bediener durch das Ex-Kommando **vi** bzw. **visual** explizit anweist.

Als Spezialfall des Einfüge-Modus ist der **Ersetzungs-Modus** (*Replace mode*) zu sehen, in dem die neu eingegebenen Zeichen existierende Zeichen des Textes überschreiben, es sei denn, es gibt an der aktuellen Cursor-Position keine alten Zeichen mehr. Dann wird eingefügt.

Die folgende Tabelle zeigt die wichtigsten Übergänge vom Kommando-Modus in die anderen Modi sowie die anschließende Rückkehr in den Kommando-Modus:

Wechsel zum	**durch**	**Rückkehr zum Kommando-Modus**
Einfüge-Modus	Kommandos zum Einfügen von Zeichen und Zeilen **i a I A c C o O**	**Esc**
Ersetzungs-Modus	**R**	**Esc**
Kommandozeilen-Modus	**: / ? !**	• **Esc** zum Abbruch • **NewLine** bzw. **LineFeed** (Enter-Taste) zur

		Ausführung des Kommandos
visuellen Modus	**v** (zeichenweise Markierung) **V** (zeilenweise Markierung) **Ctrl-V** (Markierung rechteckiger Blöcke)	• Abbruch durch die Wiederholung des einleitenden Kommandos oder auch **Esc** • Manipulation des markierten Blocks durch ein entsprechendes Kommando, z.B. **d**, **c** oder **~** (Tilde)
Select-Modus	**gh** (zeichenweise Markierung) **gH** (zeilenweise Markierung) **gCtrl-H** (Markierung rechteckiger Blöcke)	• Abbruch durch **Esc** • Manipulation des markierten Blocks durch ein entsprechendes Kommando. Die Eingabe eines druckbaren Zeichens bewirkt das Löschen des markierten Bereichs und den anschließenden

		Übergang in den Einfüge-Modus, wobei das eingegebene Zeichen an der aktuellen Cursor-Position in den Editor-Puffer eingefügt wird. Der Einfüge-Modus wird wie gewohnt durch **Esc** beendet.
Ex-Modus	**Q**	**vi** bzw. **visual**

Als Zwischenspeicher für verschiedene Daten verwaltet der VIM **Register**, von denen eines unbenannt ist. Alle anderen haben einen genau ein Zeichen langen Namen.

Beispielsweise der jeweils zuletzt gelöschte Textteil wird im unbenannten Register aufbewahrt, so dass er bei Bedarf an beliebigen Stellen desselben oder auch eines anderen Textes wieder eingefügt werden kann. Der Nutzer hat aber auch die Möglichkeit, bestimmte Textteile in benannten Registern zu speichern und später von dort wieder abzurufen. In der Mehrzahl der Fälle dürfte das unbenannte Register ausreichen.

Interessant ist auch die Eigenschaft des VIM, das zuletzt gegebene Einfüge- bzw. Löschkommando durch Eingabe eines Punktes (**.**) im Kommando-Modus an beliebigen Stellen wiederholen zu können. Dadurch lässt sich häufig der Aufwand enorm reduzieren.

Ebenfalls sehr wichtig ist der mehrstufige **Undo-/Redo**-Mechanismus. **Undo** gestattet, die letzten (vielleicht sehr komplexen oder katastrophalen) Änderungen schrittweise rückgängig zu machen. Die maximale Anzahl von Undo-Schritten wird durch die Option **undolevels** festgelegt und beträgt bei Unix standardmäßig 1000. Unerwünschte Undo-Schritte können mittels **Redo** schrittweise rückgängig gemacht werden. Dabei führt der Editor die durch Undo bereits rückgängig gemachten Operationen erneut aus.

Der VIM ist in der Lage, automatisch Backup-Dateien anzulegen. Das bedeutet, sofern eine Datei, in die ein Pufferinhalt abgespeichert werden soll, bereits existiert, wird sie umbenannt, bevor der Editor den Pufferinhalt auf den Datenträger schreibt. Dadurch wird das Überschreiben des existierenden Inhalts vermieden.

Falls die Option **backup** gesetzt ist, wird die umbenannte alte Datei erhalten, d.h. nicht automatisch gelöscht, nachdem der Pufferinhalt fehlerfrei gesichert werden konnte. In diesem Fall steht nach der Abspeicherung neben der neuen Version eines Dokuments auch noch dessen Vorgängerversion im Dateisystem zur Verfügung. Das kann sehr hilfreich sein, wenn man z.B. aus Versehen umfangreiche Modifikationen an einer Datei vorgenommen hat oder einfach nur mittels des Kommandos **diff** feststellen möchte, ob man auch genau die beabsichtigten Änderungen erzielt hat.

Generell ist zu den VIM-Kommandos anzumerken, dass viele von ihnen mehrfach hintereinander ausgeführt werden können, indem man vor dem eigentlichen Kommando eine ganze Zahl angibt, die besagt, wie oft das Kommando anzuwenden ist. Je nach Situation kann es sich bei dieser Zahl auch um eine Zeilenzahl oder -nummer handeln. Wird diese Zahl nicht angegeben, dann nimmt der Editor in der Regel eine Eins an. In der vorliegenden Beschreibung wird eine solche optionale Zahl durch durch das Präfix ***[n]*** symbolisiert.

Ex-Kommandos beziehen sich häufig auf einen **Zeilenbereich** (*range*). Dieser lässt sich auf unterschiedliche Weise angeben. Die genaue Syntax würde den Rahmen der vorliegenden Abhandlung sprengen. Sie sollte bei Bedarf der Online-Hilfe des VIM entnommen werden (**:help :range**).

Eine sehr einfache Möglichkeit der Beschreibung von Zeilen bzw. Zeilenbereichen besteht in der Angabe von Zeilennummern. Hier einige Beispiele:

12	Zeile 12
10,16	Zeilen 10 bis 16
. (Punkt)	aktuelle Zeile
$	letzte Zeile des Puffers

%	alle Zeilen (äquivalent zu **1,$**)

Die Angabe eines Bereichs (nachfolgend durch *[Bereich]* symbolisiert) ist optional. Wird sie weggelassen, beziehen sich manche Kommandos standardmäßig auf die aktuelle Zeile und andere auf den gesamten Puffer. Für die unten folgenden Kommandos wird der jeweilige Standard-Bereich angegeben.

Die Namen der meisten Ex-Kommandos müssen nicht voll ausgeschrieben werden, da der Editor die betreffenden Kommandos bereits eindeutig identifizieren kann, sobald die ersten *n* Zeichen ihres Namens angegeben wurden, wobei die Zahl *n* vom jeweiligen Kommando abhängt.

In der nachfolgenden Diskussion sowie im Online-Handbuch des VIM werden die optionalen Buchstaben eines Kommando-Namens in der Regel in eckige Klammern eingeschlossen. So besagt z.B. die Notation **co[py]**, dass das Kommando **copy** bereits nach Eingabe von **co** eindeutig spezifiziert ist (*n* hat hier also den Wert 2).

Der VIM unterstützt eine automatische Vervollständigung verschiedener Elemente der Ex-Kommandozeile. Dies betrifft die Namen von Kommandos, Optionen, Dateien und Tags. Durch Eingabe des über die Option **wildchar** festgelegten Zeichens (standardmäßig der Tabulator) wird das vor dem Cursor befindliche Muster (meist der Anfang eines Namens) so weit wie möglich vervollständigt. **Ctrl-D** listet alle Namen auf, die zu dem vor dem Cursor stehenden Muster passen. Weitere Details sind in der Online-Hilfe zu finden (**:help cmdline-completion**).

Positionierung des Cursors im Kommando-Modus

Die Cursor- und Bild-Tasten (**PageUp**, **PageDown**) sowie **Home** und **End** haben die übliche Wirkung. Im Unterschied zum Original-Vi sind diese Tasten auch im Einfüge-Modus nutzbar.

Nachfolgend wird mehrfach der Begriff **Whitespace-Zeichen** verwendet. Darunter versteht man Tabulatoren und Leerzeichen.

***[n]* G**	Sprung zu bestimmter Zeile Beispiele: **25G** für Zeile 25 **G** ohne *n* für die letzte Zeile
***[n]* |**	Sprung zu bestimmter Spalte Beispiele: **10|** für Spalte 10 **|** ohne *n* für die erste Spalte
0 (Zahl Null)	Sprung zum Zeilenanfang (Spalte 1)
$	Sprung zum Zeilenende (letzte Spalte)
[n] **w**	n Wörter vorwärts
[n] **b**	n Wörter rückwärts
[n] **W**	n WÖRTER vorwärts (WORT ist eine Folge von Nicht-Whitespace-Zeichen)
[n] **B**	n WÖRTER rückwärts (WORT ist eine Folge von Nicht-Whitespace-Zeichen)
[n] **H**	Cursor an den Anfang der n-ten vom oberen Rand gezählten Zeile eines Fensters positionieren
M	Cursor an den Anfang der in der Mitte eines Fensters befindlichen Zeile positionieren
[n] **L**	Cursor an den Anfang der n-ten vom unteren Rand gezählten Zeile eines Fensters positionieren
[n] **Ctrl-F**	n Seiten vorwärts blättern

[n] **Ctrl-B**	n Seiten rückwärts blättern
[n] **Ctrl-D**	n Zeilen vorwärts blättern; wenn n fehlt: standardmäßig eine halbe Seite blättern
[n] **Ctrl-U**	n Zeilen rückwärts rollen; wenn n fehlt: standardmäßig eine halbe Seite blättern
[n] **Ctrl-E**	n Zeilen vorwärts rollen
[n] **Ctrl-Y**	n Zeilen rückwärts rollen
[n] **zt**	Zeile n oder (falls n fehlt) die aktuelle Zeile an den oberen Fensterrand verschieben
[n] **zz**	Zeile n oder (falls n fehlt) die aktuelle Zeile in die Mitte des Fensters verschieben
[n] **zb**	Zeile n oder (falls n fehlt) die aktuelle Zeile an den unteren Fensterrand verschieben
%	sucht für bestimmte Text-Elemente, die unter oder hinter dem Cursor stehen, den bzw. die zugehörigen Partner Folgende Text-Elemente werden akzeptiert: • runde, eckige und geschweifte Klammern: **([{ }])** • Begrenzer von Kommentaren der Programmiersprache C: **/*** und ***/** • folgende Direktiven des C-Präprozessors: **#if**, **#ifdef**, **#else**, **#elif**, **#endif**
[n] **f *Zeichen***	nach rechts zum n-ten *Zeichen* gehen
[n] **t *Zeichen***	nach rechts bis vor das n-te *Zeichen* gehen
[n] **F *Zeichen***	nach links zum n-ten *Zeichen* gehen

[n] **T *Zeichen***	nach links bis hinter das n-te *Zeichen* gehen
[n] **;**	das letzte f-, F-, t- oder T-Kommando n Mal wiederholen
[n] **,**	das letzte f-, F-, t- oder T-Kommando n Mal in entgegengesetzter Richtung wiederholen
m *Marke*	Setzen einer Marke an der aktuellen Cursor-Position *Marke* ist ein einzelner Buchstabe, wobei sich Kleinbuchstaben immer auf den aktuellen Editor-Puffer beziehen, wogegen Großbuchstaben für alle aktuell geladenen Puffer global gültig sind. Das Setzen einer Marke verändert die Cursor-Position nicht. Beispiel: **ma** setzt Marke **a**
` *Marke* (Accent grave)	Sprung zu einer vorher gesetzten Marke
' *Marke* (Acccent aigu)	Sprung zum ersten Nicht-Whitespace-Zeichen der durch *Marke* markierten Zeile

---

Löschen und Einfügen von Zeichen

Das Präfix **[n]** gibt hier in der Regel an, wie oft die genannte Operation hintereinander auszuführen ist.

[n] **i**	Einfügen vor dem aktuellen Zeichen

	Beispiel: Die Zeichenfolge **50i=** gefolgt von der Escape-Taste bewirkt, dass vor dem aktuellen Zeichen 50 Mal das Zeichen **=** eingefügt wird.
[n] **a**	Einfügen nach dem aktuellen Zeichen
[n] **I**	Einfügen am Zeilenanfang, konkret vor dem ersten Nicht-Whitespace-Zeichen
[n] **A**	Anfügen am Zeilenende
[n] **x**	Zeichen unter dem Cursor löschen Statt **x** kann auch die **Del**-Taste genutzt werden.
[n] **X**	Zeichen links vom Cursor löschen
[n] **D**	Löschen ab Cursor bis zum Zeilenende sowie der n-1 folgenden Zeilen
[n] **C**	Ersetzen ab Cursor bis zum Zeilenende sowie der n-1 folgenden Zeilen Das Kommando C entspricht dem Kommando D mit anschließendem automatischem Wechsel in den Einfüge-Modus.
[n] **J**	Verbinden einer Zeile mit ihrem Nachfolger
[n] **d *Cursor-Bewegung***	Löschen des durch eine Cursor-Bewegung spezifizierten Textes
[n] **c *Cursor-Bewegung***	Ersetzen des durch eine Cursor-Bewegung spezifizierten Textes

Allgemein gilt, dass die zu den sog. Operatoren gehörenden Kommandos **d** und **c** denjenigen zusammenhängenden Teil des Textes löschen bzw. ersetzen, dessen Anfang und Ende durch die beiden vor

und nach der *Cursor-Bewegung* aktuellen Cursor-Positionen beschrieben wird.

Die Cursor-Bewegung kann durch ein beliebiges Kommando zur Positionierung des Cursors (z.B. auch ein Such-Kommando) erfolgen. Dabei unterscheidet man zwischen zeilen- und zeichenweisen Bewegungen. Letztere können den Endpunkt ein- oder ausschließen. Zeilenweise Bewegungen schließen dagegen den Endpunkt immer ein.

Die Details sowie einige Ausnahmen sind dem Online-Handbuch zu entnehmen (**:help deleting**).

Hier einige Beispiele für die Anwendung der beiden genannten Operatoren, wobei als Startpunkt immer die aktuelle Cursor-Position verwendet wird:

[n] **dw**	Löschen nach rechts bis vor den Anfang des folgenden Wortes
[n] **cw**	Ersetzen nach rechts bis zum Ende des aktuellen Wortes
[n] **db**	Löschen nach links bis zum Anfang des aktuellen Wortes
[n] **cb**	Ersetzen nach links bis zum Anfang des aktuellen Wortes
d0	Löschen nach links bis zum Zeilenanfang
c0	Ersetzen nach links bis zum Zeilenanfang

Durch Angabe eines numerischen Präfixes *n* vor den Kommandos **dw**, **cw**, **db** und **cb** lässt sich die jeweilige Operation *n* Mal hintereinander anwenden. So kann man beispielsweise durch das Kommando **3dw** drei Wörter nach rechts löschen, da das Kommando **dw** drei Mal hintereinander ausgeführt wird.

Hinweis:
Anstelle der Operatoren kann man auch den sehr bequemen visuellen Modus verwenden.

Wie bereits erwähnt, stehen im Einfüge-Modus die Cursor- und Bild-Tasten sowie **Home** und **End** zur Positionierung des Cursors zur Verfügung. Die Rückschritt-Taste (**BackSpace**) löscht im Einfüge-Modus immer ein Zeichen rückwärts, im Standardfall allerdings nur, wenn es sich um neu eingegebene Zeichen handelt. Nach Setzen einer speziellen Option (**:set bs=2**) kann die Rückschritt-Taste alle Zeichen inkl.

Zeilentrenner löschen.

Löschen und Einfügen von Zeilen

[n] **O** (Großbuchstabe O)	Einfügen einer Leerzeile oberhalb der aktuellen Zeile
[n] **o** (Kleinbuchstabe o)	Einfügen einer Leerzeile unterhalb der aktuellen Zeile
***[n]* dd**	Löschen der aktuellen Zeile
:*[Bereich]* d[elete]	Löschen eines Zeilenbereichs (Standard: aktuelle Zeile) Beispiele: **:10d** (Zeile 10) **:50,$d** (von Zeile 50 bis zum Textende)

Suchen und Ersetzen

Vorzugsweise sucht der VIM zyklisch, stoppt also nicht am Dateiende. Das Überschreiten des Dateiendes wird je nach Suchrichtung durch die Meldung

```
search hit TOP, continuing at BOTTOM
```

bzw.

```
search hit BOTTOM, continuing at TOP
```

bzw. eine entsprechende Übersetzung in eine andere Sprache im Status-Bereich angezeigt.

Suchmuster werden generell als **reguläre Ausdrücke** interpretiert. Normale Suchoperationen können aber ohne dieses Wissen ausgeführt werden. Besteht das Suchmuster z.B. nur aus Buchstaben und Ziffern, dann sucht der VIM genau nach der spezifizierten Zeichenkette.

[n] ***/Muster***	Vorwärtssuche eines Musters
[n] ***?Muster***	Rückwärtssuche eines Musters
[n] *	Vorwärtssuche des Wortes unter dem Cursor
[n] **#**	Rückwärtssuche des Wortes unter dem Cursor
[n] **n**	Wiederholung der letzten Muster-Suche
[n] **N**	Wiederholung der letzten Muster-Suche in umgekehrter Richtung
[n] **r**	Ersetzen des aktuellen Zeichens
[n] **R**	Umschalten in den Ersetzungs-Modus (*Replace*) Der eingegebene Ersetzungstext wird n-1 Mal eingegeben.
xp	Vertauschen eines Zeichens mit seinem Nachfolger
ddp	Vertauschen einer Zeile mit ihrem Nachfolger
[n] **~** (Tilde)	Umwandlung von Klein- in Großbuchstaben und umgekehrt
:[Bereich]* s[ubstitute]*/alt/neu/Modifikator [n]	Substitution innerhalb eines Zeilenbereichs (Standard: aktuelle Zeile) Dieses Kommando ersetzt im angegebenen Bereich das Auftreten des Musters *alt* gegen die Zeichenkette *neu*. Das Kommando ist auf Grund der Nutzung regulärer Ausdrücke

sehr mächtig.

Der Modifikator beeinflusst die genaue Wirkungsweise der Substitution. Hier eine Auswahl:

- **c**: jede Ersetzung ist vom Bediener zu bestätigen
- **g**: ersetze alle Vorkommen von **alt**, nicht nur die jeweils ersten einer Zeile
- **i**: Groß- und Kleinschreibung im Muster ignorieren
- **I**: Groß- und Kleinschreibung im Muster nicht ignorieren
- **p**: Ausgabe der kompletten Zeile, in der zuletzt substituiert wurde

Der optionale Zähler **n** (der hier nicht als Präfix, sondern als Suffix auftaucht) bewirkt, dass beginnend mit der letzten Zeile des Bereichs in n Zeilen die Substitutionen durchgeführt werden.

Wenn der Modifikator und der Zähler fehlen, kann man das Trennzeichen hinter dem Muster weglassen.

Das Trennzeichen zwischen den einzelnen Teilen des Kommandos muss nicht der Schrägstrich sein. Es sind bis auf folgende Ausnahmen alle Zeichen erlaubt:

- alphanumerische Zeichen
- Backslash (\)
- Doppelapostroph (")
- Pipe-Strich (|)

Die Verwendung eines alternativen Trenners ist speziell dann sinnvoll, wenn im Muster oder Modifikator selbst Schrägstriche

	vorkommen, weil man sich dann den Zusatzaufwand sparen kann, die Sonderbedeutung der Schrägstriche durch Voranstellen je eines Backslashs aufzuheben, wie folgendes Beispiel zeigt: `:s+/+//+` statt `:s/\//\/\//` Die Nutzung der Alternativ-Trenner macht die Ausdrücke auch übersichtlicher. Anwendungsbeispiele finden sich im Abschnitt über **reguläre Ausdrücke**.

Hinweis:
Die Suchmuster sowie die durch einen Doppelpunkt eingeleiteten Ex-Kommandos werden jeweils in eine eigene **History-Liste** eingetragen, um sie später in identischer oder modifizierter Form leicht wiederverwenden zu können. Das Durchlaufen der History-Listen geschieht mit Hilfe der Cursor-Tasten **Up** und **Down**. Innerhalb eines Eintrags kann man sich mit Hilfe der beiden anderen Cursor-Tasten (**Left** und **Right**) bewegen. **BackSpace** löscht das links vom Cursor stehende Zeichen. Buchstaben, Ziffern und Sonderzeichen werden an der Cursor-Position eingefügt.

Kopieren

Mit dem Operator **y** kann man Teile eines Puffers in ein Register kopieren, standardmäßig ins unbenannte Register. Hinter **y** folgt ein Kommando zur Cursor-Positionierung. Dadurch wird derjenige Teil des Pufferinhalts kopiert, dessen Anfang und Ende durch die beiden vor und nach der *Cursor-Bewegung* aktuellen Cursor-Positionen beschrieben wird.

***[n]* yy**	n Zeilen in das unbenannte Register kopieren

***[n]* Y**	Anmerkung: Lösch-Operationen, z.B. das Kommando **dd**, kopieren standardmäßig den jeweils gelöschten Text ebenfalls in das unbenannte Register.
y$	Kopieren des Zeilenrestes in das unbenannte Register
[n] **p** (Kleinbuchstabe p)	Einfügen des Inhalts des unbenannten Registers hinter der aktuellen Stelle
[n] **P** (Großbuchstabe P)	Einfügen des Inhalts des unbenannten Registers vor der aktuellen Stelle
:*[Bereich]* co[py] *nach*	Kopieren eines Zeilenbereichs (Standard: aktuelle Zeile) Der durch *Bereich* beschriebene Zeilenbereich wird hinter der durch *nach* spezifizierten Zeile in den Text eingefügt.

Unter Nutzung des visuellen Modus kann man sehr bequem kopieren, indem man den zu kopierenden Bereich markiert, durch das Kommando **y** ins unbenannte Register kopiert und mit den Kommandos **p** oder **P** an beliebigen Stellen wieder einfügt.

Anzeige des Status

Ctrl-G	aktuellen Dateinamen, Cursor-Position (sofern Option **ruler** nicht aktiv ist) und Datei-Status (**[Modified]**, **[readonly]**) anzeigen
Zahl **Ctrl-G**	wie **Ctrl-G**, aber den vollen Pfad des Dateinamens anzeigen. Falls die *Zahl* größer als 1 ist, wird zusätzlich die Puffer-Nummer ausgegeben.

gCtrl-G	Informationen über die aktuelle Cursor-Position anzeigen (logische und physische Spalte, Zeile, Zeichenposition im Text)

Anzeige und Einstellung von Optionen

Der VIM kennt drei Arten von Optionen:

boolesche Optionen	können nur ein- oder ausgeschaltet werden Beispiel: **ruler**
numerische Optionen	haben einen numerischen Wert Beispiel: **undolevels**
Zeichenketten-Optionen	haben eine Zeichenkette als Wert Beispiel: **helpfile**

Die folgende Tabelle enthält einige nützliche Kommandos:

:se[t]	Anzeigen aller Optionen, die von ihrem Default-Wert abweichen
:se[t] all	Anzeigen aller Optionen mit Ausnahme der Terminal-Optionen
:se[t] ***Option***	Setzen boolescher Optionen bzw. Anzeige des Wertes von numerischen und Zeichenketten-Optionen
:se[t] ***Option=Wert***	Setzen des Wertes von numerischen und Zeichenketten-Optionen

	Beispiele: **set undolevels=2000** **set helpfile=$VIM/doc/help.txt**
:se[t] *Option*?	Anzeige des Wertes einer Option Hinweis: Das abschließende Fragezeichen ist nur bei booleschen Optionen erforderlich. Beispiel: **set binary?**

Ausgewählte Optionen (zusammen mit ihren Abkürzungen, sofern solche existieren) und deren Bedeutung:

list	Repräsentation aller Tabulatoren durch **^I** (Ctrl-I) und Anzeige der Zeilenendezeichen als **$**
number **nu**	Voranstellen der Zeilennummer vor jede Textzeile
showmode **smd**	Anzeige des aktuellen Modus im Status-Bereich, sofern sich der Editor im Einfüge-, Ersetzungs- oder visuellen Modus befindet
undolevels **ul**	maximale Anzahl von Undo-Schritten
autoindent **ai**	automatisches Einrücken in neu erzeugten Zeilen im Einfüge-Modus
ruler **ru**	permanente Anzeige der aktuellen Zeilen- und Spaltennummer in der Statuszeile

magic	einige Zeichen erhalten bei Suchmustern und Ersetzungs-Zeichenketten eine Sonderbedeutung
ignorecase ic	Nichtbeachtung Groß-/Kleinschreibung bei der Suche
smartcase scs	Ignorieren der Option **ignorecase**, sofern das Suchmuster mindestens einen Großbuchstaben enthält
backup bk	beim Abspeichern eine Backup-Datei anlegen und erhalten (d.h. nicht löschen, nachdem der Pufferinhalt erfolgreich gesichert worden ist)

Durch Voranstellen von **no** vor eine boolesche Option wird diese ausgeschaltet, z.B. **:set nolist** oder **:set nonu**.

Abspeichern und Beenden

ZZ	Abspeichern des Editor-Puffers, sofern Änderungen vorgenommen wurden, und nachfolgendes Schließen des Fensters Sofern das letzte Fenster geschlossen wurde, wird der Editor beendet.
:w[rite]	Abspeichern des Editor-Puffers in die aktuelle Datei
:w[rite] ***Datei***	Abspeichern des Editor-Puffers in eine spezifizierte Datei
:*Bereich* w[rite] ***Datei***	Abspeichern des explizit angegebenen Zeilenbereichs in eine spezifizierte Datei
:e[dit] ***Datei***	Editieren einer anderen Datei, sofern der aktuelle Puffer nicht modifiziert wurde

q!	Beenden des Editors ohne Speichern Sofern der Text nicht modifiziert wurde, kann das Ausrufezeichen entfallen. Falls mehrere Fenster offen sind, wird nur das aktuelle Fenster geschlossen, der Editor aber nicht beendet.
:w[rite]! **:w[rite]!** ***Datei***	Überschreiben einer Read-Only-Datei

---

Auswahl sonstiger nützlicher Kommandos

:h[elp] **:h[elp]** ***Thema***	Online-Hilfe aktivieren
:ve[rsion]	Anzeige der Editor-Version
[n] **u**	Undo für das jeweils letzte Kommando (mehrstufig möglich)
[n] **Ctrl-R** **:red[o]**	Redo (Undo der Undo-Operation; ebenfalls mehrstufig möglich) Mit **:redo** kann man aber immer nur genau einen Schritt rückgängig machen.
U	Undo aller Änderungen, die zuletzt in einer Zeile vorgenommen wurden
[n] **.** (Punkt)	Wiederholung des letzten Einfüge- bzw. Lösch-Kommandos, wobei deren Zähler durch n ersetzt wird
:ar[gs]	Anzeige der Dateiliste

:files	
:n[ext]	Übergang zur nächsten Datei der Liste
:N[ext]	Übergang zur vorherigen Datei der Liste
:rew[ind]	Übergang zur ersten Datei der Liste
:la[st]	Übergang zur letzten Datei der Liste
:r[ead] ***Datei***	Einlesen einer Datei hinter die aktuelle Zeile
:0r[ead] ***Datei***	Einlesen einer Datei vor die erste Zeile eines Textes
:*[Bereich]* g[lobal]/*Muster*/*Ex-Kommando* **:*[Bereich]* v[global]/*Muster*/*Ex-Kommando***	wiederholte Anwendung eines Ex-Kommandos auf einen Zeilenbereich Sofern kein Bereich angegeben wird, beziehen sich diese beiden Kommandos auf den gesamten Text (wie bei **%** bzw. **1,$**). **g** wendet das jeweilige Ex-Kommando auf alle Zeilen des Bereichs an, die dem *Muster* entsprechen. **v** wendet dagegen das Ex-Kommando auf alle Zeilen an, die dem *Muster* nicht entsprechen. Anwendungsbeispiele finden sich im Abschnitt über **reguläre Ausdrücke**.
q ***Register***	Beginn der Aufzeichnung einer Tastenfolge in einem Register Beispiel: **qa** (Aufzeichnung in Register **a** starten)

q	Ende der Aufzeichnung einer Tastenfolge
@ *Register*	Ausführen des Inhalts eines Registers
Ctrl-A ***Zahl* Ctrl-A**	Addition einer Eins bzw. der angegebenen Zahl zu der Zahl unter bzw. der nächsten Zahl hinter dem Cursor
Ctrl-X ***Zahl* Ctrl-X**	Subtraktion einer Eins bzw. der angegebenen Zahl von der Zahl unter bzw. der nächsten Zahl hinter dem Cursor

---

Verwendung des visuellen Modus

Ein besonders elegantes Arbeiten ist mit dem visuellen Modus möglich. Mehrere der mit den oben gezeigten Kommandos erreichbaren Wirkungen lassen sich im visuellen Modus wesentlich einfacher realisieren.

Ein Textblock kann folgendermaßen markiert werden:

v	zeichenweise
V	zeilenweise
Ctrl-V	blockweise
gv	erneute Markierung des Bereichs, der zuletzt im visuellen Modus markiert war (sofern es einen solchen gibt)

Nach der Markierung eines Textblocks können auf ihm u.a. folgende Operationen ausgeführt werden:

!*Filter*	Filtern durch ein externes Programm Beispiel:

	!*sort* bewirkt eine Sortierung des markierten Bereichs durch das externe Kommando **sort**.
:	Übergang zum Ex-Modus. Dabei werden automatisch die beiden speziellen Marken **'<** und **'>**, die den Anfang bzw. das Ende des markierten Bereichs kennzeichnen, auf die Kommandozeile übernommen.
<	Verschieben nach links (um die durch die Option **shiftwidth** bestimmte Anzahl von Positionen)
>	Verschieben nach rechts (um die durch die Option **shiftwidth** bestimmte Anzahl von Positionen)
c C r R	Substituieren, d.h. Löschen des alten Inhalts und Übergang zum Einfügen
d D x X	Löschen
J	Verbinden zu einer Zeile
gq (in älteren VIM-Versionen auch **Q**)	Formatieren (Zeilen auf annähernd gleiche Länge bringen)
U	Umwandeln der Klein- in Großbuchstaben
u	Umwandeln der Groß- in Kleinbuchstaben
~ (Tilde)	Umwandeln der Groß- in Kleinbuchstaben und umgekehrt
y Y	Kopieren ins unbenannte Register
o	Vertauschen der Anfangs- und Ende-Markierung

Arbeit mit mehreren Puffern und Fenstern

:sp[lit]	horizontales Teilen des aktuellen Fensters Wird noch eine Datei als Argument angegeben (**:split *Datei***), so wird diese zur aktuellen Datei im neuen Fenster (analog Kommando **:e**).
:vs[plit]	vertikales Teilen des aktuellen Fensters Wird noch eine Datei als Argument angegeben (**:vsplit *Datei***), so wird diese zur aktuellen Datei im neuen Fenster (analog Kommando **:e**).
Ctrl-W Ctrl-W **Ctrl-W w**	Wechsel des aktuellen Fensters
:ba[ll]	pro Puffer genau ein Fenster auf dem Bildschirm darstellen
:wa[ll]	alle modifizierten Puffer ins Dateisystem schreiben, ohne den Editor zu beenden
:xa[ll]	alle modifizierten Puffer ins Dateisystem schreiben und Editor beenden
:qa[ll]	alle Fenster schließen und den Editor beenden, sofern kein Puffer modifiziert wurde

Da die Register nicht puffer-/fensterbezogen sind, bieten sie sich zum Übertragen von Text zwischen Puffern/Fenstern an. So könnte man in einem Fenster einen bestimmten Text in ein Register speichern, dann in ein anderes Fenster wechseln und den Registerinhalt dort an einer beliebigen Stelle wieder einfügen.

Syntax Highlighting

Ab Version 5.0 verfügt der VIM über ein durch Konfigurationsdateien anpassbares, flexibles **Syntax**

Highlighting. Für eine sehr große Anzahl von Sprachen bzw. Formaten (C, C++, Java, TeX, HTML, Shell, Maple, ...) sind bereits in der Standard-Distribution des VIM geeignete Syntax-Dateien enthalten, die vom Anwender unverändert genutzt bzw. bei Bedarf erweitert werden können.

Das Syntax Highlighting bietet die Möglichkeit, verschiedene Teile eines Textes durch unterschiedliche Farben oder Fonts darzustellen und so optisch hervorzuheben bzw. leichter unterscheidbar zu machen. Derartige Textteile können z.B. Schlüsselwörter einer Programmiersprache oder Zeichenfolgen, die einem bestimmten Muster entsprechen, sein. Syntax Highlighting kann beispielsweise einem Programmierer helfen, bestimmte lexikalische oder syntaktische Fehler in einem Quelltext einfacher und schneller zu erkennen.

:sy[ntax] on	automatisches Syntax Highlighting unter Verwendung der zur VIM-Distribution gehörenden Syntax-Dateien einschalten Das konkrete Highlighting-Format wird meist an Hand der Dateiendung und manchmal durch Analyse der ersten Zeile der Datei ermittelt.
:sy[ntax] off	Syntax Highlighting ausschalten
:so[urce] ***Syntax-Datei***	gezielte Auswahl eines bestimmten Highlighting-Formats Beispiel (für Vim 6.3): `:so $VIM/vim63/syntax/c.vim` Sofern die zur VIM-Distribution gehörenden Syntax-Dateien unter **$VIM/*Vim-Version*/syntax/*Format*.vim** stehen, was dem Standard entspricht, so aktiviert dieses Kommando für den aktuellen Puffer das Syntax Highlighting für die Programmiersprache C. Um den vollen Funktionsumfang zur Verfügung zu haben, empfiehlt es sich, zusätzlich (am besten vorher) das automatische Syntax Highlighting durch **:syntax on** einzuschalten. Bei Verwendung der GUI-Version des VIM (**gvim**) kann man das gewünschte

	Highlighting-Format über ein Menü auswählen.

Holger Trapp

letzte Modifikation: 25.10.2006