7.1 Spezielle Anwendungen/vi

7.1.1 Über vi

vi ist ein Editor, der auf nahezu allen Linux- und Unix-artigen Systemen zu finden ist. Manchmal muss man ein System untersuchen, das nur den vi an Bord hat. Deshalb ist es sehr sinnvoll, zumindest Grundkenntnisse in der Bedienung des vi zu besitzen. Dieser Editor ist für den Anfänger nicht leicht zu bedienen, hat aber andererseits sehr viele Möglichkeiten, die bei der tägliche Arbeit in der Administration helfen können.

7.1.2 Aufruf und erster Text

Nach dem Start von vi blickt man auf einen kleinen Informationstext. Am linken Rand sieht man eine Spalte mit Tilde-Zeichen. Sie zeigen an, dass im aktuellen Textpuffer noch nichts steht. In vielen anderen Editoren kann man jetzt sofort losschreiben. Hier geht das nicht, weil vi noch nicht im richtigen *Modus* ist. vi besitzt nämlich verschiedene Modi (oder auch Betriebsarten): Im Befehlsmodus kann man Befehle eingeben, im Editiermodus den Text. Dazu gibt es noch den Zeilenmodus, in dem man unter anderem Befehle über Dateien und Textpuffer absetzen kann.

Nach dem Start befindet sich vi im Befehlsmodus. Mit der Taste a (=append) kommt man in den Editiermodus. Nun kann man seinen Text eingeben. Die Cursortasten kann man in diesem Modus nicht benutzen. Nur die ⟵ -Taste funktioniert wie gewohnt. Mit Esc kann man die Texteingabe wieder beenden und ist zurück im Befehlsmodus. Wenn man aus Versehen nochmal auf Esc drückt, ist das kein Problem.

Jetzt kann man mit den Cursortasten im eigenen Text herumwandern. Möchte man etwas hinzufügen, kann man wieder die Taste a benutzen oder i oder o:

- a (=append) – einfügen ab dem aktuellen Zeichen
- i (=insert) – einfügen vor dem aktuellen Zeichen
- o (=open line) – einfügen zu Beginn der folgenden Zeile

Nun soll der Text in einer Datei gespeichert werden. Jetzt kommt der Zeilenmodus ins Spiel. Er beginnt mit der Eingabe eines Doppelpunktes. Im Zeilenmodus kann man den Befehl, den man eingibt, in der Fußzeile des Fensters sehen (im Gegensatz zu den Befehlen a, i und o, die bisher eingegeben wurden). Jeder Befehl wird mit ↩ abgeschlossen. Man kann z. B. diese Befehle eingeben:

- `:f hallo.txt` Wir suchen uns den Dateinamen hallo.txt aus.
- `:w` Wir schreiben den Textpuffer in die genannte Datei.
- `:q` Wir beenden den Editor.

Will man den Editor beenden, ohne zu schreiben, geht das mit dem Befehl `:q!` ↩ .

Abbildung 1 zeigt die verschiedenen Modi des vi. Nur die drei genannten sind vorerst relevant. Alle anderen Modi sind nur für den Fall aufgeführt, dass man sie aus Versehen betreten und wieder hinauskommen will.

Am Anfang weiß man eventuell nicht immer, in welchem Modus man sich befindet. Mit dem Befehl `:set showmode` lässt sich dieses Problem lösen.

7.1.3 Wandern und Suchen im Text

Nun soll gezeigt werden, wie man sich im Text bewegt und zu bestimmten Cursorpositionen kommt. Dazu öffnet man wieder vi mit dem Dateinamen als Parameter. Im Befehlsmodus kann man wie bei jedem anderen Editor sich mit den Cursortasten bewegen. Alternativ kann man die Tasten h (links), j (oben), k (unten) und l (rechts, alternativ Leerzeichen) nehmen.

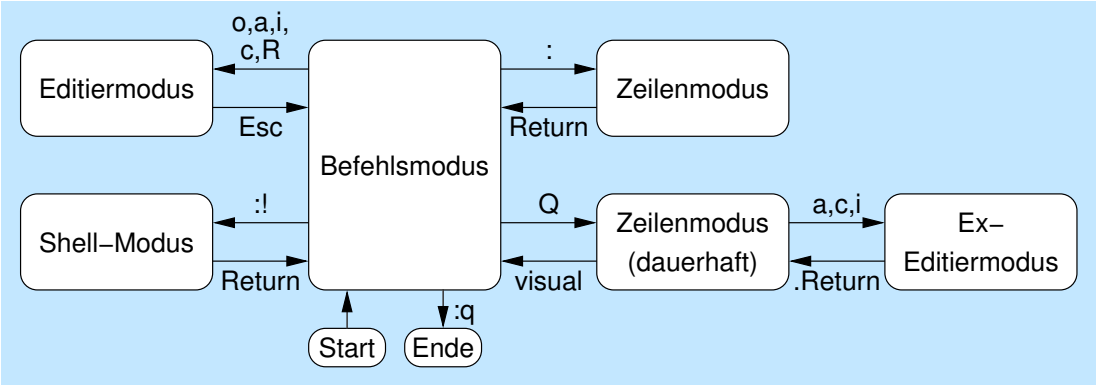

Abbildung 1: Modi des vi

7.1.3.1 Spalten und Zeilen Mit 0 (oder mit `|`)kommt man an den Beginn der Zeile, mit 30`|` zu Spalte 30 und mit `$` an das Ende. Mit `^␣` kommt man zum ersten nicht-Leerzeichen der Zeile (das Leerzeichen muss eingegeben werden, damit das Hütchen wirksam wird). Falls man sich die Zeilenenden anzeigen lassen möchte, kann man das übrigens mit dem Befehl `:set list` einschalten.

Mit der Eingabe 97G kommt man in Zeile 97. Das ist natürlich nur sinnvoll, wenn man die Zeilennummern auch sieht. Dazu gibt es den Befehl `:set number`. Gibt man nur G ein, kommt man an das Textende.

7.1.3.2 Suchen Mit tx und fx kommt man zum nächsten Vorkommen des Zeichens x im Text (kann auch jedes andere beliebige Zeichen sein), wobei tx *vor* dem Zeichen positioniert und fx *darauf*. Mit Tx und Fx geht es genauso, jedoch rückwärts.

Sehr nützlich ist die Taste %, mit der man in einem Klammergebirge wie z. B. `[(a+b)*(c-d)]` zu einer Klammer die korrespondierende Klammer findet[1].

Auch die Suche nach regulären Ausdrücken darf in einem guten Editor nicht fehlen. Nach der Eingabe `/Willi` kann man im ganzen folgenden Text abwärts nach Willi suchen. Mit `?Willi` geht die Suche aufwärts los. Mit der Taste n (=next) sucht man das nächste Vorkommen in derselben Richtung, mit N sucht man weiter in der entgegengesetzten Richtung.

7.1.3.3 Wiederfinden Mit dem Befehl ma kann man die aktuelle Position speichern. Mit der Eingabe von `` `␣a `` springt man zu dieser Position zurück (das Leerzeichen muss nur eingegeben werden, damit der Backtick wirksam wird). Statt a kann jeder Kleinbuchstabe von a bis z verwendet werden, somit erhält man 26 Sprungmarken.

7.1.3.4 Folgen All dieses Wandern und Suchen im Text hat zur Folge, dass der Cursor jeweils auf eine bestimmte Position gesetzt wird. Aber diese Positionen haben noch einen zweiten Zweck: Mehrere wichtige Befehle wie z. B. d (=delete, löscht) oder y (=yank, kopiert in die vi-Zwischenablage) nehmen die Position als Argument: d97G löscht alles vom Cursor bis nach Zeile 97. Und y% kopiert alles von der aktuellen Klammer bis zur korrespondierenden Klammer.

7.1.4 Text bearbeiten

Nun soll der bestehende Text geändert werden. Auch das geschieht im Befehlsmodus. Auch hier öffnet man vi mit dem Dateinamen als Parameter. Nun empfiehlt es sich (zumindest während der Gewöhung an vi), mit `:set showmode` den aktuellen Modus anzeigen zu lassen.

[1] Klappt für eckige, runde und geschweifte, nicht aber für spitze Klammern.

7.1.4.1 Zeile oder Zeichen ersetzen Die Eingabe von R (=Replace) erlaubt es, im Editiermodus die aktuelle Zeile zu ändern. Wenn man `:set showmode` gesetzt hat, wird nun in der Fußzeile `-- REPLACE --` angezeigt. Nach der Änderung verlässt man den Editiermodus mit der Esc -Taste. Mit u wird die letzte Änderung der Zeile wieder aufgehoben. Falls man nur ein Zeichen durch ein anderes ersetzen will, kann man den Befehl r nehmen.

7.1.4.2 Bereich ändern mit c Für zielgenauere Ersetzungen bietet sich der Befehl c (=change) an. Dieser Befehl erwartet als Parameter einen *Bereich*. Die erste Bereichsgrenze liegt in der Regel an der Cursorposition. Für die zweite Bereichsgrenze kommen die Informationen aus dem Abschnitt „Wandern und Suchen im Text“ zum Einsatz: Mit c^ z. B. kann man den Bereich von der aktuellen Cursorposition bis zum ersten nicht-Leerzeichen der Zeile ersetzen. Der Cursor springt daraufhin zum Bereichsanfang, am Bereichsende erscheint ein Dollarzeichen und in der Fußzeile die Meldung `-- INSERT --` (wie beim Befehl). Jetzt kann man seinen neuen Text eingeben. Er darf länger oder kürzer sein als der alte. Der alte Text wird nur noch zur Information angezeigt; er ist nicht mehr im Puffer. Ist man fertig, muss man die Eingabe mit Esc abschließen. Anschließend ist das Ergebnis zu besichtigen. Tabelle 1 zeigt, wie durch c und den Bereichsnamen eine ganze Reihe von Befehlen gebaut werden kann. Die Liste ist nicht vollständig[2]. Die vorletzte Zeile enthält

Befehl	Kurzform	Wirkung
cl	s	Ein Zeichen ändern
c␣		Ein Zeichen ändern
c^␣		Ändern bis zum ersten Nicht-Leerzeichen der Zeile
c$	C	Ändern bis Zeilenende
c0		Ändern bis Zeilenanfang
c27|		Ändern bis Spalte 27
c97G		Ändern bis Zeile 97
cG		Ändern bis zum Textende (Vorsicht)
cfx		Ändern bis zum nächsten x (einschließlich)
ctx		Ändern bis zum nächsten x (ausschließlich)
cFx		Ändern bis zum vorigen x (einschließlich)
cTx		Ändern bis zum vorigen x (ausschließlich)
c%		Ändern bis zur korrespondierenden Klammer
c/abc		Ändern bis zum nächsten regulären Ausdruck abc
c?abc		Ändern bis zum nächsten regulären Ausdruck abc
c`␣a		Ändern bis zur Marke a
cc	S	Ändern der aktuellen Zeile
cw		Ändern des aktuellen Wortes

Tabelle 1: Einige c-Befehle beim vi

eine Besonderheit: Die Wiederholung des Befehlsnamens bedeutet, dass sich dieser Befehl auf die gesamte aktuelle Zeile beziehen soll.

7.1.4.3 Bereich in temporären Puffer kopieren mit y Mit dem Befehl y (=yank, reißen, kopieren) kann man einen Bereich in einen temporären Puffer kopieren. Der originale Bereich bleibt dabei unverändert. Mit yl (oder y␣) kopiert man ein einzelnes Zeichen, mit yy (Kurzform Y) die ganze Zeile. Andere Bereiche funktionieren genauso wie beim c-Befehl. Mit dem Befehl p (=put) kann man den Pufferinhalt nach der aktuellen Position wieder einfügen.

7.1.4.4 Bereich löschen mit d Mit dem Befehl d (=delete) kann man einen Bereich löschen. dl (oder d␣) löscht ein einzelnes Zeichen, dd die ganze Zeile. Wiederum kann man wie beim c-

[2]Es gibt noch Parameter für Langworte (können Sonderzeichen enthalten), Sätze, Absätze und Abschnitte sowie für Anzeigefenster.

Befehl andere Bereichs zum Löschen angeben (Beispiele siehe Tabelle 2). Der gelöschte Bereich

Befehl	Kurzform	Wirkung
`dl`	`x`	Ein Zeichen löschen
`dh`	`X`	Ein Zeichen vor dem Cursor löschen
`d$`	`D`	Löschen bis Zeilenende
`dd`		Löschen der aktuellen Zeile
`dw`		Löschen des aktuellen Wortes

Tabelle 2: Einige d-Befehle beim vi

steht anschließend für den p-Befehl zur Verfügung, kann also woanders wieder eingefügt werden.

7.1.4.5 Zeilenumbruch einfügen und löschen Wenn man einige der Kommandos ausprobiert hat, kann es sein, dass ein Zeilenumbruch zu wenig oder zu viel vorhanden ist. Einfügen kann man einen Zeilenumbruch ganz normal mit dem i-Befehl: i ↩ Esc . Zum Löschen eines Zeilenumbruchs an der aktuellen Position gibt es den Befehl J (=join).

7.1.4.6 Für Programmierer: Bereich einrücken Mit den Befehlen < und > kann man Bereiche nach links oder rechts verschieben. Hat man mit `:set list` die Anzeige von Sonderzeichen sichtbar gemacht, bekommt man statt der Einrückung `^I` (=Tabulator) zu sehen. Auch hier kann man wie beim c-Befehl andere Bereichs zum Löschen angeben. Allerdings kann mindestens eine ganze Zeile verschoben werden, nicht ein Teil davon. Gibt man `>l`, `>w` oder `>$`ein, wird einfach die ganze aktuelle Zeile verschoben. Mit `:set shiftwidth=4` kann man übrigens festlegen, dass immer um vier Leerzeichen verschoben werden soll.

7.1.4.7 Bereich filtern Mit dem Befehl `!` schickt man einen Bereich an ein Filterprogramm und ersetzt ihn durch die Ausgabe des Programms. Mit `!!tr "A-Z" "a-z"` wandelt man die aktuelle Zeile in Kleinbuchstaben um.

7.1.5 Mehrere Bereiche

Mit dl löscht man ein Zeichen, mit d5l fünf aufeinanderfolgende Zeichen. Allgemein gibt: Setzt man vor die Bereichsangabe eine Zahl, kann man mehrere aufeinanderfolgende Bereiche, z. B. Zeichen, Zeilen, Suchbereiche usw. bearbeiten.

7.1.6 Wiederholfunktion

Mit >> schiebt man die Zeile einmal nach rechts, mit `3>>` schiebt man sie dreimal nach rechts. Allgemein gilt: Setzt man vor den Befehlsnamen eine Zahl n, wird der Befehl n-mal ausgeführt.

7.1.7 Puffer und p-Befehl

Die Befehle c, d und y speichern den Bereich, den sie ändern, löschen oder kopieren in einem Textpuffer ab. Diese Textpuffer sind durchnumeriert von 1 bis 9. Der zuletzt benutzte Puffer hat die Nummer 1, der vorletzte die Nummer 2 usw. Mit dem Befehl `"3p` kann man sich den Puffer mit der Nummer 3 an die aktuelle Position holen.

Zusätzlich gibt es noch weitere 26 Puffer mit den Namen a bis z. Während die Puffer 1 bis 9 dynamisch sind, kann man in den benamten Puffern a bis z Daten bis zum Programmende speichern. Sie werden wie folgt benutzt: Mit `"ayy` schreibt man die aktuelle Zeile in den Puffer a. Mit `"ap` holt man seinen Inhalt an die aktuelle Position.

SZ	Bedeutung	SZ	Bedeutung	SZ	Bedeutung
l	aktuelles Zeichen	fx	auf nächstem x	W	nächstes Langwort
^␣	log. Zeilenanfang	tx	vor nächstem x	E	Langwortende
0	phys. Zeilenanfang	Fx	auf vorigem x	B	Langwortbeginn
20|	Spalte 20	Tx	vor vorigem x	(	Satzbeginn
$	Zeilenende	/abc	auf nächstem abc	)	Satzende
H	Fensteranfang	?abc	auf vorigem abc	{	Absatzbeginn
M	Fenstermitte	%	korresp. Klammer	{	Absatzbeginn
L	Fensterende	w	nächstes Wort	[[	Abschnittsbeginn
4G	Zeile 4	e	Wortende	]]	Abschnittsende
G	letzte Zeile	b	Wortbeginn	`␣a	Marke a

Tabelle 3: Bereiche

7.1.8 Sprungziele/Bereiche im Überblick

Tabelle 3 zeigt häufig benutzte Sprungziele bzw. Bereiche für die Befehle c, y, p, >, < und !. Ein Langwort kann Sonderzeichen enthalten, ein Satz endet mit einem Punkt (danach zwei Leerzeichen), ein Absatz mit einer Leerzeile und ein Abschnitt mit einem Seitenvorschub.

7.1.9 Befehle im Überblick

Syntax für einen vi-Befehl: `["Puffername][Anzahl1]Befehlsname[Anzahl2][Bereich]`
Tabelle 4 zeigt noch einmal kurz die wichtigsten (?) Befehle im Befehlsmodus des vi. Von den

Befehl	Bedeutung	Bereich	Fußzeile	Ende mit Esc
a	Anhängen	—	INSERT	ja
i	Einfügen	—	INSERT	ja
o	Neue Zeile	—	INSERT	ja
R	Ersetzen	Zeile	REPLACE	ja
r	Ersetzen	Zeichen	—	—
c	Ändern	ja	INSERT	ja
y	Kopieren	ja	—	—
p	Einfügen	—	—	—
d	Löschen	ja	—	—
>	rechts einrücken	ja	—	—
<	links einrücken	ja	—	—
!	Filter anwenden	ja	:.!	—
m	Marke setzen	Zeichen	—	—

Tabelle 4: Befehle im Befehlsmoduls

vielen im Zeilenmodus möglichen Befehlen braucht man am Anfang nur sehr wenige; eine Auswahl zeigt Tabelle 5.

Befehl	Bedeutung
:q	Ende
:q!	Ende trotz Warnung
:w	Schreiben des Puffers in die aktuelle Datei
:w x.txt	Schreiben des Puffers in die Datei x.txt
:f x.txt	Dateiname x.txt wählen

Tabelle 5: Befehle im Zeilenmodus

7.1.10 undo und redo

Das Wichtigste zum Schluss: Mit der Taste u (oder im Zeilenmodus `:undo`) bekommt man eine Undo-Funktion, man kann also damit den letzten Befehl (in vim beliebig viele Befehle) rückgängig machen.

Eine Redo-Funktion, mit der man die letzte Undo-Funktion wieder aufheben kann, bekommt man beim vi durch nochmalige Eingabe von u. Beim vim bekommt man sie durch die Tastenkombination Strg - r (oder im Zeilenmodus `:redo`).