Der Editor Vi(m): Einführung, Tipps und Tricks

Version 1.41 — 02.01.2015

OSTC Open Source Training and Consulting GmbH
Thomas Birnthaler
E-Mail: tb@ostc.de
Web: www.ostc.de

Inhaltsverzeichnis

1 Einführung

1.1 Geschichte

Der *Vi* (*Visual Editor*) war der erste **bildschirmorientierte Editor** unter UNIX, vorher gab es nur zeilenorientierte Editoren wie *Ed* (*Editor*) oder *Ex* (*Extended Editor*) analog *Edlin* unter MS-DOS. Er wurde von **Bill Joy**, einem Studenten an der UCB (*University of California at Berkeley*) geschrieben (ehemaliger Miteigentümer der Firma SUN).

Der *Vi* ist hauptsächlich zum Erstellen von **ASCII-Texten** (z.B. Konfigurations-Dateien, *C*-Programmen, ...) gedacht, die von Textformatierern oder Compilern (*TEX, LATEX, troff, nroff, ...*) weiterverarbeitet werden.

Tatsächlich ist der *Vi* eine **Erweiterung des Zeileneditors** *Ex* (*Editor eXtended*), dieser wiederum ist eine Erweiterung des Zeileneditors *Ed* (*Editor*). Der Zeileneditor *Ex* ist immer noch hinter dem *Vi* verborgen, er kann während der Arbeit mit dem *Vi* jederzeit **temporär** für ein Kommando oder auch **permanent** aktiviert werden.

Eine moderne Variante des *Vi* ist der *Vim* (*Vi Improved*), den es auch in einer **grafischen Variante** *Gvim* (*Graphical Vim*) für viele Betriebssysteme (sogar Windows!) gibt (siehe www.vim.org).

1.2 Entwurfsprinzipien

Zur Entstehungszeit des *Vi* gab es keine PCs oder Arbeitsplatzrechner, sondern die Kommunikation mit einem zentralen UNIX-Rechner erfolgte über **Terminals**, die über relativ langsame serielle Verbindungen mit ihm verbunden waren. Sie besaßen außerdem keine Funktionstasten, geschweige denn eine Maus und die typische Bildschirm-Auflösung war 80 Zeichen $\times$ 24/25 Zeilen. Der *Vi* wurde daher so entworfen, dass:

- **Keine Funktionstasten benötigt**, sondern die normalen Tasten gleichzeitig für die Text- und für die Kommando-Eingabe verwendet werden. Er ist daher für beliebige Terminals geeignet.
- Auch über **sehr langsame Verbindungen** (z.B. üblich waren damals 300 Baud = 30 Zeichen/s, d.h. 66 s $\approx$ 1 min für den kompletten Aufbau eines 80$\times$25 Bildschirms) noch vernünftig bildschirmorientiert editiert werden kann.
- Eine **Vielzahl von Kommandos zum Bewegen** im Text vorhanden ist, d.h. mit wenigen Tastendrücken kann beliebig im Text navigiert werden.
- Die Editier-Kommandos mit den Bewegungs-Kommandos zur Auswahl des zu bearbeitenden Textes **kombiniert** werden können. D.h. mit wenigen Eingaben können beliebig große Textteile bearbeitet/ersetzt/gelöscht/gemerkt werden, *ohne sie vorher speziell zu markieren oder zu löschen und dabei mit dem Cursor herumzufahren*.
- Er **modus-orientiert** ist, d.h. die gleichen Tasten dienen zur Eingabe von Text und zur Eingabe von Editor-Befehlen.

Fast jeder Buchstabe (ohne Shift, mit Shift, mit Strg) löst im *Vi* **ein Kommando** aus. Es ist daher vorteilhaft, das **10-Finger-System** zu beherrschen.

- Jederzeit der **zeilenorientierte Ex-Modus** temporär (oder auch permanent) zur Verfügung steht, mit dem sich z.B. Such- und Ersetzungsaufgaben über Reguläre Ausdrücke formulieren lassen.

Derart **langsame Verbindungen** gibt es kaum mehr, Terminals besitzen heute **Cursor-** und **Funktionstasten** und sind auch per **Maus** bedienbar. Trotzdem sind die damaligen Entwurfsprinzipien aus folgenden Gründen heute noch nützlich:

- Der *Vi* ist **ressourcen-schonend** programmiert (Speicher, Rechenzeit, Netzwerklast).
- Unter der Voraussetzung, dass man das **10-Finger-System** beherrscht, können sämtliche Funktionen des Editors über die **normale Schreibmaschinen-Tastatur** erreicht werden, ohne die Finger von der Tastatur nehmen oder den Blick vom Bildschirm wenden zu müssen.
- Die **Geschwindigkeit** beim Erfassen und Bearbeiten von Texten ist gegenüber anderen (Funktionstasten-, Menü-, Maus-orientierten) Editoren deutlich höher (siehe Abschnitt 1.6 auf Seite 11), sofern man das Konzept der verschiedenen **Editor-Modi** verstanden und sich die (*wirklich gut gewählten!*) Namen zu den Kommando-Buchstaben eingeprägt hat.

Durch die Vielzahl der Kommandos und ihre Kombinationsmöglichkeiten ist der **Lernaufwand** für den *Vi* **etwas höher** als für übliche Editoren, der Aufwand dafür **lohnt** sich aber.

Gründe dafür sind:

- Es lohnt sich, für die **am häufigsten durchgeführte Tätigkeit** „Textbearbeitung“ ein komplexes, aber leistungsfähiges Werkzeug zu beherrschen, da der Lernaufwand gegenüber der Zeitersparnis bei ständiger Verwendung vernachlässigbar ist.
- Der *Vi* ist (neben dem (X)Emacs) immer noch **der Standard-Editor** unter UNIX/Linux, d.h. er ist auf wirklich *jedem* UNIX/Linux-System zu finden.
- *Vi*-Kommandos werden in vielen Programmen (z.B. `bash`, `ksh`, `more`, `less`, ...) sowie Spielen (z.B. `nethack`, `rogue`, ...) zur Steuerung verwendet.
- Der *Vi* wird standardmäßig von vielen UNIX/Linux-Kommandos (`crontab`, `ksh`, `more`, `visudo`, ...) automatisch aufgerufen, wenn sie Text bearbeiten lassen wollen (per Umgebungs-Variablen `EDITOR` / `VISUAL` änderbar).
- Mit dem *Vi* können auch **sehr große Dateien** (100 MByte) noch effektiv bearbeitet werden.

> Es gibt **keinerlei Zwang**, den *Vi* zu benutzen!
> Nehmen Sie bitte den Editor, der Ihnen am besten liegt.

Eine **Einarbeitung** in den *Vi* lohnt sich vor allem für:

- Heterogene UNIX/Linux-Umgebungen
- Remote-Administration von UNIX/Linux-Systemen
- UNIX/Linux-Systemadministratoren
- Programmierer unter UNIX/Linux
- Bedienung von „UNIX/Linux-Rettungssystemen“ (Notboot)
- Bedienung stark „abgespeckter“ UNIX/Linux-Spezialsysteme (z.B. Router)

1.3 Besonderheiten

- Der *Vi* ist ein **modus-orientierter Editor**, der *ausschließlich* über die normale Schreibmaschinen-Tastatur bedient wird, d.h. je nach dem aktuell aktiven Modus haben die (Buchstaben-)Tasten eine andere Bedeutung.

  > Gutes **Verständnis** für diese Modi und Bewußtsein für die **Übergänge** zwischen ihnen sind zur Beherrschung des *Vi* unabdingbar.

- Fast jeder **Vi-Kommando-Buchstabe** steht für den Anfangsbuchstaben eines **englischen Begriffs**, der seine Bedeutung beschreibt (im folgenden immer in der Form [`NAME`] angegeben). Diese systematische **mnemotechnische Art** der Abkürzung erleichtert das Einprägen der Kommandos ungemein.

  Abweichungen von dieser Regel gibt es nur sehr wenige und dann auch nur aus **ergonomischen Gründen**, z.B. bewegen die Tasten `hjkl` den Cursor nach links/unten/oben/rechts (hinter ihnen steht aber kein sinnvoller englischer Begriff) oder aus **sonstigen Zwängen** (keine Taste mit passendem Buchstaben mehr frei).

- Der *Vi* **unterscheidet Groß-/Kleinschreibung** (wie UNIX/Linux), daher kann der gleiche Kommando-Buchstabe einmal groß- und einmal kleingeschrieben völlig unterschiedliche Auswirkungen haben. In vielen Fällen sind diese Unterschiede zwar gering, aber doch wichtig (es lohnt sich, beide Versionen zu unterscheiden und ihre unterschiedliche Wirkung zu kennen — und es lohnt sich vor allem, die `CAPS LOCK`- oder Umstelltaste *nicht* zu arretieren ;-).

  > **CAPS-LOCK im *Vi* ist Unsinn**, da die klein- und die großgeschriebenen Kommandobuchstaben völlig unterschiedliche Wirkung haben.

1.4 Schreibweisen und Begriffe

In diesem Skript werden folgende Schreibweisen und Begriffe verwendet:

- [C=NAME] bezeichnet ein *Vi*-Kommando C oder ein gleichwertiges *Vi*-Ersatz-Kommando C sowie seinen englischen Namen NAME.
- [NAME] bezeichnet den englischen Begriff zu einem Kommando.
- <Strg-V> steht für die Steuerung/Control-Taste + V [verbose].
- ESC steht für die **Escape-Taste**.
- TAB oder -> stehen für die **Tabulator-Taste**.
- CR steht für die **Return-Taste**.
- BS steht für die **Backspace-Taste**.
- SPACE oder ␣ stehen für ein **Leerzeichen**.
- **Whitespace** (Leerraum) umfasst die Zeichen SPACE, TAB und CR.

1.5 Das Wichtigste in Kürze

1.5.1 Das wichtigste *Vi*-Kommando

Auch wenn man gar nichts vom *Vi* weiß, so sollte man zumindest wissen, wie man ihn — ohne Änderungen an einer Datei vorzunehmen — **sicher wieder verlassen** kann. Notwendig sind dazu die folgenden 5 Tastendrücke:

```
<ESC> : q ! CR
```

Im Einzelfall können natürlich auch weniger Tastendrücke nötig sein (minimal 3), aber diese Folge von 5 Tasten verlässt den *Vi* **ganz sicher**, egal in welchem Modus er sich gerade befindet.

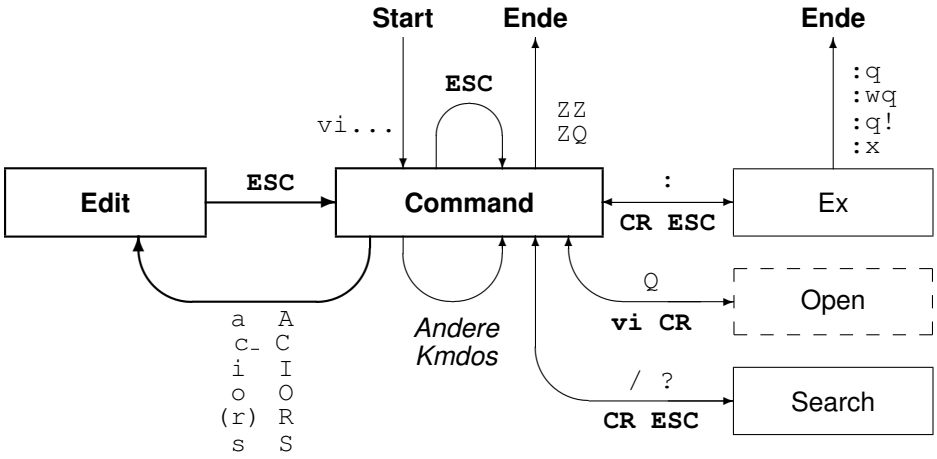
Erklärung: ESC schaltet in den Command-Modus (kann wegfallen, wenn man sich schon darin befindet), : schaltet in den Ex-Modus (notwendig), q=quit verlässt den Editor (notwendig), !=force erzwingt das Verlassen (auch wenn Änderungen an der Datei stattgefunden haben, kann wegfallen) und CR führt den Befehl schließlich aus.

1.5.2 Arbeitsmodi des *Vi*

Der *Vi* kennt folgende Arbeitsmodi (der *Vim* kennt noch mehr, siehe Seite 37), direkt nach dem Aufruf des *Vi* befindet man sich im **Command-Modus**. Zwischen den beiden (Haupt)Modi **Command** und **Edit** wechselt man während der Textbearbeitung **dauernd hin und her**.

Modus (deu)	Modus (eng)	Bedeutung
Kommando	Command	Eingabe von *Vi*-Kommandos
Edit	Insert/Replace	Eingabe von Text
Such	Search	Suchen nach Zeichenketten
Ex	Ex	Eingabe von *Ex*-Kommandos (*temporär*)
Open	Open	Eingabe von *Ex*-Kommandos (*permanent*)

Die folgende Grafik beschreibt die **Übergänge** zwischen diesen *Vi*-Arbeitsmodi. Durch Eingabe der bei den Übergangspfeilen stehenden Kommandozeichen wechselt man zwischen diesen Modi hin- und her. **Bei allen anderen Kommandos bleibt der *Vi* im Command-Modus** (in Abschnitt 5.1 auf Seite 37 befindet sich eine erweiterte Grafik für den *Vim*).



- Der **Command-Modus** löst durch Betätigen einer Taste *Vi*-Befehle aus. *In diesem Modus befindet man sich die **meiste Zeit***, durch `ESC` oder mit dem Abschluss bzw. Abbruch eines Kommandos kehrt man sofort wieder in diesen Modus zurück.
- Der **Edit-Modus** umfasst Einfügen, Überschreiben und Ersetzen von Text. *Nur in diesem Modus kann also Text bearbeitet werden!*
  - ▷ **Kommando** `c_` [`Change`] benötigt die Angabe eines Bewegungs-Kommandos zur Festlegung des **Textbereichs**, auf den es sich beziehen soll; daher der Unterstrich (siehe Abschnitt 3.4 auf Seite 20).
  - ▷ **Kommando** `r` [`Replace`] ist geklammert, da es nicht mit `ESC` zu beenden ist, sondern nach der Eingabe *eines* Zeichens **automatisch** in den Command-Modus zurückkehrt (Grund für diese Abweichung von der Regel ist, dass dieses Kommando häufig benutzt wird).
- Der **Ex-Modus** umfasst Lesen, Speichern und Verlassen des Vi; sowie viele weitere Befehle, für die keine freien Tasten mehr verfügbar waren, die aber auch seltener benötigt werden.
- Der **Search-Modus** erlaubt die Suche vorwärts und rückwärts im Text per Regulären Ausdrücken. Er wird oft auch zum schnellen Bewegen im Text eingesetzt.

- Der **Open-Modus** (auch *permanenter Ex-Modus*) erlaubt die Eingabe mehrerer Ex-Kommandos hintereinander ohne ständig die Taste `:` zu betätigen. Er ist gestrichelt dargestellt, da er nicht benötigt wird, allerdings leicht versehentlich mit dem Kommando `Q` [`Quit`] eingeschalten werden kann. Man sollte daher zumindest wissen, dass er mit dem Kommando `vi CR` wieder auszuschalten ist (*genau so zu tippen!*).
- Als zusätzliche Hilfe in diesem **Modus-Dschungel** kann man sich mit dem Kommando

  ```
  :set showmode
  ```

  in der **untersten Bildschirmzeile** anzeigen lassen, ob man sich im Edit-Modus zur Texteingabe oder in einem der anderen Modi befindet. Es erscheint dann nach der Eingabe eines der Kommandos `a A c C i I o O r R s S` eine der folgenden **Meldungen**:

  ```
  Append Mode
  Change Mode
  Insert Mode
  Open Mode
  Replace 1 Character
  Replace Mode
  ```

- Folgende Modi sind ebenfalls in der **untersten Bildschirmzeile** zu erkennen:

Anzeige	Bedeutung
`/...`	**Search-Modus** vorwärts (nach `/` bis `CR` oder `ESC`)
`?...`	**Search-Modus** rückwärts (nach `?` bis `CR` oder `ESC`)
`:...`	**Ex-Modus** (nach `:` bis `CR` oder `ESC`)
`:...`	**Open-Modus** (nach `Q` bis zur Eingabe von `vi CR`)

- Der **Command-Modus** ist (leider!) nur daran zu erkennen, dass sich weder in der untersten Bildschirmzeile einer der obigen Texte befindet, noch der Cursor in der untersten Bildschirmzeile nach dem Zeichen `/ ? :` steht.

Die **Rückkehr zum Command-Modus** ist durch folgende Befehle möglich:

- Die Kommandos `a A c C i I R s S` sind durch `ESC` **abzuschließen** (`r` nicht!), die Suchbefehle `/` und `?` sowie das *Ex*-Kommandos `:...` sind durch `CR` **abzuschließen** (alle anderen Kommandos nicht).
- Der Edit-Modus, der Search-Modus und der Ex-Modus können durch `ESC` **abgebrochen** werden (*der Open-Modus nur durch* `vi CR`).

1.5.3 Hinweise zur Taste `ESC`

Die `ESC`-Taste wird im *Vi* sehr häufig benötigt. Früher lag diese Taste auf den Tastaturen links unten, dort wo heute die `Strg`-Taste liegt. Sie war daher sehr leicht erreichbar und wurde aus **ergonomischen Gründen** ausgewählt. Heute liegt sie (leider) links oben (auf SIEMENS-Tastaturen extrem weit!) und ist nicht mehr so gut erreichbar. Aber man gewöhnt sich daran und merkt irgendwann gar nicht mehr, dass man sie relativ oft drücken muss.

- Als Alternative wäre auch möglich, die (im Vi sowieso unsinnige) `CAPS-LOCK`-Taste (oder noch besser eine der beiden `Windows`-Tasten ;-) mit `ESC` zu belegen. Diese Tasten sind auf jeder Tastatur leicht erreichbar.
- Die `ESC`-Taste sollte immer als „Abschluss“ einer Editier-Operation gedrückt werden, nicht als ihr „Anfang“.

> Vor jeder Kommandoeingabe `ESC` zu drücken, um den *Vi* sicher in den **Command-Modus** zu bringen, funktioniert zwar, ist aber umständlich und meist völlig unnötig.

- Behält man statt dessen den **aktuell gültigen Modus im Kopf** und führt nur die nötigen `ESC`-Betätigungen durch, kann der *Vi* viel flüssiger bedient werden.
- Mit `:set noerrorbells` kann das (gerade anfangs lästige) Piepsen beim überflüssigen Betätigen von `ESC` abgeschalten werden (;-).

1.5.4 Cursortasten im *Vi*

Auch wenn inzwischen meist die **Cursortasten** im *Vi* funktionieren, sollte man in der Lern- und Übungsphase *nur* die *Vi*-Kommandotasten (z.B. `hjkl`) zur Cursorbewegung verwenden.

> Man wird *nie* den **Übergang zwischen den Vi-Modi** verinnerlichen und immer mit einer gewissen **Unsicherheit** im *Vi* agieren, falls man von Beginn an die Cursortasten benutzt (später ist das kein Problem)!

Daher am besten in den ersten 1–2 Übungsstunden die Cursortasten **mit einem Stück Papier zudecken**. Außerdem sind die **Undo-** (`u U`), **Redo-** (`Strg-R`) und die **Repeat-Funktionen** (`. , ; n N`) bei Einsatz der Cursortasten nicht mehr sinnvoll einsetzbar, da sie sich auf genau das letzte Kommando beziehen.

Nach der Einarbeitung kann man die Cursortasten zwar verwenden, wird es wohl aber nicht mehr wollen, da die Bedienung über die Buchstaben schneller geht und man auf die praktische Undo- und Repeat-Funktionen nicht verzichten will, wenn man sie erst einmal schätzen gelernt hat.

1.5.5 Bildschirmaufbau des *Vi*

Als terminalorientierter Editor ist die Bildschirmdarstellung des *Vi* sehr spartanisch. Nur die unterste Zeile wird für **Statusmeldungen** und sonstige Informationen verwendet, der Rest des Bildschirms dient der Textdarstellung:

In der **Statuszeile** erscheinen Warnungen und Fehlermeldungen und weitere Informationen wie der gerade aktive Modus oder die Zeile/Spalte der aktuellen Cursorposition. Weiterhin springt der Cursor bei Eingabe von `:` für **Ex-Kommandos** und bei Eingabe von `/` bzw. `?` für die **Textsuche** in diese Zeile.

Ist eine Datei kürzer als der Bildschirm oder wird das Dateiende erreicht, erscheinen in der **1. Bildschirmspalte** `~`-Zeichen, um darauf hinzuweisen. Eine **leere Datei** wird also durch eine Reihe von `~`-Zeichen in der 1. Spalte dargestellt.

1.6 Mausorientierter Editor versus *Vi*

Dieser Abschnitt ist einer Mail aus dem Internet entnommen und ins Deutsche übersetzt worden. Text aus Vorgängermails ist mit > bzw. aus VorVorgängermails mit >> gekennzeichnet. Man kann die Mail als ziemlich **„religiöse" Diskussion** betrachten, d.h. sie beruht eher auf dem Glauben als auf Fakten, eine endgültige Wahrheit gibt es einfach nicht. Trotzdem macht sie zumindest eines deutlich: *Ein mausbasierter Editor ist nicht zwangsläufig am besten für jede Aufgabe geeignet.*

> Von Chris Torek
> In Artikel ... schreibt John Brunner:
>
> >> Ich bin mit dem „*Vi*" unter UNIX deutlich produktiver als mit irgendeinem
> >> der mausbasierten Editoren, denen ich auf dem Mac begegnet bin.
>
> In Artikel ... antwortet Pierce Wetter:
>
> > [Brüllendes Lachen]
> > Wenn ich meine Erfahrungen betrachte ... ist das die lächerlichste Behauptung,
> > die ich jemals gehört habe. Beim Programmieren ist meine häufigste Tätigkeit,
> > in der Datei herumzuspringen. Du kannst mir nicht weismachen, dass
> > das Zeigen und Klicken mit der Maus nicht bedeutend schneller geht, als auf
> > diese verdammten Cursortasten einzuhämmern.
>
> Doch ich kann und es ist auch tatsächlich so — zumindest für mich. Wenn das für dich nicht gilt, okay. (Um ein bißchen konkreter zu werden, ich „hämmere nicht auf diesen verdammten Cursortasten herum". Wenn ich z.B. von der Bildschirmmitte drei Zeilen nach unten an das Ende des siebten Wortes springen will, dann kann ich `Mjjj7E` eintippen. Ich persönlich brauche dazu etwa eine halbe Sekunde. Wenn ich einen Mac verwende,

brauche ich zum Finden der Maus, zum Zeigen, Klicken und wieder zur Tastatur zurückgehen etwa 4 Sekunden).

> Es ist richtig, dass du direkt zu einer Zeile mit einer bestimmten Nummer
> springen kannst, aber du kannst nicht einfach 5 Zeilen nach oben und 20
> Zeichen nach rechts springen.

`5k20l` erledigt das in weniger als einer Sekunde. Die einzige Schwierigkeit liegt darin, dass das visuelle Erscheinungsbild in einige zu tippende Zeichen übersetzt werden muss. Das kann man aber lernen, genauso wie man die Bedienung mit einer Maus erlernen muss.

> Mit einem mausbasierten Editor ist es viel einfacher, Cut & Paste zu
> verwenden (wenn du von mir geschriebenen Code betrachtest, wirst du verstehen,
> warum ich das mag — wenn ich z.B. eine for-next-Schleife brauche, kopiere ich
> sie mir einfach aus dem Code).

Das hängt ganz davon ab. Ich verwende die Maus auf dem Mac tatsächlich ab und zu für derartige Dinge, und zwar dann, wenn ich der Meinung bin, damit geht es schneller oder einfacher.

> Du muss jedesmal deine Hände von der Grundlinie der Tastatur nehmen, wenn
> du die `ESC`-Taste oder irgend eine andere `Strg`-Taste betätigen willst.

Eben nicht. Ich muss sie *genau dann* von der Tastatur wegnehmen, wenn ich die Maus verwenden will.

> Die Maus ist auf keinen Fall schlechter (außer deine Tippgeschwindigkeit ist
> unendlich groß).
> Eine Kleinigkeit möchte ich noch anmerken, ich verwende einen Trackball statt
> einer Maus ...

Ich möchte eigentlich eine Tastatur, eine Maus, einen Trackball, einen Lichtstift, ein Eingabetablett, einen Augenverfolger, Stimmeingabe und ich möchte von diesen Eingabegeräten jeweils das gerade sinnvolle verwenden. Tatsächlich ist der einzig vernünftige Ansatz, unabhängig vom konkreten Eingabegerät zu programmieren und die Verbindung zu jedem dieser Geräte zu erlauben.

> Nix für ungut.

Kein Problem.

1.7 Literatur

- Steve Oualline, *Vi IMproved — Vim*, New Riders.

  Umfassende und gut verständliche Beschreibung des *Vi*. Enthält jedes Thema zweimal: 1. als Einführung und 2. als Referenz.

- Kim Schulz, *Hacking Vim*, Packt Publishing.

  Ein Kochbuch um aus den neuen Eigenschaften des *Vim* das meiste herauszuholen.

- Reinhard Wobst, *vim GE-PACKT*, 2. Auflage, mitp.

  Sehr gut geeignet, um als alter „*Vi*-Hase“ endlich die vielen Erweiterungen des *Vim* kennen zu lernen.

- Linda Lamb, Arnold Robbins, *Learning the Vi Editor* oder *Textbearbeitung mit dem Vi-Editor*, O'Reilly & Associates.

  Die „Bibel“ zum *Vi*. Enthält eine kompakte Beschreibung des Editors und viele Beispiele für seine Anwendung. In der neuesten Auflage sind auch die *Vi*-Clones *Elvis*, *Nvi*, *Vile* und *Vim* berücksichtigt.

- Hewlett Packard Company, *The Ultimate Guide to the Vi and Ex Text Editors*, Benjamin/Cummings.

  Eine gründliche Einführung in die Bedienung des *Vi*.

- Arnold Robbins, *Vi Editor Pocket Reference*, O'Reilly & Associates.

  Eine tabellarische Kurzübersicht über die Kommandos des *Vi*. In der neuesten Auflage sind auch einige *Vi*-Clones berücksichtigt.

- Arnold Robbins, *vi kurz&gut*, O'Reilly.

  Übersichtliche Darstellung der Befehle des *Vi/Vim* (und einiger Clones).

2 Initialisierung

Die Initialisierung des *Vi* findet in folgender **Reihenfolge** statt (zum *Vim* siehe Abschnitt 5.2 auf Seite 38):

1. Beim Aufruf liest der *Vi* die in der **Umgebungs-Variablen** `EXINIT` definierten *Ex*-Kommandos ein und führt sie aus. Per `so`␣$file$ [`so=source`] kann hier z.B. auch eine Datei $file$ mit Kommandos eingelesen werden.

2. Ist im **Heimat-Verzeichnis** des angemeldeten Benutzers eine Datei namens `.exrc` (*ex ressource*) vorhanden, so wird sie beim Start des *Vi* eingelesen und die *Ex*-Kommandos darin ausgeführt.

3. Ist im **aktuellen Verzeichnis** eine Datei namens `.exrc` (`ex.rc` unter MS-DOS) vorhanden *und* die *Vi*-Option `exrc` (*read local ex ressource file*) aktiviert, so wird diese ebenfalls beim Start eingelesen und die *Ex*-Kommandos darin ausgeführt.

Der `:` vor den *Ex*-Kommandos ist in Initialisierungsdateien wegzulassen; **Kommentare** werden durch ein vorangestelltes `"` gekennzeichnet (*nicht* `#`*!*). Jede Initialisierungsdatei sollte mindestens folgende 4 Kommandos enthalten:

`set noerrorbells`	Bei Fehleingaben nicht piepsen
`set report=0`	Anzahl geänderter Zeilen in Statuszeile immer anzeigen
`set showmode`	Edit-Modus anzeigen
`set wrapscan`	Suche über Dateianfang/ende hinaus fortsetzen

Für **Text-Erstellung** sollte sie zusätzlich enthalten:

`set ignorecase`	Groß/Kleinschreibung bei der Suche ignorieren
`set wrapmargin=10`	Automatischer Zeilenumbruch 10 Zeichen vor Zeilenende

Für **Programmerstellung** sollte sie zusätzlich enthalten:

`set autoindent`	Einrückung aus vorheriger Zeile übernehmen (`Strg-D`=zurücknehmen, `Strg-T`=erneut einrücken)
`set showmatch`	Kurzer Sprung zur korrespondierenden öffnenden Klammer bei der Eingabe einer schließenden Klammer
`set shiftwidth=4`	Einrückungsbreite 4 Zeichen (`>>`, `<<`)
`set tabstop=4`	Tabulatorbreite 4 Zeichen

Beim *Vim* sollte sie zusätzlich enthalten:

`set nocompatible`	*Vi*-kompatibler Modus (immer ausschalten!)
`set hlsearch`	Suchergebnisse markieren (`highlight`)
`set incsearch`	Sofort während Eingabe suchen (`incremental`)
`set ruler`	Aktuelle Cursor-Zeile/Spalte in Statuszeile anzeigen (Lineal)
`set showcmd`	Unvollständiges Kommando in Statuszeile anzeigen (`command`)
`syntax on`	Syntaxcoloring einschalten (abhängig von Extension)

Optional kann die Initialisierungsdatei noch enthalten:

`set list`	Tabulatoren als `^I` anzeigen, Zeilenenden als `$`
`set number`	Zeilennummern am Zeilenanfang anzeigen

3 Kommandos

In diesem Abschnitt werden viele (aber bei weitem nicht alle!) Kommandos des *Vi* in folgende Gruppen zusammengefasst vorgestellt und Beispiele dazu angegeben:

- *Vi* aufrufen und verlassen
- Dateien einlesen und abspeichern
- Cursor bewegen
- Text editieren
- Text suchen und ersetzen
- Puffer verwenden
- Vermischtes

3.1 *Vi* aufrufen und verlassen

- Aufrufen

  Der *Vi* kann mit keinem, einem oder mehreren Dateinamen als Argument aufgerufen werden. Wird kein Dateiname `FILE` angegeben, so muss dieser beim Abspeichern

des Datei (hinter dem Kommando `:w` [`Write`]) angegeben werden. Ansonsten wird die erste angegebene Datei eingelesen (falls sie existiert) und zum Editieren angeboten (für Hinweise zum Editieren von mehr als einer Datei gleichzeitig siehe Abschnitt 3.7.2 auf Seite 28):

```
vi [OPTION...] [FILE...]
```

Die wichtigsten Kommandozeilen-Optionen `OPTION` sind:

Option	Name	Bedeutung
`-L`	`list`	Alle Dateien auflisten, die infolge eines System- oder Editorabsturzes gesichert wurden.
`-r` *file*	`recover`	Änderungen an Datei *file* nach System- oder Editorabsturz wiederherstellen und Datei weiter editieren.
`-R`	`readonly`	Read-Only Modus, editierte Dateien nicht speicherbar
`-t` *tag*	`tag`	Datei mit Definition von *tag* öffnen und zur Definition springen (→ Kommando `ctags`).
`+`	`skip`	Zur letzten Zeile der zu editierenden Datei springen.
`+`*n*	`skip`	Zur Zeile *n* der zu ...
`+/`*pattern*	`skip`	Zur 1. Zeile mit Muster *pattern* (Regulärer Ausdruck) der zu ...

- Häufig existiert auch ein Link namens `view`, mit dem der *Vi* direkt im **Read-Only-Modus** startet und ein Link namens `ex`, mit dem der *Vi* im **Open-Modus** (permanenter Ex-Modus) startet.
- Verlassen [`w=write`, `q=quit`, `x=exit`]

`:wq`	*Vi* mit Abspeichern verlassen
`:x`	*Vi* mit Abspeichern verlassen (speichert nur, wenn Text geändert wurde)
`ZZ`	*Vi* mit Abspeichern verlassen (keine Abkürzung, letzter Bst. im Alphabet)
`:q`	*Vi* ohne Abspeichern verlassen (nur möglich, falls Datei nicht geändert)
`:q!`	*Vi* ohne Abspeichern verlassen (trotz Änderung an Datei)
`ZQ`	*Vi* ohne Abspeichern verlassen (trotz Änderung an Datei)

3.2 Dateien einlesen und abspeichern

- Einlesen [`r=read`]

`:r` *file*	Datei *file* nach aktueller Zeile einfügen
`:`*n*`r` *file*	Datei *file* nach Zeile *n* einfügen
`:0r` *file*	Datei *file* am Dateianfang einfügen (*nach Zeile 0*)
`:$r` *file*	Datei *file* am Dateiende (`$`) einfügen
`:r!`*cmd*	Ergebnis des Betriebssystem-Kommandos *cmd* nach akt. Zl. einfügen

- Abspeichern [`w=write`]

`:w`	Aktuelle Datei speichern
`:w` *file*	Aktuelle Datei unter dem Namen *file* speichern (Datei ex. nicht)
`:w!`*file*	Aktuelle Datei unter dem Namen *file* speichern (Datei ex. bereits oder ist schreibgeschützt)
`:w>>`*file*	Aktuelle Datei an Datei *file* anhängen
`:10,20w` *file*	Zeile 10-20 in Datei *file* schreiben
`:10,20w>>`*file*	Zeile 10-20 an Datei *file* anhängen

3.3 Cursor bewegen

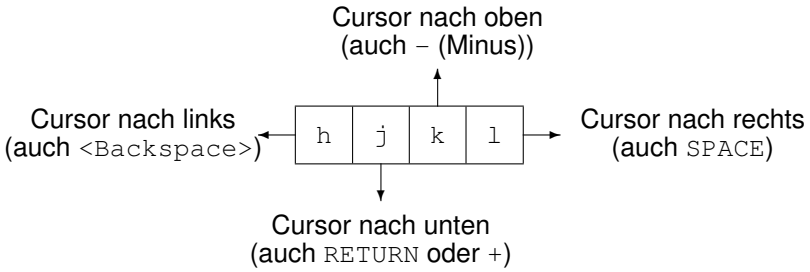
Alle Bewegungs-Kommandos *können davor* mit einem **Wiederholungsfaktor** versehen werden, sie wiederholen dann die Bewegung entsprechend oft. Sie sind nur im Command-Modus verwendbar (also vorher mit `ESC` dorthin wechseln). Die Kommandos `c d y < > !` *müssen danach* mit einem **Bewegungs-Kommando** zur Auswahl des zu bearbeitenden Textteils versehen werden.

3.3.1 Zeichen- und zeilenweise

Die Finger der rechten Hand liegen direkt auf den Tasten `(h)jkl` zur Cursorbewegung. Diese vier Tasten stellen ausnahmsweise keine mnemotechnische Abkürzung dar, sondern sind aufgrund ihrer schnellen Erreichbarkeit für die Cursorbewegung ausgewählt worden. Sie lassen sich aber über folgende **Eselsbrücke** leicht merken:

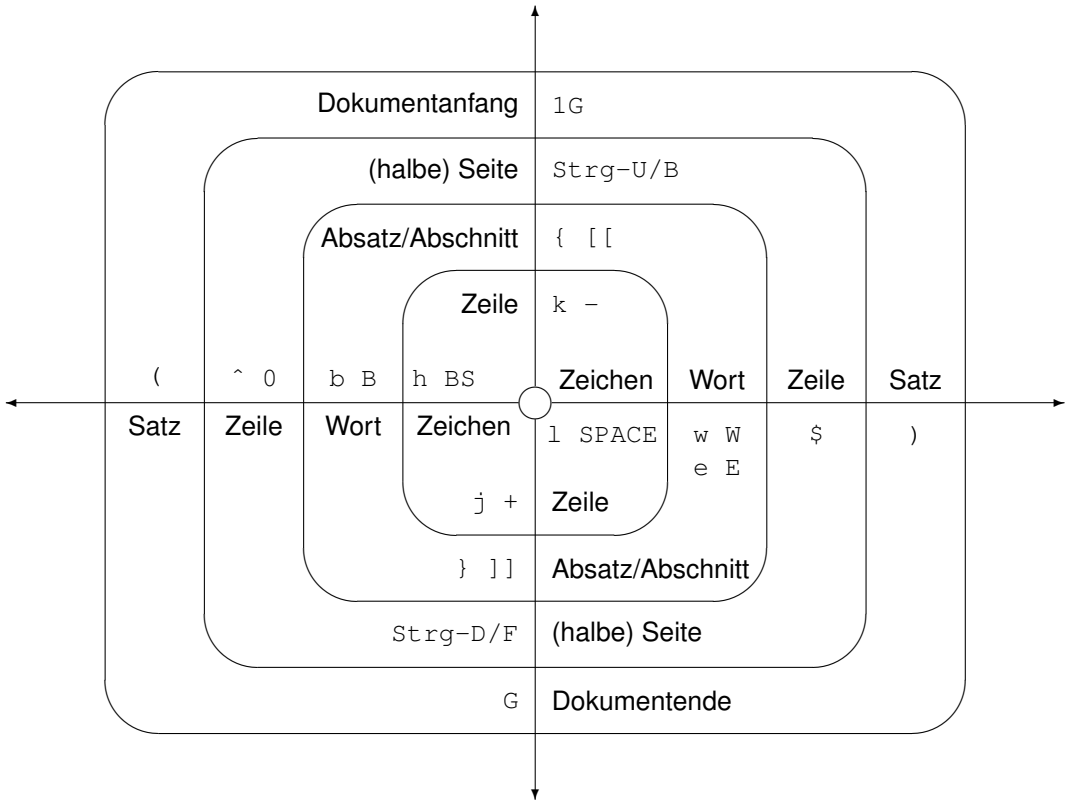
- `h` liegt *links*
- `j` zeigt nach *unten* (hat eine Unterlänge)
- `k` zeigt nach *oben* (hat eine Oberlänge)
- `l` liegt *rechts* (hat nichts mit `left` zu tun!)

> Auch wenn **Cursortasten** vorhanden sind, bitte daran gewöhnen, nur die Tasten `hjkl` zu verwenden (ihre Bedienung geht einfach schneller).



Hinweis: Bewegt man sich um mehr als ein paar Zeichen oder Zeilen, so gibt es eine **Vielzahl wesentlich effektiverer Möglichkeiten**, das Ziel anzusteuern, als den Cursor um eine Spalte oder Zeile zu bewegen.

> Den Cursor mit Cursortasten „rumschubsen“ machen nur *Vi*-Novizen. *Vi*-Experten „springen“ in großen Einheiten bzw. per Textsuche.



Tipp: Kommandos zum Bewegen des Cursors (oder auch beliebige andere Kommandos) können per `:map` auf vorhandene Cursor- oder Funktionstasten gelegt werden. Meist sind in der `.exrc`-Datei bereits entsprechende Vorbelegungen eingetragen.

3.3.2 Cursor allgemein bewegen

- Zeichenweise (Zeilenanfang/ende wird *nicht* übersprungen)

`h`	Ein Zeichen nach links (bis Zeilenanfang)
`BS`	Ein Zeichen nach links (bis Zeilenanfang)
`l`	Ein Zeichen nach rechts (bis Zeilenende)
`SPACE`	Ein Zeichen nach rechts (bis Zeilenende)

- In einer Zeile

`0`	An den Zeilenanfang springen (1. Spalte)
`^`	An den Zeilenanfang springen (1. Zeichen $\neq$ `SPACE/TAB`)
`$`	An das Zeilenende springen
`12|`	Zur Spalte 12 oder zur letzten Spalte springen (falls weniger Spalten vorhanden sein sollten)
`|`	An den Zeilenanfang springen (1. Spalte)

- Wortweise [`w=word`, `e=endword`, `b=backword`]

`w`	Zum Anfang des nächsten `Wortes` springen
`W`	Zum Anfang des nächsten `WORTES` springen
`e`	Zum Ende des aktuellen `Wortes` springen

`E`	Zum Ende des aktuellen `WORTES` springen
`b`	Zum Anfang des vorherigen `Wortes` zurückspringen
`B`	Zum Anfang des vorherigen `WORTES` zurückspringen
`3w`	Zum Anfang des über-über-nächsten `Wortes` springen

- ▷ Ein **Wort** ist eine Folge von Zeichen (Buchstaben und Ziffern) ohne Whitespaces, Satzzeichen (`.,;:!?`) oder Sonderzeichen darin.
- ▷ Ein **WORT** ist eine Folge von beliebigen Zeichen *ohne Whitespaces* darin (beim wortweisen Springen werden Zeilenanfang und -ende übersprungen).

- Zeilenweise [`n=next`, `p=previous`]

`j`	Zeile nach unten (Spalte möglichst beibehalten)
`+`	Zeile nach unten (zum 1. Zeichen der Zeile)
`CR`	Zeile nach unten (zum 1. Zeichen der Zeile)
`<Strg-N>`	Zeile nach unten (Spalte möglichst beibehalten)
`k`	Zeile nach oben (Spalte möglichst beibehalten)
`-`	Zeile nach oben (zum 1. Zeichen der Zeile)
`<Strg-P>`	Zeile nach oben (Spalte möglichst beibehalten)

- Satz-, Absatz- und Abschnittsweise

`(`	Zum vorherigen Satz (parenthesis = sentence)
`)`	Zum nächsten Satz
`{`	Zum vorherigen Absatz (brace = paragraph)
`}`	Zum nächsten Absatz
`[[`	Zum vorherigen Abschnitt (bracket = section/function)
`]]`	Zum nächsten Abschnitt
`[]`	Zum Ende des vorherigen Abschnitts
`][`	Zum Ende des nächsten Abschnitts

- ▷ Ein **Satz** endet mit einem der **Satzzeichen** `. ! ?`, wenn auf dieses mindestens 2 Leerzeichen folgen oder wenn es am Zeilenende steht (im *Vim* genügt 1 Leerzeichen).
- ▷ Ein **Absatz** endet mit einer **Leerzeile oder einem** *Troff*-**Absatz-Makro** `.IP .LP...` am Zeilenanfang (durch die Option `section` festgelegt, *Troff* ist ein UNIX-Programm zur Textformatierung analog LaTeX).
- ▷ Ein **Abschnitt** endet bei einer **öffnenden geschweiften Klammer**, wenn diese am Zeilenanfang steht (typisch für C/C++-Funktion/Struktur/Klasse) oder einem *Troff*-Abschnitt-Makro `.NH .SH...` am Zeilenanfang (durch die Option `paragraph` festgelegt).

- Zur korrespondierenden Klammer `( [ { < > } ] )` springen (sehr nützlich für Programmierer!).

`%`	Zur korrespondierenden Klammer springen

3.3.3 Bildschirmorientiert bewegen

- Zeilenweise [`e=expose next line`, `y=yield previous line`]

<Strg-E>	Bildschirm eine Zeile nach oben schieben (Zeile möglichst beibehalten)
<Strg-Y>	Bildschirm eine Zeile nach unten schieben (Zeile möglichst beibehalten)

- Auf dem Bildschirm [H=Home, M=Middle, L=Last]

H	Zur 1. Zeile des Bildschirmausschnitts springen
5H	Zur 5. Zeile des Bildschirmausschnitts springen
M	Zur Mitte des Bildschirmausschnitts springen
L	Zur letzten Zeile des Bildschirmauschnitts springen
3L	Zur 3. Zeile von unten des Bildschirmauschnitts springen

- Bildschirm verschieben [z=zone]

z CR	Aktuelle Zeile zur ersten Bildschirmzeile machen
z.	Aktuelle Zeile zur mittleren Bildschirmzeile machen
z-	Aktuelle Zeile zur letzten Bildschirmzeile machen

3.3.4 Seitenweise und absolut positionieren

- Seitenweise [F=Forward, B=Backward, D=Down, U=Up]

<Strg-F>	Eine Seite vorwärts blättern
<Strg-B>	Eine Seite rückwärts blättern
<Strg-D>	Eine halbe Seite vorwärts blättern
<Strg-U>	Eine halbe Seite rückwärts blättern

- Zeilen direkt anspringen [G=Go]

1G	Zum Dateianfang springen [:1 CR]
nG	Zu Zeile n springen [:n CR]
G	Zum Dateiende springen [:$ CR]
G$	Zum letzten Zeichen der Datei springen

3.3.5 Marken setzen und anspringen

Es gibt 26 **Marken** a-z, sie bleiben beim Dateiwechsel *nicht* erhalten, ebenso nicht beim Verlassen des *Vi* (beim *Vim* schon!).

ma	Aktuelle Zeile und Cursorposition mit Marke 'a markieren
'b	Zum Anfang der Zeile mit Marke 'b springen (' = Quote)
`z	Zum markierten Zeichen in Zeile mit Marke 'z springen (` = Backquote)
''	Zur Zeile vor dem letzten / ? G oder ''-Kommando springen (*hin und her*)
``	Zum Zeichen vor dem letzten / ? G oder ``-Kommando springen (*hin und her*)

> Bitte die Zeichen ` (nach links geneigter „Backquote“) und ' (senkrecht bzw. nach rechts geneigter „Quote“) unterscheiden!

Marken können im **Command-Modus** anstelle eines Bewegungs-Kommandos verwendet werden [m=mark].

`d`a`	Bis einschließlich Zeile mit Marke `'a` löschen
`c`b`	Bis einschließlich Zeile mit Marke `'b` ändern
`y`c`	Bis einschließlich Zeile mit Marke `'c` merken

Im **Ex-Modus** sind Marken als **Zeilenadressen** für Kommandos zum Suchen, Ersetzen, Verschieben, Löschen, ... verwendbar [`!=execute`, `!=force`] :

`:'a,'bc`	Textblock zwischen Marke `'a` und Marke `'b` ändern [`change`]
`:'a,'bd`	... löschen [`delete`]
`:'a,'by` x	... in Puffer x merken (x=`a-z`) [`yank/copy`]
`:'a,'b!`cmd	... an Betriebssystem-Kommando cmd übergeben
`:'a,'bw` $file$	... in Datei $file$ schreiben
`:'a,'bw>>`$file$	... an Datei $file$ anhängen
`:'a,'b>`	... um 1 `TAB` einrücken [`rightshift`]
`:'a,'b<<<`	... um 3 `TAB`s ausrücken [`leftshift`]

Alle Marken `'a-'z` mit ihren Positionen (Zeile + Spalte + Datei) auflisten:

```
:marks
```

3.4 Text editieren

Ganz egal, wieviel Text im *Vi* zur Bearbeitung selektiert wird, er ist immer durch beliebig viel (oder wenig) Text mit beliebig vielen Zeilen ersetzbar (*d.h. vor dem Einfügen von neuen Zeilen muss kein Platz für sie geschaffen werden*).

Wählt man einen Textbereich innerhalb einer Zeile zum Ändern aus (per `c_`, `C` oder `s`), so wird das Ende des ausgewählten Bereiches mit dem Zeichen `$` markiert (aus Geschwindigkeitsgründen, im *Vim* nicht mehr!).

Solange eine Zeile nicht mit `CR` abgeschlossen wurde, können mit `BS` die letzten eingegebenen Zeichen zurückgenommen und durch andere ersetzt werden. Die damit zurückgenommenen Zeichen zeigt der *Vi* aus Geschwindigkeitsgründen nach wie vor an, nur der Cursor wird über sie hinwegbewegt (*nicht dadurch verwirren lassen*, findet im *Vim* nicht mehr statt!).

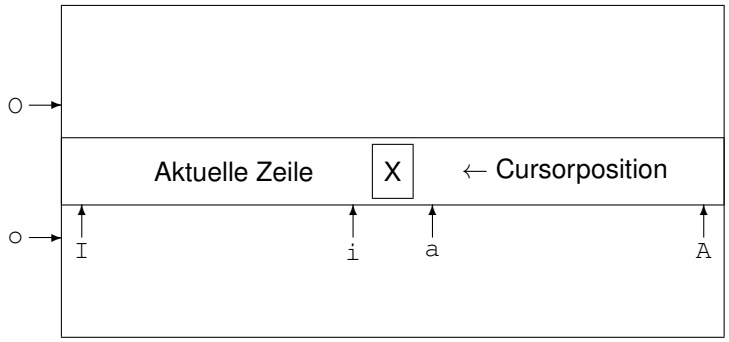
Mit `d` oder `x` gelöschte Textteile bleiben in **temporären Puffern** zur Wiederverwendung erhalten.

3.4.1 Einfügen, ersetzen, überschreiben und löschen

- Einfügen (mit `ESC` beenden) [`i=insert`, `a=append`, `o=open`].
  Je nachdem, ob man am Zeilenanfang, am Zeilenende, am Dateianfang oder am Dateiende steht, ist eines der Kommandos `i a O o` notwendig, um Zeichen am Zeilenanfang/ende oder Zeilen am Dateianfang/ende einfügen zu können. Die Kommandos `A` und `I` stellen Abkürzungen dar, die häufig benötigt werden.

`i`	Vor Cursor einfügen
`I`	Vor 1. Zeichen in Zeile einfügen [`^i`] (*nicht am Zeilenanfang* = [`0i`])
`a`	Nach Cursor anhängen
`A`	Am Zeilenende anhängen [`$a`]
`o`	Nach aktueller Zeile einfügen (*neue Zeile öffnen*)

`O` Vor aktueller Zeile einfügen (*neue Zeile öffnen*)



Tipp: Das „große Oh“ `O` fügt eine neue Zeile *vor* der aktuellen Zeile ein und ist mit `Shift` einzugeben (hebt bei alten Schreibmaschinen den Wagen nach oben). Analog fügt das „große Pe“ `P` gemerkten Text *vor* der aktuellen Cursorposition/Zeile ein und ist mit `Shift` einzugeben.

- Ersetzen (mit `ESC` beenden) [`c=change`, `n=next`, `w=word`, `G=go`]

  `c...` Textbereich ... ersetzen, z.B.:
  `c␣` ... aktuelles Zeichen [`s`]
  `c10␣` ... aktuelles + die nächsten 9 Zeichen
  `cw` ... aktuelles `Wort` (häufig notwendig!)
  `c3w` ... aktuelles `Wort` + die nächsten 2 `Worte`
  `c0` ... ab Cursor bis zum Zeilenanfang
  `c^` ... ab Cursor bis zum 1. Zeichen ($\neq$ `SPACE/TAB`) der Zeile
  `c$` ... ab Cursor bis zum Zeilenende [`C`]
  `c3j` ... ab aktueller Zeile + die nächsten 3 Zeilen
  `c/`abc ... ab Cursor bis zum 1. Auftreten der Zeichenkette abc *danach*
  `cn` ... ab Cursor bis zum *nächsten* Auftreten des letzten Suchmusters
  `cG` ... ab aktueller Zeile bis zum Dateiende
  `cc` ... aktuelle Zeile [`0c$`]
  `10cc` ... aktuelle Zeile + nächste 9 Zeilen
  `c%` ... ab Klammer unter Cursor bis zur korrespondierenden Klammer
  `c'`x ... ab aktueller Zeile bis zur Zeile mit Marke `'`x
  `C` ... ab Cursor bis zum Zeilenende [`c$`]

- Ersetzen (mit `ESC` beenden) [`s=substitute`]

  `s` Aktuelles Zeichen [`c␣`]
  `S` Aktuelle Zeile [`cc`]

- Überschreiben (nur `R` mit `ESC` beenden) [`r=replace`]

  `r`x Aktuelles Zeichen (*eines*) durch Zeichen x (*ohne* `ESC` *als Abschluß*)
  `R` Zeichen ab Cursorposition übertippen
  (nur aktuelle Zeile, beliebig viele neue Zeilen eingebbar)

- Löschen [`x=crossout`]

  `x` Aktuelles Zeichen [`d␣`, `dl`], entspricht `Delete`-Taste
  `10x` Aktuelles Zeichen + 9 Zeichen danach löschen
  `X` Zeichen *vor* Cursor [`hx`, `dh`], entspricht `BS`-Taste

- Löschen [d=delete, n=next, b=backword, G=go]

d...	Textbereich ... löschen, z.B.:
dh	...vorheriges Zeichen
d10h	...die vorherigen 10 Zeichen
dB	...WORT vor Cursor
d3B	...3 WORTE vor Cursor
d0	...ab Cursor bis zum Zeilenanfang
d^	...ab Cursor bis zum 1. Zeichen ($\neq$ SPACE/TAB) der Zeile
d$	...ab Cursor bis zum Zeilenende [D]
d2k	...ab aktueller Zeile + die vorherigen 2 Zeilen
d?abc	...ab Cursor bis zum 1. Auftreten der Zeichenkette abc *davor*
dN	...ab Cursor bis zum *vorherigen* Auftreten des letzten Suchmusters
d1G	...ab aktueller Zeile bis zum Dateianfang
dd	...aktuelle Zeile [0d$]
10dd	...aktuelle Zeile + nächste 9 Zeilen
d%	...ab Klammer unter Cursor bis zur korrespondierenden Klammer
d`x	...ab aktuellem Zeichen bis zum Zeichen mit Marke `x
D	...ab Cursor bis zum Zeilenende [d$]

3.4.2 Änderungen zurücknehmen

Der *Vi* macht Änderungen an einer Datei nicht sofort, sondern führt sie nur an einer **temporären Kopie** durch. Erst beim Abspeichern der Datei wird der alte Inhalt überschrieben. Eine **Sicherheitskopie** mit der alten Version der editierten Datei wird *nicht* angelegt (beim *Vim* möglich!). Folgende Kommandos stehen zur Rücknahme von Änderungen zur Verfügung [u=undo, e=edit, q=quit, w=write].

u	Die *letzte* Änderung rückgängig machen, ganz egal wie groß sie war (auch globale Ersetzungs-Kommandos)
U	*Alle* Änderungen *innerhalb einer Zeile* rückgängig machen, (*solange* die Zeile nicht verlassen wurde!)
Strg-R	*Letzte* zurückgenommene Änderungen wieder durchführen [Redo] (nur *Vim*!)
:e!	*Alle* Änderungen seit dem letzten Abspeichern rückgängig machen
:q!	*Vi* ohne Änderung an der editierten Datei verlassen
:w	Aktuellen Stand zwischenspeichern (*nicht mehr zurücknehmbar!*)

Der *Vi* kann leider nur **eine Änderung** zurücknehmen (U ist eine kleine Erleichterung dafür). Beim *Vim* können mit u=undo **beliebig viele Änderungen** bis zum ursprünglichen Zustand der editierten Datei zurückgenommen werden (U ist dort überflüssig). Zurückgenommene Änderungen können im *Vim* mit Strg-R [Redo] selbst wieder zurückgenommen werden (beliebig oft bis zum letzten Zustand).

Die Änderungen an einer Datei werden in der temporären Kopie vermerkt. Sollte die Verbindung zum Rechner unterbrochen werden, ohne dass die geänderte Datei FILE (im Verzeichnis DIR) abgespeichert wurde, so kann nach dem Wiederanmelden im gleichen Verzeichnis (cd DIR) mit dem Kommando [r=recover]

```
vi -r FILE
```

die Datei (bis auf die allerletzte Änderung) wiederhergestellt werden. Dies sollte allerdings sofort als erste Operation nach dem Anmelden erfolgen.

3.4.3 Gelöschten Text zurückholen

Der *Vi* kennt 9 **numerierte Löschpuffer** namens `"1-"9`, in denen automatisch die letzten 9 gelöschten Texte aufgehoben werden (der **unbenannte Puffer** ist mit dem Puffer `"1` identisch). Bei jeder Löschoperation geht der Inhalt von Puffer `"9` verloren, die Inhalte der Puffer `"1-"8` werden in den nächsten Puffer geschoben und Puffer `"1` wird mit dem gelöschten Text belegt. Die numerierten Puffer bleiben beim Wechsel der Datei erhalten, allerdings nicht beim Verlassen des *Vi* [`p=put`] (bleiben beim *Vim* doch erhalten!):

`p`	Letztes gelöschtes Element *nach* Cursor/aktueller Zeile einfügen (*analog* `o`).
`P`	Letztes gelöschtes Element *vor* Cursor/aktueller Zeile einfügen (*analog* `O`).
`"`n`p`	n-tes gelöschtes Element nach Cursor/aktueller Zeile einfügen.
`"3p...`	Inhalte der Löschpuffer 3,4,5,6 nach aktueller Zeile einfügen. (*das .-Kommando inkrementiert die Puffernummer automatisch!*)
`"2p.u.u`	Inhalte der Löschpuffer 2,3,4 nach aktueller Zeile einfügen und mit dem `u`-Kommando wieder entfernen (*das .-Kommando inkrementiert die Puffernummer automatisch!*)
`"3pu`	Nachsehen, was in Puffer 3 enthalten ist

- Ist **ein Stück einer Zeile** gelöscht worden, so kann es an beliebiger Stelle **innerhalb** beliebiger Zeilen eingefügt werden.
- Sind **eine oder mehr Zeilen** gelöscht worden, so kann dieser Zeilenblock **vor oder nach** beliebigen Zeilen eingefügt werden.

Die **Unterscheidung** zwischen diesen beiden Fällen erfolgt **automatisch**, je nachdem ob ein Zeilenvorschub im gelöschten Text enthalten ist oder nicht.

3.5 Text suchen und ersetzen

Im Suchmuster können (erweiterte) **Reguläre Ausdrücke** verwendet werden (siehe getrennte Beschreibung regex.pdf zu Regulären Ausdrücken). Kommt im Ersetzungsteil `&` vor, so ist es als `\&` zu schreiben, da es sonst für den Text steht, auf den das **Suchmuster passte**. Kommt im Ersetzungsteil `~` vor, so ist es als `\~` zu schreiben, da `~` im Ersetzungsteil für den Ersetzungstext des **letzten Suchmusters** steht.

3.5.1 In aktueller Zeile

Suchen von Zeichen in der gleichen Zeile [`f=find`, `t=to`]

`f`x	Sucht nächstes Zeichen x nach rechts (inklusive)
`3f`x	Sucht über-über-nächstes Zeichen x nach rechts (inklusive)
`F`x	Sucht vorheriges Zeichen x nach links (inklusive)
`2F`x	Sucht vor-vorheriges Zeichen x nach links (inklusive)
`t`x	Springt zu Zeichen *vor* nächstem Zeichen x nach rechts (exklusive)
`T`x	Springt zu Zeichen *nach* vorherigem Zeichen x nach links (exklusive)
`;`	Wiederholt Zeichensuche (*analog Punkt-Kommando*)
`,`	Wiederholt Zeichensuche in die andere Richtung (`Nicht-Shift+;`)

3.5.2 In ganzer Datei

Ob die Suche nach einem Text über Anfang oder Ende einer Datei hinweg erfolgt, entscheidet die Option `wrapscan`. Die Such- und Ersetzungsoperationen im **Ex-Modus** können per Zeilennummern oder Marken auf bestimmte Zeilenbereiche beschränkt werden. Mit `u=undo` kann die letzte Ersetzungsoperation vollständig zurückgenommen (und wiederhergestellt) werden (mit dem *Vim* beliebig viele).

- Suchen von Textstücken

/*abc*	Nach nächster Zeichenkette *abc* suchen (Richtung Dateiende)
?*abc*	Nach vorheriger Zeichenkette *abc* suchen (Richtung Dateianfang)

- Wiederholen der letzten Suche [`n=next`]

`n`	Letzte Suche wiederholen
`N`	Letzte Suche in umgekehrter Richtung wiederholen
`n.`	Letzte Suche und letzte Änderung wiederholen (beliebig oft)
`n.n.n.`	Letzte Suche und letzte Änderung 3x wiederholen
`//`	Letzte Suche wiederholen (Richtung Dateiende)
`??`	Letzte Suche wiederholen (Richtung Dateianfang)

- Suchen nach Zeilen- und Wortanfängen/enden

/^*abc*	*abc* am Zeilenanfang suchen
/*abc*$	*abc* am Zeilenende suchen
/^*abc*$	Zeile suchen, die nur *abc* enthält
/\<*abc*	*abc* am Wortanfang suchen
/*abc*\>	*abc* am Wortende suchen
/\<*abc*\>	Wort *abc* suchen

- Suchen und Ersetzen [`s=substitute`, `g=global`, `c=confirm`, `%=all lines`]

:s/*abc*/*xyz*/	Ersetzt in aktueller Zeile *die erste* Zeichenkette der Form *abc* durch *xyz* (`/` am Ende kann entfallen)
:s/*abc*/*xyz*/*n*	Ersetzt in aktueller Zeile die *n*-te Zeichenkette der Form *abc* durch *xyz*
:s/*abc*/*xyz*/g	Ersetzt in aktueller Zeile *alle* Textstücke der Form *abc* durch *xyz*
:%s/*abc*/*xyz*/g	Ersetzt *in ganzer Datei alle* Textstücke der Form *abc* durch *xyz*
:%s/*abc*/*xyz*/gc	Ersetzt *in ganzer Datei alle* Textstücke der Form *abc* durch *xyz* und *fragt vorher* jedesmal nach (der zu ersetzende Teil wird mit `^^^` darunter gekennzeichnet, bei `y=yes` wird ersetzt, bei `CR` oder `n=no` nicht)

- Suchen und Ersetzen von ganzen Worten (nicht von Wortteilen)

:%s/\<*abc*\>/*xyz*/g	Ersetzt in ganzer Datei *alle Wörter* der Form *abc* durch *xyz*, Wortteile werden nicht ersetzt.

3.5.3 Reguläre Ausdrücke

In den Regulären Ausdrücken der Such- und Ersetzungs-Kommandos `/ ? :s` können folgende **Metazeichen** verwendet werden:

Metazeichen	Bedeutung
c	Zeichen c selbst (kein Metazeichen)
.	1 beliebiges Zeichen
x*	0–∞ Wiederholungen des Zeichens x davor
^	Zeilenanfang
$	Zeilenende
\x	Folgendes Zeichen x *quotieren* (z.B. ein Metazeichen)
[abc], [a–z]	Menge von Zeichen (a–z = Zeichenbereich)
[^abc], [^a–z]	Negierte Menge von Zeichen (alle anderen)
\< \>	Wortanfang, Wortende
\(...\)	Zeichenkette merken (in \1..\9)
\1...\9	Gemerkte Zeichenkette einfügen
x\{m,n\}	m–n Wiederholungen des Zeichens x davor
x\{m,\}	m–∞ Wiederholungen des Zeichens x davor
x\{m\}	m Wiederholungen des Zeichens x davor (genau)
\u$x$	Nächstes Zeichen x in Großschreibung umwandeln
\l$x$	Nächstes Zeichen x in Kleinschreibung umwandeln
\U$xyz$\E	Alle Zeichen xyz in Großschreibung umwandeln
\L$xyz$\E	Alle Zeichen xyz in Kleinschreibung umwandeln

- & im Ersetzungsteil steht für den Text, auf den das **Suchmuster passt**.
  Kommt im Ersetzungsteil das Zeichen & selbst vor, so ist es als \& zu schreiben.
- ~ im Ersetzungsteil steht für den Ersetzungstext des **letzten Suchmusters**.
  Kommt im Ersetzungsteil das Zeichen ~ selbst vor, so ist es als \~ zu schreiben.

3.5.4 Weitere Möglichkeiten

- Spaltenbereiche vertauschen [s=substitute]

  `:s/^\(...\)\(..\)/\2\1/` Vertauscht die Spalten 1-3 mit den Spalten 4-5, der zu merkende Ausdruck wird mit \(...\) markiert, max. 9 gemerkte Ausdrücke können durch \1 bis \9 angesprochen werden.

  `:s/^\(.....\).../\1/` Löscht die Spalten 6-9, da die Spalten 1-9 durch die gemerkten Spalten 1-5 ersetzt werden.

- Leerzeichen ␣ am Zeilenende entfernen

  `:s/␣*$//` 0 oder mehr Leerzeichen am Zeilenende löschen
  `:s/␣␣*$//` 1 oder mehr Leerzeichen am Zeilenende löschen

- Leerzeichen ␣ am Zeilenanfang entfernen

  `:s/^␣*//` 0 oder mehr Leerzeichen am Zeilenende löschen
  `:s/^␣␣*//` 1 oder mehr Leerzeichen am Zeilenende löschen

- Leerzeichen ␣ am Zeilenanfang und Zeilenende entfernen

  `:s/^␣*\(.*[^␣]\)␣*$/&/` 0 oder mehr Leerz. am Zeilenanfang/ende löschen

- Leerzeilen löschen [g=global, d=delete]

  `:g/^␣*$/d`

- Jeweils 4 führende Leerzeichen ␣ am Zeilenanfang durch TAB (->) ersetzen

:%s/^␣␣␣␣/->/	Ersetzen der ersten 4 Leerzeichen durch ein TAB
:%s/^->␣␣␣␣/->->/	Ersetzen der nächsten 4 Leerzeichen durch ein TAB
:%s/^->->␣␣␣␣/->->->/	Ersetzen der nächsten 4 Leerzeichen durch ein TAB

- Alle Buchstaben verdoppeln

```
:%s/./&&/g
```

- Alle Buchstaben in Groß/Kleinschreibung umwandeln [u=uppercase, l=lowercase]

```
:%s/.*/\U&/g
:%s/.*/\L&/g
```

- Text am Zeilenanfang bis zum ersten : groß schreiben (nur falls die Zeile einen Doppelpunkt enthält) [E=endcase]

```
:%s/^\([^:]*\):/\U\1\E:/
:%s/^[^:]*:/\U&:/
```

- Ersten Buchstaben jeder Zeile in Groß/Kleinschreibung umwandeln [u=uppercase, l=lowercase]

```
:%s/.*/\u&/
:%s/.*/\l&/
```

- Soll die gleiche Ersetzung (1x) in mehreren Dateien gemacht werden, so kann der *Vi* mit allen Dateien aufgerufen werden. In der ersten Datei nach dem zu ersetzenden Text suchen, die entsprechende Ersetzung machen (möglichst unabhängig von der Umgebung) und davor oder danach folgendes Makro definieren (w=write, n=next, v=verbose):

```
map + :w <Strg-V> CR :n <Strg-V> CR n.
```

  Danach bis zur letzten Datei nur noch + drücken (und eventuell n. falls mehrere Ersetzungen in einer Datei nötig sein sollten).

- Zeilen mit best. Inhalt bearbeiten [g=global, v=vice versa, s=substitute]

:g/abc/cmd	cmd für alle Zeilen ausführen, die abc enthalten
:v/abc/cmd	cmd für alle Zeilen ausführen, die abc *nicht* enthalten
:g/abc/d	Alle Zeilen löschen, die abc enthalten
:v/abc/d	Alle Zeilen löschen, die abc *nicht* enthalten
:g/abc/s/def/xyz/	In den Zeilen, die abc enthalten, def durch xyz ersetzen
:v/abc/s/def/xyz/	In den Zeilen, die abc *nicht* enthalten, ...

3.6 Puffer/Register verwenden

3.6.1 Puffer/Register belegen und ausgeben

Die 26 **benannten Puffer/Register** "a-"z und die **automatischen Löschpuffer** "1-"9 bleiben zwischen Dateiwechseln erhalten, allerdings nicht beim Verlassen des *Vi* (im *Vim* schon). Der **unbenannte Puffer** bleibt beim Dateiwechsel nicht erhalten (im *Vim* schon) [y=yank/copy, d=delete, p=put].

- Unbenannter Puffer [y=yank/copy, w=word, d=delete, p=paste/put]

y...	Textbereich ... im unbenannten Puffer merken, z.B.:
yw	Aktuelles Wort merken (häufig benötigt)
yy	Aktuelle Zeile merken (häufig benötigt)
y^	Von Cursor bis Zeilenanfang merken
y$	Von Cursor bis Zeilenende merken
dW	Aktuelles WORT löschen und merken (häufig benötigt)
dd	Aktuelle Zeile löschen und merken (häufig benötigt)
p	Inhalt des unbenannten Puffers *nach* Cursor/aktueller Zeile einfügen (*analog* o)
P	Inhalt des unbenannten Puffers *vor* Cursor/aktueller Zeile einfügen (*analog* O)

- Benannte Puffer [y=yank/copy, d=delete, p=paste/put]

"a4yy	Aktuelle + die nächsten 3 Zeilen im Puffer "a merken
"Ayy	Aktuelle Zeile an Inhalt von Puffer "a *anhängen*
"b4dd	Aktuelle + die nächsten 3 Zeilen löschen und im Puffer "b merken
"Bdd	Aktuelle Zeile löschen und an Inhalt von Puffer "b *anhängen*
"zp	Inhalt des Puffer "z *nach* Cursor/aktueller Zeile einfügen (*analog* o)
"zP	Inhalt des Puffer "z *vor* Cursor/aktueller Zeile einfügen (*analog* O)
:'a,'byx	Bereich zwischen Marken 'a und 'b im Puffer "x ablegen
:'a,'bya	Funktioniert nicht, da y und ya das gleiche Kommando bezeichnen
:'a,'by a	Funktioniert

- Alle Register 'a-'z mit ihrem Inhalt auflisten:

```
:register
```

3.6.2 Pufferinhalt als Kommando ausführen

Komplizierte (*Ex*-)Kommandos oder mehrere *Vi*-Kommandos hintereinander sollten erst in eine Zeile eingetippt, dann in einen Puffer übernommen und von dort aus ausgeführt werden. Steuerzeichen muss ein <Strg-V> [verbose] vorangestellt werden, wenn sie im Kommando enthalten sind [@=apply]:

"ayy	Eine Zeile (mit Kommandos) im Puffer "a merken
"add	Eine Zeile (mit Kommandos) im Puffer "a merken und löschen
@a	Kommandos ausführen, die im Puffer "a stehen
@@	Letzte Puffer-Kommandosequenz wiederholen

3.7 Vermischtes

3.7.1 Zurücknehmen und Wiederholen

Folgende Kommandos sind **sehr nützlich**, weil gerade beim Bearbeiten von Text häufig eine Tätigkeit oder Suche **wiederholt** werden muss. Weiterhin ist es sehr angenehm, wenn man (versehentliche) Änderungen leicht und schnell **zurücknehmen** kann.

- Das **letzte Kommando** läßt sich mit `.` wiederholen (*egal welches*).
- Die **letzte Änderung** am Text läßt sich mit `u` [`u=undo`] wieder zurücknehmen (*ganz egal wie groß sie war*).
- Die **letzte zurückgenommene Änderung** am Text läßt sich mit `Strg-R` [`r=redo`] (*im Vim*) bzw. `u` [`u=undo`] (*im Vi*) wieder zurücknehmen, ganz egal wie groß sie war).
- Der **letzte gemerkte oder gelöschte** Text wird automatisch im unbenannten Puffer zwischengespeichert und kann mit `p/P` [`Put`] wieder hervorgeholt werden.
- Der **letzte Suchbefehl** läßt sich mit `n` bzw. `N` [`n=next`] wiederholen (auch mit `//` bzw. `??`).
- Der **letzte Suchbefehl** nach einem Zeichen in einer Zeile (`t=to`, `f=find`) läßt sich mit `;` bzw. `,` wiederholen (ähnlich zum Punkt-Kommando `.`).
- Der **letzte Such- und Ersetzungsbefehl** läßt sich mit `&` oder `:%s` [`s=substitute`] wiederholen.
- Das **letzte Betriebssystem-Kommando** läßt sich mit `:!!` wiederholen.

3.7.2 Editieren mehrerer Dateien

`%` steht für den Namen der **aktuellen Datei**, `#` für den Namen der **vorherigen Datei**.

- Hin- und Herschalten zwischen Dateien [`n=next`, `e=edit`, `f=file`, `p=previous`, `args=arguments`, `rew=rewind`]

`:n`	Schaltet zur nächsten Datei weiter (sofern aktuelle Datei nicht verändert)
`:n!`	Schaltet zur nächsten Datei weiter (egal ob aktuelle Datei verändert)
`:prev`	*Zur vorherigen Datei weiterschalten gibt es leider nicht im Vi* (*aber im Vim, hin- und her*)
`:e` $file$	Datei $file$ editieren (sofern aktuelle Datei nicht verändert)
`:e!`	Aktuelle Datei von vorne editieren (egal ob verändert oder nicht)
`:e!`$file$	Datei $file$ editieren (egal ob aktuelle Datei verändert)
`:e#`	Zur vorherigen Datei umschalten (sofern aktuelle Datei nicht verändert)
`:e!#`	Zur vorherigen Datei umschalten (egal ob aktuelle Datei verändert)
`<Strg-6>`	Analog `:e#`
`:e %.c`	An den Namen der aktuellen Datei `.c` anhängen und diese editieren (Nützlich, wenn man beim Aufruf die Endung vergessen hat)
`:f` $file$	$file$ zum Namen der aktuell editierten Datei machen

`:args`	Liste der beim Aufruf angegebenen Dateien anzeigen (die aktuelle Datei steht in `[...]`)
`:rew`	Erste der beim Aufruf angegebenen Dateien erneut editieren
`:rew!`	Analog (egal ob aktuelle Datei verändert)
`:n` *file*	Datei *file* editieren (sofern aktuelle Datei nicht verändert)
`:n!` *file*	Datei *file* editieren (egal ob aktuelle Datei verändert)
`:n *.c`	Alle C-Dateien im aktuellen Verzeichnis editieren (`:e *.c` *geht nicht*)

- Vorschlag zur Tastenbelegung

`:map # :args <Strg-V> CR`	Dateiliste anzeigen
`:map + :w <Strg-V> CR :n <Strg-V> CR`	Schreiben + zu nächster Datei
`:map - :w CR :rew <Strg-V> CR`	Schreiben + zu erster Datei

3.7.3 C-Quellcode editieren

- Zum vorherigen/nächsten **C-Funktionsanfang** springen

  `[[` Zum vorherigen Funktionsanfang springen (in Richtung Dateianfang)
  `]]` Zum nächsten Funktionsanfang springen (in Richtung Dateiende)

  Funktioniert nur dann, wenn die geschweifte Klammer vor dem Funktionskörper am Zeilenanfang steht und der Funktionskörper eingerückt ist. Beispiel:

```
int
FUNC(void)
{
    CODE
}
```

- Zum zugehörigen öffnenden bzw. schließenden **Klammernpartner** springen (Verschachtelung der gleichen Klammernart wird berücksichtigt)

  `%` Springt von `{ [ (` bzw. `) ] }` aus zum zugehörigen Klammernpartner

  Um damit z.B. die **Klammernstruktur einer C-Programmdatei** vollständig überprüfen zu können, sollten folgende Kommentare am Anfang bzw. Ende der C-Programmdatei stehen und *zu jeder Klammer* im Programm ein Partner vorhanden sein (z.B. als Kommentar). Kommt z.B. innerhalb von `printf` eine geschweifte Klammer vor, so sollte danach ein Kommentar mit einer geschweiften Klammer stehen.

```
/* {[( BRACE MATCHIN' HACK */
...
printf("{"); /*}*/
...
/* )]} BRACE MATCHIN' HACK */
```

- Steht der Cursor auf einer der Klammern `{ [ (` bzw. `) ] }`, so sind auf dem Textblock bis zur zugehörigen Klammer folgende Operationen möglich [`c=change`, `d=delete`, `y=yank/copy`].

`c%`	Ändern
`d%`	Löschen (und im temporären Puffer merken)
`y%`	Merken im temporären Puffer
`>%`	Einrücken um einen Tabulator
`<%`	Ausrücken um einen Tabulator

- Text ein- oder ausrücken [`shift left/right`]

`>>`	Aktuelle Zeile um einen Tabulator einrücken
`10>>`	Nächsten 10 Zeilen um einen Tabulator einrücken (inkl. aktueller Zl.)
`<<`	Aktuelle Zeile um einen Tabulator ausrücken
`20<<`	Nächsten 20 Zeilen um einen Tabulator ausrücken (inkl. aktueller Zl.)

3.7.4 Zeilenbereiche ansprechen

Folgende **Zeilenadressen** sind bei der Angabe von Zeilen oder Zeilenbereichen zu einem *Ex*-Kommando möglich. Anfang und Ende eines Zeilenbereiches werden durch `,` getrennt. Zu einer Adresse kann auch noch ein konstanter Wert addiert oder von ihr subtrahiert werden.

Adresse	Bedeutung
`1`	Erste Zeile
`.`	Aktuelle Zeile
n	n-te Zeile
`'`x	Zeile mit Marke `'`x
`$`	Letzte Zeile
$/regex/$	Zeilen auf die Suchmuster $regex$ passt
n`,`m	n-te bis m-te Zeile
`'`x`,'`y	Zeile von Marke `'`x bis Marke `'`y
`%`	Alle Zeilen (entspricht `1,$`)
$/regex1/$`,`$/regex2/$	Zeilen auf die Suchmuster $regex1$ passt bis Zeilen auf die Suchmuster $regex2$ passt

`:20,30d`	Von Zeile 20 bis Zeile 30 löschen
`:.,$c`	Ab aktueller Zeile bis Dateiende ändern
`:1,.w` $file$	Von Dateianfang bis aktueller Zeile auf Datei $file$ schreiben
`:1,$s/^/#␣/`	Gesamte Datei auskommentieren (`# SPACE` am Zeilenanfang)
`:%s/^#␣//`	Kommentar in gesamter Datei wieder entfernen
`:'a,'by␣z`	Von Marke `'a` bis Marke `'b` im Puffer `"z` merken
`:.-4,.+3d`	Von aktueller Zeile -4 bis aktueller Zeile $+3$ löschen

3.7.5 Betriebssystem-Kommandos ausführen

- Mit `<Strg-Z>` kann man den *Vi* in den **Hintergrund** schicken und dann ganz normal mit dem Betriebssystem arbeiten. Mit

  ```
  jobs
  ```

  können die vorhandenen Hintergrundprozesse aufgelistet werden. Hat der *Vi*-Prozeß z.B. die Nummer 3, so kann er mit [`fg=foreground`]

```
fg %3
```

wieder in den **Vordergrund** geschickt werden. Der letzte unterbrochene Prozeß kann einfach mit

```
fg
```

wieder in den Vordergrund geschickt werden.

> Bitte nicht vergessen, dass eine *Vi*-Sitzung in den Hintergrund geschickt wurde. Bricht die Verbindung ab oder editiert man die gleiche Datei noch einmal, zählt der letzte durchgeführte Speichervorgang.

- Ein Betriebssystem-Kommando `CMD` kann mit [`!=execute`]

  ```
  :!CMD
  ```

  direkt vom *Vi* aus aufgerufen werden. Beispiel [`co=checkout`, `l=lock`]:

  ```
  :!co -l %
  ```

  um die aktuell bearbeitete, aber nur lesend ausgecheckte Datei nachträglich zu locken (wenn sie mit RCS=Ressource Control System verwaltet wird). (Das Schreiben muss dann allerdings mit `:w!` erfolgen, da sich der *Vi* das bisher vorhandene *Read-Only-Flag* gemerkt hat. Mit `:e` kann anschließend die Datei ohne das Read-Only-Flag editiert werden).

- Das Ergebnis des Betriebssystem-Kommandos `CMD` kann mit [`r=read`]

  ```
  :r!CMD
  ```

  direkt *nach* der aktuellen Zeile eingefügt werden. Beispiel:

  ```
  :r!ls *.c | sort
  ```

  um eine sortierte Liste aller C-Dateien im aktuellen Verzeichnis in den Text einzufügen.

3.7.6 Optionen

Optionen haben einen **(Lang)Namen** und meist eine **Abkürzung**, sie beeinflussen das Verhalten des *Vi*. Es gibt **Schalter-Optionen** mit 2 Werten (`yes` und `no`), **numerische Optionen** mit einem Zahlenwert und **Textoptionen** mit einem Textwert (spezielle Optionen des *Vim* sind in Abschnitt 5.3 auf Seite 39 zu finden).

- Einschalten, Ausschalten, Setzen und Anzeigen

`:set` *opt*	Option *opt* einschalten (Schalter)
`:set no`*opt*	Option *opt* ausschalten (Schalter, `no`=nein)
`:set` *opt*=*n*	Option *opt* auf Wert *n* setzen (Numerisch)
`:set` *opt*=*text*	Option *opt* auf Text *text* setzen (Text)
`:set`	Alle Optionen anzeigen, die *nicht* den Standardwert haben
`:set all`	Alle Optionen anzeigen
`:set` *opt*`?`	Wert von Option *opt* anzeigen

- Wichtige Optionen zur **Textbearbeitung**:

Name	Std	Bedeutung
`ignorecase(ic)`	`no`	Groß/Kleinschreibung bei Suche ignorieren
`list`	`no`	Tabulatoren als `^I` anzeigen, Zeilenenden als `$`
`number(nu)`	`no`	Zeilennummer am Zeilenanfang anzeigen
`report`	`5`	Ab 5 Zeilen Anzahl geänderter Zl. in Statuszeile anzeigen (0 = Bei beliebiger Anzahl in Statuszeile anzeigen)
`showmode`	`no`	Aktuellen *Vi*-Modus in Statuszeile anzeigen
`wrapmargin(wm)`	`0`	Automatischer Zeilenumbruch *n* Zeichen vor Zeilenende (0 = Kein automatischer Zeilenumbruch)
`wrapscan(ws)`	(`yes`)	Suche über Dateianfang/ende hinaus fortsetzen

- Wichtige Optionen für **Programmierer**:

`autoindent(ai)`	`no`	Einrückung aus vorheriger Zeile übernehmen `<Strg-D>` (Delete) nimmt einen Tabulator wieder weg `<Strg-T>` (Tab) fügt einen Tabulator dazu
`showmatch(sm)`	`no`	Passende `({[` beim Eintippen von `]})` kurz anspringen
`shiftwidth(sw)`	`8`	Breite bei nachträglichem Ein/Ausrücken per `<< >>`
`tabstop(ts)`	`8`	Tabulatorstop alle 8 Zeichen

- Eher **unwichtige** Optionen:

Name	Std	Bedeutung
`errorbells(eb)`	(`yes`)	Bei Fehler piepsen
`dir[ectory]`	`/tmp`	Temporäre Kopie der editierten Datei steht dort
`hardtab(ht)`	`8`	Tabulatorstop alle 8 Zeichen
`exrc(ex)`	`no`	Initialisierungsdatei `.exrc` zusätzlich einlesen
`lisp`	`no`	LISP-Modus einschalten
`magic`	(`yes`)	Metazeichen beim Suchen ohne `\` angeben
`para[graphs]`	`IPLPPPQP..`	*Troff*-Absätze für `( )` festlegen
`readonly(ro)`	`no`	`!` zum Schreiben erforderlich
`remap`	(`yes`)	Makros in Makros werden rekursiv aufgerufen
`scroll`	(`yes`)	Anzahl Zeilen bei `Strg-U/D` festlegen
`sect[ions]`	`SHNHH HU..`	*Troff*-Abschnitte für `[[ ]]` festlegen
`shell(sh)`	`/bin/sh`	Bei Shell-Escape aufzurufende Shell (`$SHELL`)
`tag[s]`	`/usr/lib/tags`	Tag-Datei(en)
`term`	`$TERM`	Name des Terminals
`terse`	`no`	Kurze Hinweise statt langer Meldungen
`warn`	(`yes`)	falls seit letzter Änderung nicht geschrieben
`window(w)`	`???`	Anzahl Bildschirmzeilen
`taglength(tl)`	`0`	Gültig für die Tag-Suche sind erste *n* Zeichen eines Wortes (0 = gesamte Wortlänge)

3.7.7 Tastenbelegung und Abkürzungen

Sollen Tasten wie `ESC`, `<Strg-M>`, `<Strg-C>`, ... Teil einer Tastenbelegung oder eines Such- und Ersetzungs-Kommandos sein, so muss vor ihnen `<Strg-V>` [`v=verbose`] eingetippt werden. Dies gilt auch für `<Strg-V>` selbst. *In Initialisierungsdateien sind die maskierenden* `<Strg-V>` *zu verdoppeln*. Die Zeichenfolge s darf auch eine `ESC`-Sequenz und beliebig lang sein, keines ihrer *Präfixe* darf allerdings eine bereits für eine andere Abkürzung verwendete Zeichenfolge sein. Das Zeichen `"` mit `\` versehen (wird sonst als Kommentaranfang interpretiert) [`map=mapping`, `ab=abbreviate`].

- Tastenbelegung im Command-Modus (beginnen im Command-Modus, enden in einem beliebigen Modus)

`:map` s cmd	Zeichenfolge s mit Kommando cmd belegen
`:unmap` s	Zeichenfolge s wieder mit ursprünglicher Belegung versehen
`:map`	Alle Tastenbelegungen des Command-Modus auflisten

- Tastenbelegung im Edit- und Search-Modus

`:map!` s cmd	Zeichenfolge s mit Kommando cmd belegen
`:unmap!` s	Zeichenfolge s wieder mit ursprünglicher Belegung versehen
`:map!`	Alle Tastenbelegungen des Edit/Search-Modus auflisten

- Abkürzung

`:ab` in out	Bei Eingabe von in mit nachfolgendem Leerzeichen out einsetzen
`:unab` in	Abkürzung in wieder entfernen
`:ab`	Alle Abkürzungen auflisten

3.7.8 Kommando-Buchstaben

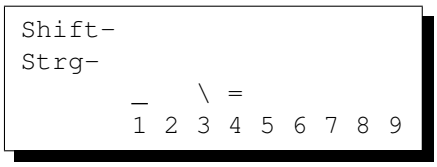
- **Belegte** Kommando-Buchstaben im *Vi*:

```
       a b c d e f   h i j k l m n o p   r s t u   w x y z
Shift- A B C D E F G H I J   L M N O P Q R S T U   W X Y Z
Strg-    B C D E F G H I J   L M N   P Q   S   U       Y Z
       0 + - . , ; : / ? ` ' " % & $ ~ ! ( ) [ ] < > { } @
       ESC  BS  TAB  SPACE  ␣
```

- **Nicht belegte** Kommando-Buchstabe im *Vi* (mit eigenen Kommandos belegbar):

```
          g     q       v
Shift-      K           V
Strg-  A    K O   R T   V W
       _ * \ =   #
       1 2 3 4 5 6 7 8 9
```

- **Nicht belegte** Kommando-Buchstabe im *Vim* (mit eigenen Kommandos belegbar):



- **Entbehrliche belegte** Kommando-Buchstaben (in eckigen Klammern steht der Ersatz dafür). Sie können daher ohne Funktionalitätsverlust mit eigenen Kommandos belegt werden:

`C`	Ab Cursor bis Zeilenende Text ersetzen [`c$`]
`D`	Ab Cursor bis Zeilenende löschen [`d$`]
`Q`	Permanent in den Ex-Modus umschalten [`:ex`] (mit `vi CR` verlassen)
`S`	Aktuelle Zeile ersetzen [`cc`]
`s`	Aktuelles Zeichen ersetzen [`c␣`, `cl`]
`X`	Zeichen *vor* Cursor löschen [`hx`]
`Y`	Aktuelle Zeile im temporären Puffer merken [`yy`]
`ZZ`	*Vi* mit Abspeichern verlassen [`:wq`, `:x`]
`+`	Eine Zeile nach oben gehen [`k`]
`-`	Eine Zeile nach unten gehen [`j`]
`SPACE`	Ein Zeichen nach rechts [`l`]
`BS`	Ein Zeichen nach links [`h`]
`Strg-N`	Eine Zeile nach unten [`j`]
`Strg-P`	Eine Zeile nach oben [`k`]
`Strg-E`	Bildschirm Zeile nach oben schieben (Cursor bleibt stehen)
`Strg-Y`	Bildschirm Zeile nach unten schieben (Cursor bleibt stehen)

3.7.9 Sonstige Kommandos

- Häufig benötigt [`x=crossout`, `d=delete`, `w=word`, `e=endword`, `p=paste/put`]:

`.`	Letztes Editier-Kommando wiederholen
`~`	Für aktuelles Zeichen die Groß/Kleinschreibung umkehren
`xp`	Aktuelles Zeichen und nächstes Zeichen vertauschen
`ddp`	Aktuelle Zeile mit nächster Zeile vertauschen
`dwwP`	Aktuelles Wort mit nächstem Wort vertauschen (Beg. auf 1.Z 1.W)
`deep`	Analog (Beginn auf Leerzeichen vor dem 1. Wort)
`X`	Zeichen vor dem Cursor löschen (entspricht `BS`)

- Aktuelle Zeile bearbeiten `c=change`, `d=delete`, `y=yank/copy`, `p=paste/put`:

`cc`	Aktuelle Zeile durch eingetippten Text ersetzten
`dd`	Aktuelle Zeile löschen und im temporären Puffer merken
`yy`	Aktuelle Zeile im temporären Puffer merken
`<<`	Aktuelle Zeile ausrücken
`>>`	Aktuelle Zeile einrücken
`!!`cmd	Aktuelle Zeile an Betriebssystem-Kommando cmd übergeben und durch Ergebnis ersetzen

- Zeilen aneinanderhängen [`J=join`]:

`J`	Nächste Zeile an aktuelle Zeile anhängen
n`J`	Die nächsten $n-1$ Zeilen an aktuelle Zeile anhängen

- Steuerung/Control-Kommandos [`l=load`, `g=get info`]:

`<Strg-L>`	Bildschirm neu aufbauen
`<Strg-G>`	Zeigt in unterster Zeile an: Aktuelle Datei, [Modified], [Read-Only], aktuelle Zeilennummer, Anzahl Zeilen, %-Abstand vom Dateianfang
`<Strg-6>`	Zwischen aktueller und letzter Datei hin- und herspringen (eigentlich `Strg-[` auf englischen Tastaturen) (sofern sie nicht editiert wurden, dann hilft nur noch `:e!#`)

- Betriebssystem-Kommandos auf Zeilenbereiche anwenden [`G=go`, `!=execute`]:

`!...`cmd	Textbereich ... an Betriebssystem-Kommando cmd übergeben und durch Ergebnis ersetzen
`:10,30!sort`	Zeilen 10-30 sortieren
`1G!Gsort`	Gesamte Datei sortieren

- Interessante *Ex*-Kommandos [`ta=tag`, `sh=shell`, `so=source`, `cd=change dir`]:

`:ta` $func$	Zu Funktion $func$ in ihrer Quelldatei springen (vorher das Kommando `ctags *.c *.h` aufrufen)
`:sh`	Betriebssystem-Shell starten, mit `exit/<Strg-D>` verlassen
`:so` $file$	*Vi*-Kommandos in Datei $file$ ausführen
`:cd` dir	dir zum aktuellen Verzeichnis machen
`<Strg-]>`	Zur Definitions-Stelle des Namens unter dem Cursor springen (inklusive Dateiwechsel!)

4 Sonstiges

4.1 Vorsicht

- Versehentliches Betätigen der **Umschaltungs-Feststellung** (`CAPS LOCK`) führt zu seltsamen Reaktionen des Editors, da die großen Kommando-Buchstaben eine andere Bedeutung haben als die kleinen (`j` wird dann z.B. als `J` interpretiert und verkettet Zeilen, anstatt den Cursor nach unten zu verschieben). Nach dem Aufheben der Umschaltungs-Feststellung kann (leider) nur die *allerletzte* der irrtümlichen Änderungen zurückgenommen werden (im *Vim* auch mehrere).

- Der *Vi* bietet **keine Sicherheit gegen mehrfaches gleichzeitiges Editieren** der gleichen Datei, der letzte Schreibvorgang zählt.

- Der *Vi* sichert beim Start die zu editierende Datei in einer **temporären Datei** in dem Verzeichnis, das die Option `dir=...` festlegt (Default: `/tmp`). In diesem Verzeichnis muss der Benutzer Schreibrecht haben, außerdem muss dort Platz in der Größe der zu editierenden Datei zur Verfügung stehen. Besteht nicht mehr genügend Platz für die temporäre Datei, so wird beim Start nur ein Teil der Datei in den Editor geladen und/oder er schaltet während dem Editieren in den **Open-Modus** um (kann mit `vi CR` verlassen werden).

- Da das Erstellen der Dateikopie Rechenzeit und Plattenplatz kostet, sollte für das **reine Anzeigen** einer Datei besser mit sogenannten „Viewern“ (`more`, `less`, `pg`, `...`) gearbeitet werden. Innerhalb dieser Kommandos kann man blättern, suchen und bei Bedarf

sogar den *Vi* aufrufen (`h=help` ausprobieren). Dies gilt vor allem für große Dateien (ab 1 MByte).

Alternativ den *Vi* per `view` im Read-Only Modus benutzen.

4.2 Defizite des *Vi*

Die Defizite des *Vi* sind gleichzeitig **Verbesserungen** im *Vim*, da dieser alle dort fehlenden Verhaltensweise und Kommandos (und noch viel mehr!) implementiert. In Klammern ist immer das *Vim*-Kommando angegeben:

- Undo/Redo um mehr als 1 Schritt (`u <Strg-R>`).
- Ex-, Such- und Ersetzungs-Kommandos editierbar und wiederholbar (per Cursor-Auf/Ab).
- Ex-, Such- und Ersetzungs-Kommandos beim Verlassen des *Vi* merken.
- Marken und Puffer pro Datei merken.
- Kommando zum Zurückspringen zur vorherigen Datei (`:prev=previous`).
- Anzeige der aktuellen Cursorposition (Zeile + Spalte) (`:set ruler`).
- Anzeige der begonnenen Kommandosequenz (`:set showcmd`).
- Anzeige des eingetippten Wiederholungsfaktors (`:set showcmd`).
- Textblöcke visuell markieren (Visual-Mode) (`v V <Strg-V>`).
- Gleichzeitig viele Dateien im Terminal anzeigen (Fenster) (`<Strg-W>`...).
- Folding (Verbergen) von Textblöcken (`z...`).
- Syntaxcoloring (`:syntax on/off`).
- Absätze reformatieren (`!} fmt -80 -u` unter dem *Vi*) (`gq<BEW>`).
- Hilfefunktion (`:help  :help CMD`).
- Mausbedienung (Gvim).
- Menüs (Gvim).
- Spaltenorientierte Textblöcke (`<Strg-V>`).
- Horizontaler Umbruch bei langen Zeilen (`:set wrap`).
- Programmierbar (Kontrollstrukturen und Funktionen).
- Textobjekte (Buchstaben, Worte, Sätze, Absätze, ...).

4.3 *Vi*-Clones

Es gibt eine ganze Reihe von *Vi*-Clones wie z.B. *Nvi, Vim, Vis, Elvis, Stevie, Vile*, die frei verfügbar sind und gegenüber dem *Vi* weniger Fehler und viele Erweiterungen bieten. Der *Vim* (*Vi Improved*) ist der bei weitem mächtigste und auch eleganteste.

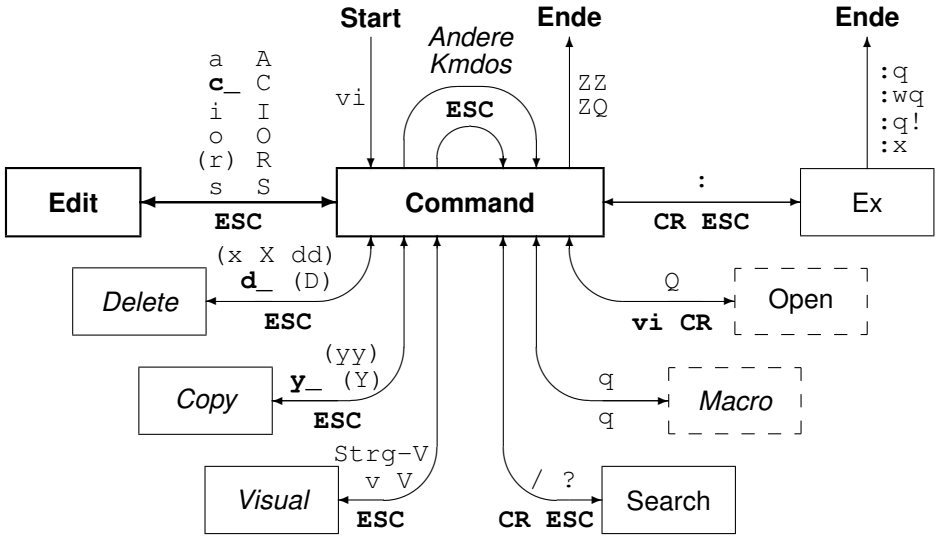
5 Verbesserungen im *Vim*

5.1 Arbeitsmodi des *Vim*

Die Arbeitsmodi des *Vim* sind gegenüber denen des *Vi* (siehe Abschnitt 1.5.2 auf Seite 7) um folgende erweitert worden:

Modus (deu)	Modus (eng)	Bedeutung
Löschen	Delete	Löschen von Text
Kopieren	Copy	Kopieren von Text
Visuell	Visual	Visuelle Markierung
Makro	Macro	Makro-Aufzeichnung
Wartend	Operator Pending	Unvollständiger Befehl

Die folgende Grafik beschreibt die **Übergänge** zwischen allen *Vim*-Arbeitsmodi. Durch Eingabe der bei den Übergangspfeilen stehenden Kommandozeichen wechselt man zwischen diesen Modi hin- und her. **Bei allen anderen Kommandos bleibt der *Vim* im Command-Modus.**



- Der **Delete-Modus** umfasst das Löschen von Text.
  - ▷ **Kommando** `d_` [`Delete`] benötigt die Angabe eines **Bewegungs-Kommandos** zur Festlegung des **Textbereichs**, auf den es sich beziehen soll; daher der Unterstrich (siehe Abschnitt 3.4 auf Seite 20).

  - ▷ **Kommandos** `x` und `X` [`Crossout`] sind geklammert, da sie nicht mit `ESC` zu beenden sind, sondern **automatisch** wieder in den Command-Modus zurückkehren (Grund für diese Abweichung von der Regel ist, dass diese Kommandos sehr häufig benötigt werden).

- Der **Copy-Modus** umfasst das Kopieren von Text.

  - ▷ **Kommando** `y_` [`Yank/CopY`] benötigt die Angabe eines **Bewegungs-Kommandos** zur Festlegung des **Textbereichs**, auf den es sich beziehen soll; daher der Unterstrich (siehe Abschnitt 3.4 auf Seite 20).

- Der **Visual-Modus** erlaubt das Markieren von Text per **Bewegungs-Kommandos**, anschließend wird der Text mit einem der Kommandos `c d y p > < ~ u U !` bearbeitet. Diese Arbeitsweise ist analog zu der in *Word* oder anderen grafischen Editoren.

  Dies steht im Gegensatz zur im *Vi* normalerweise üblichen Arbeitsweise, die zuerst die Art der Bearbeitung auswählt (Einfügen, Überschreiben, Ersetzen, Löschen) und anschließend den zu bearbeitenden Textteil über Bewegungsbefehle selektiert.

- Der **Macro-Modus** bietet das Aufzeichnen von Kommandofolgen in **Registern** R=`"a-"z`, die anschließend per `@`R beliebig oft wieder "abgespult" werden können. Er ist gestrichelt dargestellt, da er selten benötigt wird, allerdings leicht versehentlich mit dem Kommando `q` eingeschalten werden kann. Man sollte daher zumindest wissen, dass er mit dem Kommando `q` wieder auszuschalten ist.

- Im **Pending-Modus** ist ein Kommando noch unvollständig und der *Vim* wartet auf weitere Eingaben. Mit Hilfe der Option `showcmd` werden die bisherigen Tastendrücke des unvollständigen Kommandos in der Statuszeile angezeigt.

- Folgende Modi sind zusätzlich in der untersten Bildschirmzeile zu erkennen:

Anzeige	Bedeutung
`recording`	Macro-Modus (nach `q` bis zur Eingabe von `q`)
`VISUAL`	Visual-Modus (nach `v V Strg-V` bis zur Eingabe von `c d y p > < ~ u U !` oder `ESC`)

5.2 Initialisierung des *Vim*

Beim *Vim* gibt es zusätzlich die **Umgebungs-Variable** `VIMINIT` und die **Konfigurations-Dateien** `/etc/vimrc` und `.vimrc`. Sie werden in folgender Reihenfolge gelesen (die 1. Datei auf jeden Fall, von den anderen 4 Möglichkeiten wird nur die erste gefundene verwendet):

```
/etc/vimrc      # immer eingelesen
$VIMINIT        # 1. gefundene eingelesen
.vimrc          # 1. gefundene eingelesen
$EXINIT         # 1. gefundene eingelesen
.exrc           # 1. gefundene eingelesen
```

Zur Initialisierung des *Vi* siehe Abschnitt 2 auf Seite 13):

5.3 Spezielle Optionen des *Vim*

Der *Vim* kennt etwa 20-30 Mal mehr Optionen als der *Vi*, die wichtigsten neuen Optionen sind im folgenden aufgelistet (die Optionen des *Vi* sind in Abschnitt 3.7.6 auf Seite 31 zu finden):

`compatible`	*Vi*-kompatibel (immer ausschalten!)
`backup`	Backupdatei erzeugen (zusätzliche Endung `~`)
`fileformat (ff)`	Format von Textdateien (`dos`, `unix` oder `mac`) (bestimmt Zeilentrenner `CR+NL`, `NL` oder `CR`)
`hlsearch`	Suchergebnisse markieren (`highlight`)
`incsearch`	Sofort während Eingabe suchen (`incremental`)
`ruler`	Cursor-Zeile/Spalte in Statuszeile anzeigen (Lineal)
`showcmd`	Unvollständiges Kommando in Statuszeile anzeigen (`command`)
`title`	Dateiname im Fenstertitel anzeigen
`smartcase`	Beim Suchen Gross/Kleinschreibung „intelligent“ ignorieren (alles klein → ignorieren, ein Zeichen groß oder `\C` → nicht ignorieren)
`syntax on/off`	Syntaxcoloring ein/ausschalten (abhängig von Extension)
`wrap`	Zu lange Zeilen umbrechen
`sidescroll`	Rand innerhalb dem nach links/rechts gescrollt wird
`sidescrolloff`	Restspaltenbreite, ab der nach links/rechts gescrollt wird

Achtung: Bei `syntax on` das `set` davor weglassen!

5.4 Visual-Mode im *Vim*

Im *Vim* ist Text **visuell** markierbar (wird **invertiert** dargestellt), anschließend kann dieser markierte Text mit den Kommandos `c d y p > < ~ u U !` bearbeitet werden (`c=change`, `d=delete`, `y=yank/copy`, `p=put/paste`, `<=shift left`, `>=shift right`, `~=swap case`, `u=low case`, `U=upper case`, `!=execute`):

`v`	Textblock *zeichenweise* markieren
`V`	Textblock *zeilenweise* markieren
`<Strg-V>`	Textblock *spaltenweise* markieren

Größe + Form des markierten Bereichs wird durch **Bewegungskommandos** gesteuert.

5.5 Spezielle Bewegungen des *Vim*

Häufig muss man zwischen mehreren bestimmten Textstellen hin- und herspringen. Dafür bietet der *Vim* eine schöne Lösung [`o=old`, `TAB=to`]:

`<Strg-O>`	Zur vorherigen Cursorposition zurückspringen (mehrfach)
`TAB`	Zur neueren Cursorposition springen (nach `Strg-O`, mehrfach)

5.6 Spezielle Editier-Operationen des *Vim*

Häufig editiert man Texte, in denen bestimmte Worte mehrfach vorkommen und evtl. aufgrund der Länge oder Buchstabenkombination schwer fehlerfrei einzutippen sind (z.B. Variablen- oder Funktionsnamen in Programmen!). Dafür bietet der *Vim* eine schöne Lösung [p=previous, n=next]:

<Strg-P>	Wort gemäß *vorherigem* passenden vervollständigen (mehrfach)
<Strg-N>	Wort gemäß *nächstem* passenden vervollständigen (mehrfach)

Recht einfach ist auch das Kopieren von Text aus der Zeile darüber oder darunter in die aktuelle Zeile:

<Strg-Y>	Zeichen *über* Cursor kopieren
<Strg-E>	Zeichen *unter* Cursor kopieren

Ein eher exotisches Feature des *Vim* ist die Möglichkeit, eine Zahl unter dem Cursor um 1 zu erhöhen bzw. zu erniedrigen (auch oktale oder hexadezimale Zahlen):

<Strg-A>	Zahl unter Cursor *inkrementieren* (mehrfach)
<Strg-X>	Zahl unter Cursor *dekrementieren* (mehrfach)

5.7 Tastennamen im *Vim*

Im *Vim* können Sondertasten in Such-Befehlen, Mappings und Abkürzungen in **Klartextform** angegeben werden, indem Sie mit <...> umrahmt werden:

<F1>...<F12>	Funktionstasten
<SPACE>	Leertaste
<ESC>	ESC-Taste
<TAB>	Tabulator-Taste
<BS>	Backspace-Taste
<CR>	Return-Taste
<NL>	Newline
<FF>	Formfeed
<BSLASH>	Backslash
<LT>	<-Zeichen
<DEL>	Delete-Taste
<INS>	Insert-Taste

<LEFT>	Cursor nach links-Taste
<RIGHT>	Cursor nach rechts-Taste
<UP>	Cursor nach oben-Taste
<DOWN>	Cursor nach unten-Taste
<HOME>	Home-Taste
<END>	End-Taste
<PAGEUP>	Seite auf-Taste
<PAGEDOWN>	Seite ab-Taste
<S-XXX>	Shift + Taste XXX
<A-XXX>	Alt + Taste XXX
<C-XXX>	Steuerung + Taste XXX

5.8 Im *Vim* zusätzlich belegte Kommando-Buchstaben

Bereits der *Vi* belegt sehr viele Tasten im Command-Modus mit Befehlen, meist sind die Tasten sogar **dreifach belegt** (ohne Shift, mit Shift, mit Strg). Aufgrund der Vielzahl an hinzugefügten Funktionen belegt der *Vim* die meisten der noch freien Tastenkombinationen:

```
          g          q v
Shift-      K          V
Strg-  A    K O R T    V W X
```

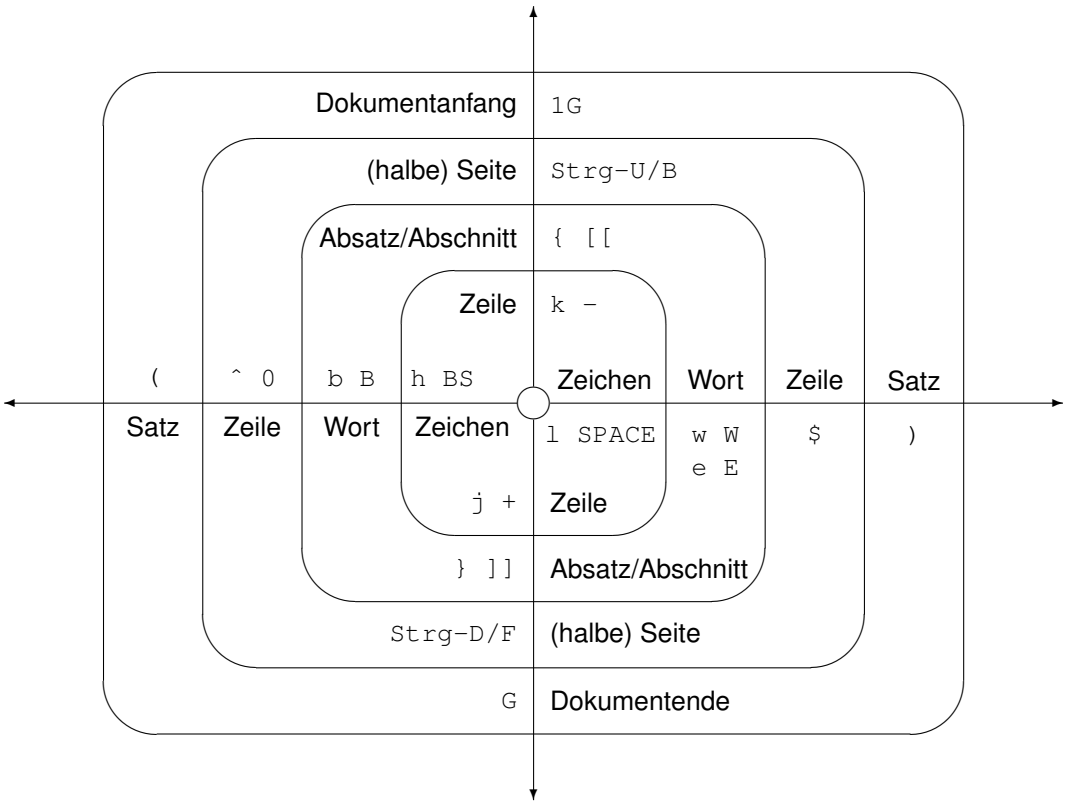
Hervorzuheben sind die Kommandos `g=global`, `Strg-W=window` und `z=fold`, unter denen eine **Vielzahl von Funktionalitäten** zusammengefasst ist. Daher ist bei diesen beiden Kommandos immer ein 2. Buchstabe notwendig, um die eigentliche Funktionalität auszuwählen.

6 Übersichten

6.1 Visualisierung der Bewegungs-Kommandos

Die aktuelle Cursorposition ist jeweils durch `X` gekennzeichnet, Zeilenanfang und -ende sind durch `|` markiert, Dateianfang und -ende sind durch `---` markiert

- Cursorbewegungen



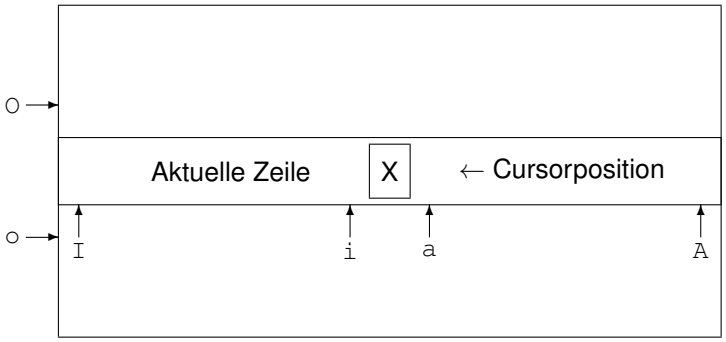
- Horizontales Bewegen des Cursors

```
///|      2Fo Fa Fo              fo  2fo 3fo fa          |///
///|      B|  |  | b   eE   2E    |3E |   |  |          |///
///|      ||  |  | |   ||   |     ||  |   |  |          |///
///|      Kommandos, Xie, den Cursor horizontal bewegen.|///
///||     | |  |  | | | | |   |  |   |   |  |  |   |      ||///
///|0     ^ |  |  | h l | |   |  |   |   |  |  |   |      $|///
///|        |  |  |     w 2w 3w  |  4w   |  |  5W       |///
///|        |  |  |       W  2W  |  3W   |  |  5w       |///
///|       2To Ta To            to  2to 3to ta          |///
```

- Vertikales Bewegen des Cursors

```
                k
          ////////|/////////
          ///+----|-----+///
     1G0 -->|Komm|ndos,|<-- 1G$
       - -->|die vlle  |///
          ///|den Xursor|///
       + -->|vertikal  |///
      G0 -->|bewe^en.<----- G$
          ///+----|-----+///
          ////////|/////////
                  j
```

- Einfügen von Text



- Löschen von Text (analog ändern mit `c` und Kopieren mit `y` statt `d`).

```
///|<-------------------dd------------------->|///
///|<------------d0----------|-------D------->|///
///|    <----------d^--------|-------d$------>|///
///|    |      <------dB-----|----dW---->     |///
///|    |      |             |----dE--->|     |///
///|    |      |        <-2db|--2de->  ||     |///
///|    |      |        |<-db|--dw->|  ||     |///
///|    Einige Kommandos,zum Xöschen,von Text.|///
///|                         ||               |///
///|                         Xx               |///
```

Einzelzeichen löschen mit `X=crossout` (vor Cursor) und `x=crossout` (auf Cursor).

6.2 Kommandoübersicht A

Kmdo	Typ	Bedeutung
a A	**e**	**Append** after cursor [line]
b B	*m*	**Back** one `word` [`WORD`]
c C	**e s**	**Change** ... [to end of line] (`cc` = current line)
d D	**s**	**Delete** ... [to end of line] (`dd` = current line)
e E	*m*	**End** of `word` [`WORD`]
fx Fx	*m*	**Find** next [prev] char x in line
G nG	*m*	**Goto** last [nth] line
h l	*m*	Cursor left [right]
H ML	*m*	To **home** [**middle**/**last**] line on screen
i I	**e**	**Insert** before cursor [line]
j k	*m*	Cursor down [up]
J		**Join** line with next
mx ’x	*m*	**Mark** [return to] position x (start of line)
`x	*m*	Return to position x (*exact* character position)
n N	*m*	To **next** [prev] search occurence
o O	**e**	**Open** a line below [above]
p P		**Put** in after [before]
Q		**Quit** go to `ex` mode (return with `vi CR`)
r R	(**e**)	**Replace** 1 [all] char[s] (no `ESC` with `r`)
s S	**e**	**Substitute** char [line]
tx Tx	*m*	**To** next [prev] char x in line
u U		**Undo** last [all] change[s] in file [line]
w W	*m*	One **Word** [`WORD`] forward
x X		**Crossout** char at [before] cursor
y Y	**s**	**Yank**/**Copy** ... [line] (`yy` = current line)
zx	*m*	**Zone** at top/middle/bottom line x=`CR`/`.`/`-`
ZZ		Write changes to file, exit editor ($\approx$ Omega)
^b ^f	*m*	**Backward** [**forward**] one page
^d ^u	*m*	**Downward** [**upward**] half a page
^d ^t		**Delete** [**tab**] one `shiftwidth` during insert
^e ^y		**Expose** 1 more line at bottom [top]
^g		**Get** file name and statistics
~		Change case of char
+ −	*m*	To first char in next [prev] line
0 ^ $	*m*	To first [first true/last] char in line
n ; ,	*m*	Repeat [reverse] last `f`, `F`, `t`, or `T`
.		Repeat last change of the text (dot)
< >	**s**	Shift left [right] ... one `shiftwidth`
<< >>		Shift current line one `shiftwidth` left [right]
( )	*m*	To beginning of prev [next] sentence
{ }	*m*	To beginning of prev [next] paragraph
[[ ]]	*m*	To beginning of prev [next] section/function
/ ?	*m*	Search forward [backward]
`` ’’	*m*	Return to prev position [line]
: :!		Execute `ex` [`sh`] command
! !!	**s**	Shell command on ... [this] line

- **e**[`edit`] kennzeichnet Kommandos, die in den **Edit-Modus** umschalten (mit `ESC` wieder zu beenden).
- **s**[`select`] heißt, der betroffene Bereich wird über ein **Bewegungs-Kommando** ausgewählt.
- *m*[`move`] kennzeichnet ein **Bewegungs-Kommando**.
- `ESC` eingeben, um aus dem EdiEdit-Modus wieder in den Command-Modus zu **wechseln**.
- Nach `c d y < > !` wählt ein beliebiges Bewegungs-Kommando den **Bereich aus**.
- Verdoppelung von `c d y < > !` wirkt auf die **aktuelle Zeile**.
- Vor einem Kommando n eingeben, um es n-mal zu **wiederholen**.
- Vor einem Bewegungs-Kommando m eingeben, um es m-mal zu **wiederholen**.
- Ein `Word` ist eine Folge von Zeichen **ohne Leerzeichen + Satzzeichen** darin.
- Ein `WORD` ist eine Folge von beliebigen Zeichen **ohne Leerzeichen** darin.
- `shiftwidth` ist eine *Vi*-Option und legt die **Einrückungsbreite** fest.

6.3 Kommandoübersicht B

Diese Tabelle nutzt die Tatsache aus, dass die Editier-Kommandos `d c y < > !` des *Vi* **orthogonal** mit den Bewegungs-Kommandos **kombinierbar** sind. D.h. `h` bewegt den Cursor nach links, `dh` löscht nach links. $/abc$ springt zum ersten Auftreten der Zeichenkette abc, `d`$/abc$ löscht bis zum ersten Auftreten der Zeichenkette abc usw.

Die folgenden *Vi*-Kommandos bilden die Form `<Aktion> [Objekt]` (mögliche Kombinationen sind durch $\star$ gekennzeichnet):

Objekt	**Aktion** d	c	y	<	>	!
h l	*	*	*			
j k	*	*	*	*	*	*
0 ^ $	*	*	*			
w W	*	*	*			
b B	*	*	*			
e E	*	*	*			
f F	*	*	*			
t T	*	*	*			
n|	*	*	*			
H M L	*	*	*	*	*	*
/*pattern*	*	*	*	*	*	*
?*pattern*	*	*	*	*	*	*
nG	*	*	*	*	*	*
%	*	*	*	*	*	*
( )	*	*	*	*	*	*
{ }	*	*	*	*	*	*
[[ ]]	*	*	*	*	*	*
Objekt	d	c	y	<	>	!
	Aktion					

Aktion		
d = delete	y = yank/copy	> = shift right
c = change	! = filter	< = shift left
		s = substitute
		r = replace
		x = cross out

Objekt		
h = left char	l = right char	k = up line
+ = next line	- = prev line	j = down line
0 = line start	$ = line end	^ = first non-white space
w = forw word	b = back word	e = end of word
W = forw WORD	B = back WORD	E = end of WORD
/ = srch forw	? = srch back	nG = goto line n
H = home scrn	M = mid scrn	L = last scrn line
(= sent start	) = sent end	fc = forw to c (incl)
{ = para start	} = para end	Fc = back to c (incl)
[[= sect start	]] = sect end	tc = forw to c (non-incl)
% = match {[()]}		Tc = back to c (non-incl)
n| = to col n		nG = to line n