

1 Grundlegende Kommandos

Befehl/Tastenkombinat.	Beschreibung	Beispiel
:w <Dateipfad>	Aktuellen Buffer in Datei speichern	:w test.txt
:e <Dateipfad>	Datei laden	:e text.txt
:q	Aktuelles vim-Fenster schließen	:q
:<Zeilennummer>	Gehe zu Zeile N	:5
:source <Dateipfad>	Einstellungen aus dieser Datei laden	:source ~/.vimrc
:set ft=<t>	Syntax-Hervorhebung für Sprache <t> aktivier.	:set ft=c
i	In den INSERT-Modus wechseln	–
v	In den VISUAL-Modus wechseln	–
V	In den VISUAL LINE-Modus wechseln	–
ESC	Aktuellen Modus verlassen	–

2 Cursor bewegen

B/TK	Beschreibung
h	Cursor nach ◀ links bewegen
j	Cursor nach ▼ unten bewegen
k	Cursor nach ▲ oben bewegen
l	Cursor nach ▶ rechts bewegen
w	Cursor Wort für Wort vorwärts bewegen (von Wortteil zu Wortteil) (Auto-mobil → Auto- mobil → Auto- mobil → Auto-mobil)
W	Cursor Wort für Wort vorwärts bewegen (von Space zu Space) (Auto-mobil → Auto-mobil)
b	Cursor Wort für Wort zurück bewegen (von Wortteil zu Wortteil) (Auto-mobil → Auto- mobil → Auto- mobil → Auto-mobil)
B	Cursor Wort für Wort zurück bewegen (von Space zu Space) (Auto-mobil → Auto-mobil)
\$	Cursor Zeile <i>Ende</i>
^	Cursor Zeile <i>Anfang</i> (erstes non-space-Zeichen)
0	Cursor Zeile <i>Anfang</i> (erstes Zeichen)
gg	Cursor Datei <i>Anfang</i>
G	Cursor Datei <i>Ende</i>
{	Cursor akt. Absatz vor
}	Cursor akt. Absatz hinter
%	Cursor von Klammer o.ä. zum entsprechenden Gegenstück bewegen
fX	Cursor nächstes Vorkommen von X (vorwärts) (X=bel. Zeichen) (exakt)
tX	Cursor nächstes Vorkommen von X (vorwärts) (X=bel. Zeichen) (1 Stelle davor)
FX	Cursor letztes Vorkommen von X (rückwärts) (X=bel. Zeichen) (exakt)
TX	Cursor letztes Vorkommen von X (rückwärts) (X=bel. Zeichen) (1 Stelle dahinter)

Befehl/TK	Beschreibung
Ctrl-d	Cursor ½ Bildschirm nach unten (down)
Ctrl-f	Cursor 1 Bildschirm nach unten (forwards)
Ctrl-u	Cursor ½ Bildschirm nach oben (up)
Ctrl-b	Cursor 1 Bildschirm nach oben (backwards)

Befehl/TK	Beschreibung
H	Cursor in erste angezeigte Zeile (relativ zum Fenster) (home) [3H = 3. Zeile von oben]
M	Cursor in mittlere Zeile (relativ zum Fenster) (middle)
L	Cursor in letzte angezeigte Zeile (relativ zum Fenster) (last line) [3L = 3. Zeile von unten]
zt	Aktuelle Zeile als oberste Zeile des Fensters setzen (top)
zz	Aktuelle Zeile in die Mitte des Fensters setzen
zb	aktuelle Zeile als unterste Zeile des Fensters setzen (bottom)
m<Register>	Cursorposition in das Register <Register> speichern
'<Register>	Cursor an im Register <Register> gespeicherte Position bewegen

3 Nützliches für die .vimrc

(i)

In die Datei .vimrc werden normale vim-Kommandos eingetragen, die beim Programmstart automatisch ausgeführt werden sollen. Pro Zeile ein Kommando. Kein ; o.ä. am Zeilenende. Kommentarzeilen beginnen mit ".

Befehl/Zeile	Beschreibung
syntax on	Syntaxhervorhebungen einschalten
set number	Zeilennummern anzeigen
set incsearch	Inkrementelle Suche aktivieren (Suchen, während der Suchbegriff getippt wird)
set ignorecase	Case-sensitivity der Suche deaktivieren
set hlsearch	„highlight search“ aktivieren: Alle Suchtreffer werden hervorgehoben
set wrap/nowrap	Zeilenumbruch am Fensterende aktivieren/deaktivieren
set tabstop= <i>n</i>	Ein Tab ist so breit wie <i>n</i> Spaces
set smartindent	Einrückungen in Programmcode automatisch vornehmen

4 Grundlegende Textbearbeitung

(i)

Fast alle vim-Kommandos verstehen **Zähler**, z.B.

- 2fX lässt den Cursor an das *zweitnächste* Vorkommen von X springen (s.o.)
- 5dd entfernt *fünf* Zeilen
- 3cw ersetzt die *drei* nächsten (also rechts des Cursors liegenden) Worte

Befehl/TK	Beschreibung
u	Letzte Aktion rückgängig (<i>undo</i>)
Ctrl-r/:red	Letzte Aktion wiederherstellen (<i>redo</i>)
x	Zeichen unter dem Cursor löschen
X	Zeichen vor (links vom) Cursor löschen
o	Neue Zeile <i>nach</i> der aktuellen Zeile und Wechsel in den INSERT-Modus
O	Neue Zeile <i>vor</i> der aktuellen Zeile und Wechsel in den INSERT-Modus

(i)

Bei den folgenden Kommandos kann <?> durch mehrere Kommandos ersetzt werden:

- Um die Befehle auf **Zeilen** anzuwenden, wird der **Buchstabe des Befehls wiederholt**.
Beispiel: dd löscht eine Zeile, cc ersetzt eine Zeile, ...
- **i wie „inside“:** Bezieht sich auf den **Text zwischen etwas**, z.B. Text in **Klammern** oder in **Anführungszeichen**. Die Delimiter bleiben unberührt.
Beispiel: Befindet sich der Cursor zwischen 2 Anführungszeichen (in einem String o.ä.) kann ci" ausgeführt werden, um den gesamten Text innerhalb der Anführungszeichen zu ersetzen. Funktioniert auch mit Klammern (ci ().
- **a wie „around“:** Wie i, schließt aber auch die Delimiter selbst (Klammern, Anführungszeichen, ...) mit ein.
- **Bewegungskommandos aus Kapitel 2, z.B.:**
 - {h, l} (einzelne Zeichen links bzw. unter dem Cursor), {j, k} (aktuelle Zeile und die darüber/darunter)
 - Bewegungskommandos zum Zeilenanfang und -ende, Dateianfang und -ende, Abschnittsanfang und -ende, ...
 - {w, W} (Worte vorwärts), {b, B} (Worte rückwärts).
Beispiel: cW ersetzt das nächste Wort.
- **Suchkommandos aus Kapitel 2 und Kapitel 6:**
{tX, TX, fX, FX}; Suche mit /<S> und ?<S> (s.u.)
Beispiel: d2fi entfernt den Text vom Cursor bis zum übernächsten „i“.

Befehl/TK	Beschreibung
d<?>	<i>Etwas</i> (<?>) löschen
c<?>	<i>Etwas</i> (<?>) löschen und in den INSERT-Modus wechseln

5 Kopieren, ausschneiden, einfügen

Befehl/TK	Beschreibung
d<?>, c<?>, x	<i>Etwas</i> (<?>) ausschneiden (Löschen-Kommandos (s. o.) schneiden auch aus)
y<?>	<i>Etwas</i> (<?>) kopieren (yank)
p	Einfügen hinter den Cursor (rechts davon) (paste)
P	Einfügen vor den Cursor (links davon) (paste)

6 Suche

Befehl/TK	Beschreibung
/<Suchbegriff>	Das Dokument vorwärts nach dem Suchbegriff durchsuchen (cs-sen, regex)
?<Suchbegriff>	Das Dokument rückwärts nach dem Suchbegriff durchsuchen (cs-sen, regex)
n	Zum nächsten Treffer in Suchrichtung springen (bei /<S>: nach unten , bei ?<S>: nach oben)
N	Zum vorherigen Treffer in Suchrichtung springen (bei /<S>: nach oben , bei ?<S>: nach unten)
:noh	Aktuelle Hervorhebung von Suchtreffern entfernen

7 Ersetzen

Befehl/TK	Beschreibung
:s/<S>/<E>	Nächstes Vorkommen von <S> in der aktuellen Zeile durch <E> ersetzen (<i>case-sensitive, RegEx erlaubt</i>)
:%s/<S>/<E>	Erstes Vorkommen von <S> in der jeder Zeile des Dokuments (%) durch <E> ersetzen (<i>case-sensitive, RegEx erlaubt</i>)
:%s/<S>/<E>/g	Alle (g) Vorkommen von <S> im gesamten Dokument (%) durch <E> ersetzen (<i>case-sensitive, RegEx erlaubt</i>)
:%s/<S>/<E>/gc	Alle (g) Vorkommen von <S> im gesamten Dokument (%) durch <E> ersetzen ; vor Ersetzen nachfragen (c) (<i>case-sensitive, RegEx erlaubt</i>)
:'<, '>s/<S>/<E>/g	Alle (g) Vorkommen von <S> in der Auswahl ('<, '>) durch <E> ersetzen (<i>case-sensitive, RegEx erlaubt</i>) (<i>'<, '> wird durch Tippen von : im visuellen Modus automatisch eingefügt</i>)
:s//<E>	Nächstes Vorkommen der letzten Suche (mit / oder ?, s. Kapitel 6) in der aktuellen Zeile durch <E> ersetzen (<i>case-sensitive, RegEx erlaubt</i>)

Der Aufbau des :s-Kommandos lässt sich auch **allgemein** ausdrücken:

:<Bereich>s/<Suchbegriff>/<Ersatz>/<Flags>

(i)

- Wichtige **Bereiche**:
 - (nichts): Die **aktuelle Zeile**, in der sich der Cursor befindet
 - %: Das **ganze Dokument**
 - '<, '>: Die **Auswahl** des visuellen Modus (automatisch eingefügt)
- Wichtige **Flags**:
 - g wie **global**: Der Befehl wirkt sich nicht nur auf den 1. Treffer pro Zeile, sondern auf alle Treffer pro Zeile aus
 - c wie **confirm**: Vor dem Ersetzen nachfragen
 - i wie **insensitive**: Kommando case-insensitive durchführen
- Mehr Informationen zeigt das Kommando :help :s an.

8 Makros und Register

vim unterstützt Makros. Diese Funktion möchte ich an einem Beispiel deutlich machen:

Gegeben ist eine Liste der Länder der Welt, jedoch eingefasst in HTML-<option>-Tags. Diese Liste soll so bereinigt werden, dass am Ende nur noch die Ländernamen stehen bleiben.

Gegebene Liste	Gewünschtes Ergebnis
1 <option value="AF">Afghanistan</option>	1 Afghanistan
2 <option value="AL">Albania</option>	2 Albania
3 <option value="DZ">Algeria</option>	3 Algeria
4 <option value="AS">American Samoa</option>	4 American Samoa
5 <option value="AD">Andorra</option>	5 Andorra
6 <option value="AG">Angola</option>	6 Angola
...	...

Mithilfe der Makrofunktion von vim ist dies ganz einfach möglich:

1. Setzen Sie den Cursor an den Anfang des Dokuments (gg).
2. Drücken Sie q, um den Makro-Recorder zu aktivieren. Tippen Sie anschließend einen Buchstaben Ihrer Wahl, z.B. a. Der Buchstabe repräsentiert ein Register, eine Art Slot oder Speicherplatz für Ihr Makro, den Sie frei wählen dürfen.

Am unteren Bildschirmrand steht nun „recording“, der Makro-Recorder ist aktiv.

3. Tippen Sie nun die Befehle, die anschließend wiederholt werden sollen, für die 1. Zeile ein. Im Beispiel könnten das folgende Befehle sein:
 - a. da< (öffnenden <option>-Tag entfernen)
 - b. f< (zum Anfang des schließenden </option>-Tags springen)
 - c. d\$ (alles vom Cursor bis zum Zeilenende löschen)
 - d. j0 (Cursor in die nächste Zeile und an den Zeilenanfang bewegen)

Beachten Sie, dass am Ende der Befehlsfolge wieder nahtlos mit dem 1. Befehl angefangen werden kann, damit das Makro ohne Benutzerinteraktion durchlaufen kann!
4. Beenden Sie die Aufzeichnung mit einem Druck auf q. *Sie können sich den Inhalt der Register über den Befehl :reg anzeigen lassen und so das aufgezeichnete Makro überprüfen.*
5. Führen Sie das Makro aus: @<Registernamen>, also z.B. @a. Natürlich können Sie das Makro auch gleich mehrmals hintereinander ausführen: 50@a.

(i)**Zusammenfassung:**

q → Register aussuchen (z.B. a) → Befehle → q → @<Register> (z.B. @a)

9 Nutzung der Kommandozeile in vim

Befehl/TK	Beschreibung
:!<Befehl>	Befehl <Befehl> ausführen
:r !<Befehl>	Befehl <Befehl> ausführen und Ausgabe an Cursorposition einfügen
: '<, '>!<Befehl>	In Zusammenspiel mit dem visuellen Modus: Markierter Text wird als Eingabe für den Befehl <Befehl> verwendet; die Ausgabe des Befehls überschreibt den markierten Text in vim. ('<, '> wird automatisch eingefügt)

10 Puffer (Buffers)

Alle in vim geöffneten Dateien sind in einen sog. Puffer, also einen Speicherbereich, geladen. Es gibt aber auch Puffer, die nicht mit einer Datei verbunden sind.

Befehl/TK	Beschreibung
:ls	Geöffnete Puffer auflisten
:b<num>	Gehe zum Puffer mit der Nummer <num>
:bf	Gehe zum ersten Puffer (in der Liste)
:bl	Gehe zum letzten Puffer (in der Liste)
:bn	Gehe zum nächsten Puffer (in der Liste)
:bp	Gehe zum vorherigen Puffer (in der Liste)
:b#	Gehe zum vorher angesehenen Puffer (in der Historie)
:bd	Lösche den aktuellen Puffer
:bd<num>	Lösche den Puffer mit der Nummer <num>

11 Fenster und Tabs

Befehl/TK	Beschreibung
:sp	vim horizontal splitten (neuer Puffer)
:vs	vim vertikal splitten (neuer Puffer)
:sp <Dateipfad>	vim horizontal splitten (Datei <Dateipfad> wird geladen)
:vs <Dateipfad>	vim vertikal splitten (Datei <Dateipfad> wird geladen)

Befehl/TK	Beschreibung
Ctrl-w {hjkl}	Zwischen vim-Fenstern navigieren
Ctrl-w {HJKL}	vim-Fenster <i>bewegen</i>
Ctrl-w {+-}	Vertikale Größe des aktiven vim-Fensters <i>anpassen</i> (Tipp: Ctrl-w 10+)
Ctrl-w {<>}	Horizontale Größe des aktiven vim-Fensters <i>anpassen</i> (Tipp: Ctrl-w 10<)
Ctrl-w =	Offene Fenster gleichmäßig verteilen
:sb<num>	vim-Fenster horizontal <i>splitten</i> ; (Puffer <num> öffnen)
:vert sb<num>	vim-Fenster vertikal <i>splitten</i> ; (Puffer <num> öffnen)
:tabe	Neuer Tab mit neuem Puffer
:tabe <Dateipfad>	Datei <Dateipfad> in neuem Tab öffnen
gt	Zu nächstem Tab wechseln
gT	Zu vorherigem Tab wechseln

12 Einrückungen

Befehl/TK	Beschreibung
>>	Aktuelle Zeile eine Ebene weiter <i>einrücken</i> (Tipp: 4>>) (Im VISUAL-Modus: >)
<<	Aktuelle Zeile eine Ebene weniger <i>einrücken</i> (Tipp: 4<<) (Im VISUAL-Modus: <)
Ctrl-t	Aktuelle Zeile im INSERT-Modus eine Ebene weiter <i>einrücken</i>
Ctrl-d	Aktuelle Zeile im INSERT-Modus eine Ebene weniger <i>einrücken</i>
:set list/nolist	Macht unsichtbare Zeichen sichtbar/wieder unsichtbar (Man sieht, ob Tabs oder Spaces zum Einrücken verwendet werden)
:set expandtabs /noexpandtabs	Leerzeichen statt Tabs für Einrückungen verwenden/oder nicht
:set shiftwidth=n	Anzahl der Leerzeichen pro Einrückungsebene (bei Verwendung von >>/<<)
:set softtabstop=n	Anzahl der Leerzeichen pro Einrückungsebene (bei Verwendung der Tabtaste im INSERT-Modus)
=<Bereich>	z.B. =10j: Rückt die nächsten 10 Zeilen automatisch richtig ein

13 Folding

Befehl/TK	Beschreibung
zf<Bereich>	<Bereich> einfalten . z.B.: <ul style="list-style-type: none"> zf5j: Aktuelle und 5 weitere Zeilen einfalten. zf%: Bereich von Klammer (Cursorpos.) bis Gegenstück einfalten
zo	Fold an Cursorposition öffnen (<i>open</i>)
zc	Fold an Cursorposition schließen (<i>close</i>)
zd	Fold an Cursorposition entfernen (<i>delete</i>)
zi	Alle Folds öffnen/wieder schließen
:set fdm=syntax	Diese foldmethod macht Code entsprechend der Sprachsyntax faltbar
:set fdm=marker	Diese foldmethod macht das Dokument an den Markerstellen faltbar
:set fmr=<Marker>	Legt den foldmarker fest , z.B. :set fmr={{ {, } }}.

Beispiel zur Anwendung von Foldmarkern:

Der folgende Code wäre faltbar zwischen den Kommentarzeilen, die die Marker enthalten (Kommentarzeilen eingeschlossen):

(i)

```
1 // Kreativer Beispielcode {{{
2 if (this) {
3     that();
4 }
5 // }}}}
```

```
1 +-- 5 Zeilen: Kreativer Beispielcode -----
```

(Cheatsheet auf Basis des tuts+-Kurses „Venture Into Vim“, <https://tutsplus.com/course/venture-into-vim/>)