Regionales Rechenzentrum für Niedersachsen (RRZN)	UNX 7
Universität Hannover	Januar 1996
Gültig für Sun-Cluster, HP-Cluster, Workstation-Pool	

Bedienung des Editors ex/vi

Der Editor ex/vi gehört zu den meistverwendeten Programmen an UNIX-Systemen, obwohl er nicht gerade einfach zu bedienen ist. Grund für seine "Beliebtheit" ist die Tatsache, daß er auf (fast) jedem UNIX-System vorhanden ist, kaum Anforderungen an die Terminalausstattung stellt und, insbesondere in der zeilenorientierten Variante ex, ein wichtiges Werkzeug für Systemverwalter ist. Wer also an verschiedenen, unterschiedlich administrierten UNIX-Systemen arbeiten oder ein UNIX-System verwalten möchte, muß mit dem ex/vi vertraut sein.
Erheblich einfacher zu bedienen ist der Editor Micro-EMACS. Das RRZN hat ihn so konfiguriert, daß er auch für gelegentliche Benutzer gut handhabbar ist und empfiehlt den Micro-EMACS als Editor für den "Normalbenutzer".

Die vorliegende Beschreibung wurde auf Grundlage der Schrift „Bedienung des ex/vi–Editors", C. Rank, Fakultät für Mathematik und Informatik an der Universität Passau, erstellt. Für die freundliche Überlassung dieser Schrift möchten wir an dieser Stelle herzlich danken.

Obwohl dieser Umdruck nach bestem Wissen erstellt wurde, übernimmt das RRZN keine Garantie für seine Korrektheit.

1 Einführung

1.1 ex und vi – ein Editor in zwei Betriebsarten

Für das Betriebssystem UNIX steht eine Vielzahl von Editoren zur Verfügung. Die Editorenfamilie ex und vi, in Berkeley zusammen mit UNIX 4.1 entwickelt, hat den Vorteil, auf nahezu allen Terminaltypen lauffähig zu sein. Während ex zeilenorientiert arbeitet, ist vi ein full-screen-Editor, d.h. man sieht auf dem Bildschirm den gerade bearbeiteten Textausschnitt und alle Änderungen, die daran vorgenommen werden. Der vi ist also ein praktisches Werkzeug zum Eingeben und manuellen Editieren von beliebigen Texten; ex dagegen ist besonders geeignet für die Bearbeitung von schon vorhandenen Texten anhand von sogenannten regulären Ausdrücken, die zum Auffinden, Streichen oder Ersetzen bestimmter Textstellen verwendet werden können. Der Clou jedoch ist, daß ex und vi zwei Betriebsarten ein- und desselben Editors sind. Der Vorteil dieser Tatsache ist, daß auch im vi-Modus ex-Kommandos ausgeführt werden können. Umgekehrt kann man vom ex-Modus in den vi-Modus überwechseln. Trotzdem wird in diesem Dokument meist nicht vom vi- bzw. ex-Modus, sondern vom Editor vi bzw. ex gesprochen. Dies hat durchaus seine Berechtigung, da man ohne weiteres mit vi arbeiten kann, ohne den ex zu kennen (und umgekehrt), d.h. jede der beiden Betriebsarten des Editors erscheint dem Benutzer als eigener Editor.

1.2 Zum Konzept des Editors vi

Für Anwender, die bereits auf anderen Systemen mit full-screen-Editoren gearbeitet haben, seien hier ein paar Worte zur Bedienungsfreundlichkeit des vi gesagt: Im vi ist es nicht so einfach möglich, zwischen verschiedenen Textstellen hin- und herzuspringen und sofort zu editieren. Dieser Nachteil kommt vom Konzept des vi, der zwischen dem *Kommandomodus*, dem *Einfügemodus* und dem *ex-Modus* unterscheidet. Die Bedienung ist daher vor allem für Anfänger sehr umständlich. Warum dieses Konzept?

Zur Beantwortung dieser Frage muß man sich vor Augen führen, daß UNIX ein Betriebssystem ist, das auf den unterschiedlichsten Rechenanlagen läuft, an denen die verschiedensten Terminals angeschlossen sein können. Mit dem vi soll man aber möglichst auf allen Terminals arbeiten können, und da nicht jedes Terminal über spezielle Funktionstasten verfügt, muß man von dem ausgehen, was alle Terminals gemeinsam haben: Eine Tastatur, auf der Groß- und Kleinbuchstaben, Zahlen, Satzzeichen und noch ein paar Sonderzeichen getippt werden können. Zusätzlich sind auf den Terminals noch besondere Tasten vorhanden: Einmal die `ESCAPE`-Taste (im folgenden mit `ESC` bezeichnet). Diese dient beim vi zum Beenden des Einfügemodus. Die `RETURN`-Taste hat im vi im großen und ganzen die gleiche Funktion wie sonst auch: Sie dient im Einfügemodus zum Erzeugen von neuen Zeilen. Mit der DELETE- (oder DEL-) Taste kann im Einfügemodus jeweils das letzte getippte Zeichen gelöscht werden. Die `SHIFT`-Taste hat nur eine Wirkung, wenn sie gleichzeitig mit einer anderen Taste gedrückt wird. Diese Aktion wird im folgenden mit `<SHIFT-Taste>` bezeichnet. Die `SHIFT`-Taste wird in Verbindung mit Buchstabentasten zur Erzeugung von Großbuchstaben verwendet, daher wird in dieser Beschreibung z.B. die Tastenkombination `<SHIFT-a>` einfach mit A bezeichnet. Eine weitere Taste, die nur in Verbindung mit einer anderen Taste etwas bewirkt, ist die CONTROL–Taste. Im folgenden wird das Betätigen der `CONTROL`–Taste

gleichzeitig mit einer anderen Taste mit <CTRL-Taste> bezeichnet. Da die SHIFT-Taste, wenn sie gleichzeitig mit der CONTROL-Taste und einer Buchstabentaste gedrückt wird, ignoriert wird, sind z.B. <CTRL-A> und <CTRL-a> äquivalent. Dies sind an und für sich die wichtigsten Sondertasten, die auf allen Terminals, die dem ANSI-Standard entsprechen, vorhanden sind. Auf dem Terminal, an dem Sie arbeiten, sind wahrscheinlich noch weitere Tasten vorhanden, aber Sie können nicht davon ausgehen, daß diese vom vi unterstützt werden.

Der Editor vi interpretiert (fast) jede Taste, die am Terminal gedrückt wird, je nach Modus unterschiedlich: Im Einfügemodus haben die Tasten die Wirkung, daß das entsprechende Zeichen in den Text eingefügt wird, im Kommandomodus dagegen wird das der Taste(nkombination) entsprechende Kommando ausgeführt.

2 vi: Grundlegendes

2.1 Aufruf von vi

Die Befehlszeile zum Aufruf von vi lautet

vi *dateiname*

Wird der Editor in der obigen Form aufgerufen, erscheinen nach kurzer Zeit die ersten Zeilen der angegebenen Datei auf dem Bildschirm, am RRZN werden die Zeilen mit Zeilennummern angezeigt. Wenn Sie eine neue Datei anlegen, sehen Sie hingegen lauter Zeilen, die nur aus dem Zeichen ~ bestehen. Allgemein wird eine leere Zeile mit einem ~ am Anfang dann angezeigt, wenn sich die Zeile jenseits des logischen Dateiendes befindet. Der letzten Bildschirmzeile kommt beim vi besondere Bedeutung zu: Sie enthält (wenn vorhanden) gewisse Informationen über die gerade bearbeitete Datei oder auch Fehlermeldungen bei der Ausführung von vi-Kommandos. Diese Zeile wird im folgenden als *Infozeile* bezeichnet. In dieser Zeile werden übrigens auch die sog. *langen* Kommandos angezeigt, die Sie gerade eingeben.

Gleich nach dem Aufrufen von vi wird in der Infozeile der Name der editierten Datei sowie die Zahl der Zeilen und Zeichen, die Datei enthält, angezeigt. Beim Editieren einer neuen Datei wird statt der Zeilen- und Zeichenzahl die Meldung "[New file]" angezeigt. Die Infozeile bleibt immer stehen, bis sie von neuen Editormeldungen überschrieben oder einem neuen Bildschirmaufbau gelöscht wird.

Beispiel:

```
$ vi testfile
     1
~
~
~
~
~
"testfile" [New file]
```

Hinweis: Wenn Sie das obige Beipiel nicht im RRZN, sondern an einem anderen Rechner ausprobieren, kann die Bildschirmausgabe etwas anders aussehen. Der vi kann

unterschiedlich konfiguriert und so der eigenen Arbeitsweise angepaßt werden. So können z. B. Zeilennummern angezeigt werden, man kann aber auch darauf verzichten, wenn man alle Spalten des Bildschirms für die Textanzeige nutzen möchte. Die Konfiguration erfolgt über die Umgebungsvariable EXINIT oder eine Datei .exrc (siehe Kapitel 7)

2.2 Verlassen von vi

Für den Anfang ist auch noch wichtig, wie Sie nach dem Editieren den vi wieder verlassen. Für die folgenden Operationen müssen Sie sich im Kommandomodus befinden (in den Sie auf jeden Fall gelangen, wenn Sie erst die Taste RETURN und anschließend die Taste ESC drücken).

Falls Sie die editierte Datei beim Verlassen des Editors nicht speichern wollen, geben Sie **:q!** ein (tippen Sie alle drei Zeichen ein, erst den Doppelpunkt, dann den Buchstaben q, dann das Ausrufungszeichen). Diese Eingabe sehen Sie in der Infozeile angezeigt, und Sie müssen dieses Kommando mit der RETURN-Taste abschließen, wenn es ausgeführt werden soll.

Wenn Sie den Editor verlassen und die Datei abspeichern wollen, gibt es zwei Möglichkeiten. Die erste besteht darin, daß Sie zunächst den Inhalt der Datei abspeichern und anschließend den vi verlassen. Beide Schritte werden zusammengefaßt in dem Kommando **:wq**, welches Sie mit der RETURN-Taste abschließen. Sie erhalten einige Informationen über die gespeicherte Datei, bevor wieder der übliche UNIX-Prompt erscheint. Von dieser unterscheidet sich die zweite Möglichkeit darin, daß die Datei nur dann abgespeichert wird, wenn sie verändert worden ist. das Kommando lautet **Z Z** (SHIFT-Taste drücken und gleichzeitig zweimal die Taste z betätigen). Wurde die Datei im vi verändert, erhalten Sie wieder Informationen über die gespeicherte Datei, bevor wieder der übliche UNIX-Prompt erscheint. Verabschiedet sich der vi ohne Anzeige dieser Informationen, dann wurde die Datei gar nicht im vi verändert. Daher brauchte sie auch nicht abgespeichert zu werden (also keine Panik, alles ist in Ordnung). Den kleinen Unterschied zwischen **:wq** und **ZZ** sehen Sie am besten, wenn Sie den vi für eine neue Datei aufrufen und sofort mit **:wq** bzw. **ZZ** wieder verlassen; im ersten Fall haben Sie in Ihrem aktuellen Verzeichnis eine leere Datei erzeugt.

:q! RETURN	Verlassen **ohne** Sichern
:wq RETURN	Verlassen **mit** Sichern
Z Z	Verlassen **mit** Sichern

2.3 Kommandomodus, Einfügemodus, ex-Modus

Direkt nach dem Aufrufen des vi befindet man sich im **Kommandomodus**. Der Kommandomodus ermöglicht – wie der Name schon sagt – die Eingabe von verschiedenen Editorkommandos. Die Kommandos können unterschieden werden in einfache Kommandos, die durch einen einzigen Tastendruck ausgelöst werden, und zusammengesetzte Kommandos, für die mehrere (bis zu fünf) Tasten hintereinander betätigt werden müssen. Der vi zeigt im Kommandomodus keine Tastendrücke an. Die Wirkung eines Kommandos sehen Sie also nur durch die Änderungen, die das Kommando in der bearbeiteten Datei verursacht. Wenn Sie sich bei einem zusammengesetzten Kommando vertippen, können Sie das unvollständige Kommando durch Drücken der `ESC`-Taste löschen (eine Ausnahme bilden die später erklärten Kommandos / und ?, bei denen die `ESC`-Taste die Wirkung der `RETURN`-Taste zeigt).

Vor die meisten Kommandos können Zählfaktoren (*counts*) gesetzt werden, die angeben, auf welche Zeile bzw. Spalte sich das Kommando bezieht bzw. wie oft das Kommando ausgeführt werden soll. Einen Zählfaktor geben Sie einfach durch Tippen der gewünschten Zahl vor dem Kommando an (dies wird ebenfalls nicht auf dem Bildschirm angezeigt, kann aber mit `ESC` zurückgenommen werden).

Der **Einfügemodus** (richtiger wäre eigentlich Editiermodus, da manchmal in diesem Modus auch überschrieben wird) dient zum direkten Einfügen von Text in die Datei. Im Einfügemodus wird (fast) jedes Zeichen, das von der Tastatur kommt, einfach an die Stelle eingefügt, an der sich der Cursor gerade befindet, und der Cursor wird dann ein Zeichen nach rechts gesetzt, was mit einer Verschiebung des rechts vom Cursor stehenden Textes (d.h. des Zeilenrestes) einhergeht.

Vom Kommandomodus aus gelangen Sie in den Einfügemodus unter anderem durch Eingabe eines der Kommandos **`i`**, **`I`**, **`a`**, **`A`**, **`o`**, **`O`**, **`s`** oder **`S`** (was diese genauer bewirken, wird später erklärt). Ganz wichtig ist, wie Sie aus dem Einfügemodus wieder zurück in den Kommandomodus kommen. Im Einfügemodus wird ja fast jedes Tastaturzeichen als in den Text einzufügendes Zeichen angesehen. Eine Ausnahme ist hier die `ESC`-Taste: Sie veranlaßt eine Beendigung des Einfügemodus und eine Rückkehr in den Kommandomodus (Hinweis für Anfänger: mehrfaches Drücken der `ESC`-Taste ist idempotent, man kann also nichts falsch machen).

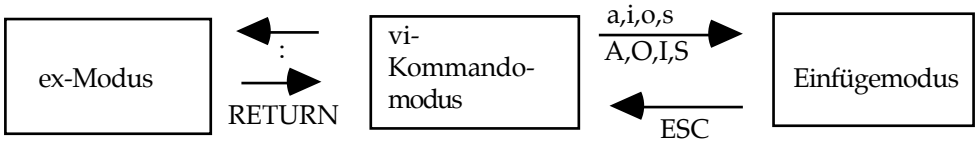
Eine weitere Sonderfunktion hat die `DEL`-Taste. Mit ihr können Sie während der Eingabe Tippfehler verbessern, denn sie bewirkt, daß das Zeichen vor dem Cursor gelöscht und der Cursor auf dieses Zeichen gesetzt wird. Leider sehen Sie als Wirkung von `DEL` nur, daß der Cursor um ein Zeichen nach links bewegt wird, daß hier etwas gelöscht wird, ist nicht zu sehen. Daß ein Zeichen gelöscht wurde, sehen Sie erst, wenn Sie den Einfügemodus wieder verlassen oder gelöschte Zeichen überschreiben. Bitte beachten Sie, daß Sie mit `DEL` nur Zeichen auf der aktuellen Zeile löschen können (Rücksprung in vorhergehende Zeile ist nicht möglich), außerdem kann auf diese Weise nur neu eingefügter Text gelöscht werden. Ein Drücken der `DEL`-Taste gleich nach dem Betreten des Einfügemodus ist also in keinem Fall erfolgreich.

Beachten Sie bitte, daß der Unterschied zwischen Kommando- und Einfügemodus auf dem Bildschirm nicht unbedingt sichtbar ist; in welchem Modus Sie gerade sind, merken

Sie im Zweifelsfall erst, wenn Sie eine Taste drücken: Wenn das entsprechende Zeichen dann in den Text eingefügt wird, befinden Sie sich im Einfügemodus. Es gibt allerdings ein Mittel, um sicherzustellen, daß Sie im Kommandomodus sind: Drücken Sie die ESC-Taste; danach ist auf alle Fälle der Kommandomodus ausgewählt (wenn Sie schon im Kommandomodus waren, wird zwar das Terminal kurz piepen, das hat aber weiter keine Bedeutung).

Neben dem Kommando- und dem Einfügemodus gibt noch einen dritten Editormodus, den sogenannten **ex-Modus**. In diesen Modus müssen Sie fast immer verzweigen, wenn Sie ein Kommando absetzen wollen, welches größere Bereiche der Datei oder die gesamte Datei betrifft (z.B. Sichern der Datei, Einlesen einer anderen Datei, globales Ersetzen). Wählen Sie den ex-Modus durch Drücken der Taste **:** (Doppelpunkt) an; in der Infozeile erscheint dann der Doppelpunkt, dahinter wird der Cursor positioniert. Nun können Sie ein ex-Kommando eintippen. Im Unterschied zum normalen Kommandomodus sehen Sie hier, was Sie eingeben, und es ist auch möglich, mit der DEL-Taste die Tippfehler zu korrigieren. Ein vollständiges ex-Kommando übergeben Sie dem vi mit der RETURN-Taste zur Ausführung. Wurde das Kommando ausgeführt, kehrt vi wieder in den Kommandomodus zurück. Der ex-Modus wird also mit : zur Eingabe nur eines einzigen ex-Kommandos angewählt; sind Sie so versehentlich in den ex-Modus geraten, drücken Sie einfach RETURN, um ihn abzubrechen (evt. nach Löschen aller bisher eingegebenen Zeichen mit der DEL-Taste). Sie können auch für mehrere Kommandos in den ex-Modus verzweigen, dazu siehe Kapitel 11.

Noch ein Wort zu den Fehlermeldungen des vi: Wenn Sie im Kommandomodus eine unerlaubte Taste drücken, weist der vi durch ein akustisches Signal darauf hin. Kann hingegen ein Kommando aus irgendwelchen Gründen nicht ausgeführt werden, erscheint eine Fehlermeldung in der Infozeile (das gilt auch für den ex-Modus).



3 vi: Bewegen im Text

3.1 Die Cursortasten

Beim Arbeiten mit vi sehen Sie auf dem Bildschirm auf einem Zeichen eine Markierung, die je nach Terminaleinstellung ein inverses Quadrat oder eine Unterstreichung darstellt, vielleicht blinkt sie auch zusätzlich noch. Diese Markierung ist der sogenannte Cursor. Die Cursorposition bestimmt beim Einfügemodus die Stelle, an der das nächste Zeichen in den Text eingesetzt wird. Im Kommandomodus zeigt sie die Textstelle (manchmal auch nur die Zeile) an, auf die sich ein folgendes Kommando bezieht.

Wenn ihr Terminal über Cursortasten verfügt, können Sie damit (im Kommandomodus!) den Cursor an eine beliebige Textstelle bewegen. Beim Betätigen einer Cursortaste be-

wegt sich der Cursor in die Richtung, in die der Pfeil auf der Taste zeigt. Steht der Cursor auf der letzten Bildschirmzeile, so wird beim Bewegen nach unten der Bildschirm um eine Zeile nach oben gescrollt. Entsprechendes gilt für die Cursorbewegung nach oben, wenn der Cursor bereits in der ersten Bildschirmzeile steht; dann wird der Bildschirm um eine Zeile nach unten gescrollt. Beim Versuch, den Cursor jenseits des Dateianfangs oder Dateiendes zu bewegen, gibt vi ein akustisches Signal.

Mit den Cursor-links- und Cursor-rechts-Tasten kann der Cursor nach links oder rechts bewegt werden, allerdings nur innerhalb der aktuellen Zeile. Das heißt, daß der Cursor nicht automatisch nach dem Erreichen des Zeilenanfangs bzw. -endes in die vorhergehende bzw. nächste Zeile gelangt, vielmehr gibt es bei einem entsprechenden Versuch wieder ein akustisches Signal.

Beim Bewegen des Cursors nach oben oder unten gibt es noch eine Nebenwirkung: Würde der Cursor dabei in eine Spalte kommen, die jenseits des logischen Zeilenendes liegt, wird der Cursor auf das Zeilenende gesetzt. Der Editor merkt sich aber, daß der Cursor in Wirklichkeit in eine andere Spalte gehört, d.h. wenn der Cursor wieder auf eine genügend lange Zeile gesetzt wird, springt er auch wieder in die richtige Spalte.

Wenn Ihr Terminal keine Cursortasten hat, nimmt der vi auch die Tasten **h**, **j**, **k**, **l** als Ersatz an:

h	←	Cursor nach links
j	↓	Cursor nach unten
k	↑	Cursor nach oben
l	→	Cursor nach rechts

Alle Cursorsteuerungskommandos können mit einem Zählfaktor versehen werden; dieser gibt dann an, um wieviele Zeilen oder Spalten der Cursor versetzt werden soll.

3.2 Cursorbewegungen am Bildschirm

Im folgenden werden weitere Kommandos angegeben, die den Cursor am Bildschirm hin- und herbewegen. Würde der Cursor dabei aus dem physikalischen Textfenster wandern, wird der Bildschirm um eine Zeile nach unten oder oben weitergeblättert (sogenanntes Scrolling).

Die folgenden zwei Kommandos wirken ähnlich wie die Cursor-oben- und Cursor-unten-Tasten, nur mit dem Unterschied, daß gleichzeitig der Cursor auf das erste nicht-Blank-Zeichen in der Zeile gesetzt wird. **+** oder RETURN bewegen den Cursor mit diesem Effekt um eine Zeile nach unten, **–** wirkt in die Gegenrichtung. Die Angabe von Zählfaktoren wirkt wie bei den gewöhnlichen Cursorsteuerungstasten.

Die nächsten drei Kommandos bewegen den Cursor am Bildschirm, ohne daß der Dateiausschnitt verändert wird (also kein Scrollen). Mit **H** (**h**igh) setzen Sie den Cursor an den Anfang (genauer: auf das erste nicht-Blank-Zeichen) der ersten Bildschirmzeile. Eine Zählung wird hier als die Nummer einer Bildschirmzeile interpretiert, auf die

der Cursor gesetzt werden soll, d.h. mit `4H` springt der Cursor an den Anfang der vierten Bildschirmzeile. Entsprechend wirkt **`L`** (**l**ow), nur daß sich hier alles auf die letzte Bildschirmzeile bezieht, also geht mit `3L` der Cursor in die drittletzte Bildschirmzeile. Um den Cursor in die mittlere Bildschirmzeile zu positionieren, gibt es schließlich noch das Kommando **`M`** (**m**iddle), bei dem eine Zählung ausnahmsweise mal nicht sinnvoll ist.

`+`	Anfang der nächsten Zeile
`RETURN`	Anfang der nächsten Zeile
`-`	Anfang der vorhergehenden Zeile
`H`	erste Bildschirmzeile
`M`	mittlere Bildschirmzeile
`L`	letzte Bildschirmzeile

3.3 Cursorbewegungen in einer Zeile

Die folgenden Kommandos dienen zum schnellen Auffinden einer bestimmten Stelle innerhalb einer Textzeile. Um den Anfänger nicht mit einem Wust von Kommandos zu verwirren, werden hier nur die wichtigsten beschrieben, die diesem Zweck dienen. Für eine vollständige Erläuterung aller Kommandos schlagen Sie bitte im Abschnitt 10.2 nach.

Eine häufig benötigte Funktion ist das wortweise Bewegen des Cursors. Ein Wort ist im vi eine Zeichenkette, die durch Blanks oder Satzzeichen begrenzt ist. Das Kommando w (**w**ord forward) bewegt den Cursor an den Anfang des nächsten Wortes und, falls dies notwendig ist, in eine neue Zeile. Entsprechend wirkt b (word **b**ackward) in der anderen Richtung, d.h. der Cursor wird an den Anfang des vorhergehenden Wortes gesetzt. Ferner kann man mit e (**e**nd of word) den Cursor ans Ende des nächsten Wortes setzen. Zu diesen drei Kommandos gibt es auch großgeschriebene Äquivalente, die dasselbe machen, allerdings wird hier ein Wort als eine nur von Leerzeichen begrenzte Zeichenkette aufgefaßt, d.h. daß sich dann in einem Wort auch Satzzeichen befinden können. Zählungen bewirken in Verbindung mit diesen sechs Kommandos die übliche Wiederholfunktion.

Hinweis: Allgemein kann man beim vi sagen, daß Großbuchstabenkommandos oft die gleiche Funktion wie die entsprechenden Kleinbuchstabenkommandos besitzen; allerdings beziehen sich diese auf größere Wirkungsbereiche bzw. bei gerichteten Bereichen auf die entgegengesetzte Richtung.

An den Anfang oder das Ende einer Zeile kann man mit folgendesn Kommandos positionieren: 0 (Ziffer 0) setzt den Cursor auf den Zeilenanfang (wirklich auf das erste Zeichen), während `^` den Cursor auf das erste nicht-Blank-Zeichen bewegt. Ans Zeilenende kann der Cursor mit **`$`** gebracht werden. Besonders nützlich für längere Zeilen ist das Kommando **`|`**, das den Cursor auf diejenige Spalte setzt, die als Zählung vor dem Kommando angegeben wird (oder auf die erste Spalte, falls keine Zählung vorhanden ist).

w	Anfang des nächsten Wortes oder Sonderzeichen
e	Ende des nächsten Wortes oder Sonderzeichen
b	Anfang des vorhergehenden Wortes oder Sonderzeichen
W	Anfang des nächsten Wortes
E	Ende des nächsten Wortes
B	Anfang des vorhergehenden Wortes
0	Zeilenanfang
^	erste nicht-Blank-Zeichen am Zeilenanfang
$	Zeilenende
|	Spalte 1
n **|**	Spalte *n*

3.4 Zeilen- und seitenweises Scrollen

Die folgenden Kommandos sind hilfreich, wenn die Datei seitenweise durchgeblättert werden soll. CTRL-D scrollt den Bildschirm um eine halbe Bildschirmseite nach oben. Der Cursor bleibt jedoch in der gleichen physischen Bildschirmzeile (nicht Dateizeile) stehen. CTRL-U scrollt den Bildschirm um eine halbe Bildschirmseite nach unten. Bei diesen beiden Kommandos ist eine Zählung zugelassen, die die Zeilenzahl angibt, um die gescrollt werden soll. Diese wird gleichzeitig als Standardwert für alle folgenden CTRL-D- und CTRL-U-Kommandos eingestellt, der so lange wirkt, bis wieder eine neue Zählung angegeben wird.
Um gleich über ganze Bildschirmseiten hinwegzublättern, gibt es die Kommandos CTRL-F, das um eine Bildschirmseite vorwärts scrollt und CTRL-B das um eine Bildschirmseite rückwärts scrollt. Hier kann der übliche Wiederholungsfaktor angegeben werden. Zum Scrollen um eine Zeile nach oben bzw. unten (ohne daß sich der Cursor unbedingt in der letzten bzw. ersten Bildschirmzeile befindet) verwenden Sie die Kommandos CTRL-E bzw. CTRL-Y. Bei Anwendung eines solchen Kommandos bleibt der Cursor in der aktuellen Dateizeile stehen, falls dies möglich ist, d.h. er wird mit nach oben oder unten gescrollt. Eine angegebene Zählung wird als Wiederholungsfaktor interpretiert.

Das Kommando z bietet eine weitere Möglichkeit, den Bildschirm zu justieren. Je nachdem, ob Sie danach RETURN, . (den Punkt) oder - drücken, wird die Cursorzeile an den Bildschirmanfang, die Bildschirmmitte oder das Bildschirmende positioniert.

CTRL-D	halbe Seite nach unten blättern
CTRL-U	halbe Seite nach oben blättern
CTRL-F	ganze Seite nach unten blättern
CTRL-B	ganze Seite nach oben blättern
CTRL-E	eine Zeile nach unten blättern
CTRL-Y	eine Zeile nach oben blättern
z RETURN	Cursorzeile an Bildschirmanfang
z .	Cursorzeile in Bildschirmmitte
z -	Cursorzeile an Bildschirmende

3.5 Textstelle suchen

Eine Möglichkeit zum gezielten Suchen einer Textstelle bietet das Kommando **/**. Tippen Sie dieses Kommando, so erscheint der Schrägstrich auch in der Infozeile, dahinter wird der Cursor gesetzt. Dies ist eine Aufforderung an Sie, die Zeichenkette einzugeben, die Sie suchen wollen. Hier sehen Sie wieder, was Sie eingeben, darum können Sie mit der DEL-Taste korrigieren. Nach Eingabe des Strings drücken Sie RETURN, damit die Suche beginnen kann. Gesucht wird immer vorwärts ab der aktuellen Cursorposition. Wenn die gewünschte Zeichenkette jedoch nicht im Rest des Textes gefunden wird, sucht der Editor noch vom Dateianfang an nach ihr, d.h. die Suche ist zyklisch. Erst wenn auch hier die Zeichenkette nicht gefunden wird, bricht der Editor mit der Meldung "Pattern not found" ab, ansonsten wird der Cursor auf die gefundene Zeichenkette positioniert.

Da durchaus die gewünschte Zeichenkette im Text mehrmals vorkommen kann, wird beim ersten Mal nicht unbedingt die richtige Textstelle gefunden. Damit man sich das nochmalige Eintippen des Suchstrings sparen kann, gibt es das Kommando **n** (**n**ext), das nach dem nächsten Vorkommen des zuvor gesuchten Strings sucht. Das Kommando **N** macht dasselbe in der umgekehrten Richtung. Wenn Sie übrigens eine Zeichenkette gleich in rückwärtiger Richtung suchen wollen, geht dies mit dem Kommando **?**, das sich ansonsten wie **/** verhält. Falls Sie ein Suchkommando abbrechen wollen, drücken Sie danach einfach RETURN, ohne einen Suchstring einzugeben.

Nach einem Suchvorgang möchte man oft die vor dem letzten Suchbefehl bearbeitete Textstelle wieder ansteuern. Dies kann man mit den Kommandos ``` `` ``` (zweimal `)und `''` (zweimal ') bewirken. ``` `` ``` setzt den Cursor genau auf die alte Textstelle, `''` auf das erste non-white-character dieser Zeile.

`/`*`Suchbegriff`*	Suche in Richtung Dateiende nach *`Suchbegriff`*
`?`*`Suchbegriff`*	Suche in Richtung Dateianfang nach *`Suchbegriff`*
`n`	erneute Suche in dieselbe Richtung nach *`Suchbegriff`*
`N`	erneute Suche in entgegengesetzte Richtung nach *`Suchbegriff`*
`` `` ``	zurück auf Textstelle vor Suchbefehl
`' '`	zurück auf Anfang der vor Suchbefehl aktuellen Zeile

Bei der Suche nach einem Suchbegriff unterscheidet der vi standardmäßig zwischen Groß- und Kleinschreibung. Wenn dies nicht gewünscht ist, z.B. bei der Suche nach einem Variablennamen in einem Fortran-Quelltext, kann man diese Unterscheidung durch Setzen einer Editorvariablen (`ignorecase`) aufheben. (zu Editorvariablen siehe Abschnitt 7.2)

Als Suchbegriff kann auch ein regulärer Ausdruck verwendet werden. Damit kann die Suche eingeschränkt werden (z.B. auf ein Vorkommen der gesuchten Zeichenkette am Zeilenanfang oder Zeilenende), aber auch erweitert werden auf die gleichzeitige Suche nach mehreren Zeichenketten (suche nach Meyer oder Mayer,) Für die Formulierung regulärer Ausdrücke werden sog. Metasymbole verwendet. Diese werden ausführlich in Abschnitt 7.1.4 beschrieben.

3.6 Klammer suchen

Das folgende Kommando erleichtert die Bearbeitung von Programmen: Bei komplizierten Klammerausdrücken kann man durchaus mal in die Lage kommen, nicht mehr zu wissen, wo zu einer öffnenden Klammer die schließende ist (oder umgekehrt). Aber mit dem vi ist es kein Problem, dies festzustellen. Setzen Sie den Cursor auf die problematische Klammer (ob runde, eckige oder geschweifte bzw. öffnende oder schließende ist egal), und geben Sie das Kommando **`%`**. vi wird dann den Cursor auf die zugehörige schließende oder öffnende Klammer setzen; dies aber natürlich nur, wenn sie existiert, falls nicht, gibt es einen Piepston.

3.7 Zeile suchen

Um einfach eine bestimmte Zeile, deren Nummer bekannt ist, anzusteuern, gibt es das Kommando **`G`** (**g**o). Geben Sie vor diesem Kommando die gewünschte Zeilennummer als Zählfaktor an, der Cursor wird dann auf die entsprechende Zeile gesetzt. Fehlt die Zählung vor dem **`G`**, wird der Cursor an das Textende bewegt.

*`n`***`G`**	Cursor auf Anfang von Zeile *n* positionieren
`G`	Cursor auf Anfang der letzten Zeile positionieren

4 vi: Einfache Änderungen am Text

4.1 Ein- und Anfügen

In diesem Abschnitt sollen nun die Kommandos zum Wechseln in den Einfügemodus genauer besprochen werden. Alle nun folgenden Kommandos haben die Wirkung des Umschaltens in den Einfügemodus, sie unterscheiden sich nur durch die Seiteneffekte. Das Standardkommando zum Einfügen ist **i** (**i**nsert). Nach diesem Kommando wird eingegebener Text vor der Cursorposition eingefügt. Da beim Springen auf das Zeilenende der Cursor immer auf das letzte Zeichen der Zeile gesetzt wird, ist mit **i** kein Einfügen hinter dem Zeilenende möglich. Dazu dient das Kommando **a** (**a**ppend), das im Gegensatz zu **i** hinter der Cursorposition einfügt.

Die restlichen Kommandos arbeiten wie **i** oder **a**, nur daß vor Betreten des Einfügemodus noch andere Aktionen durchgeführt werden. Das Kommando **I** bewirkt immer ein Einfügen vor dem ersten nicht white-space-Zeichen der aktuellen Zeile, auch wenn der Cursor nicht am Zeilenanfang stand. Entgegengesetzt dazu kann mit **A** hinter dem Zeilenende eingefügt werden, ohne vorher den Cursor auf das Zeilenende zu setzen. Oft kommt man in die Situation, daß man unterhalb oder oberhalb der aktuellen Zeile eine neue Zeile einfügen möchte. Das Kommando **o** (**o**pen line) erzeugt zunächst eine Leerzeile unterhalb der aktuellen Zeile und setzt dann den Cursor an den Anfang dieser Leerzeile, so daß man nun Text in eine neue Zeile tippen kann. Entsprechend wirkt **O**, allerdings oberhalb der aktuellen Zeile.

Umschalten in den Eingabemodus und nachfolgenden Text einfügen:

i	vor Cursorposition
I	am Anfang der aktuellen Zeile
a	hinter Cursorposition
A	am Ende der aktuellen Zeile
o	in neuer Zeile hinter der aktuellen Zeile
O	in neuer Zeile vor der aktuellen Zeile
ESC	Zurück in Kommandomodus

4.2 Textstellen ersetzen

Beim Editieren kommt es sicher auch mal vor, daß man zwar einen neuen Text eingeben möchte, aber gleichzeitig ein alter Text dadurch überschrieben werden soll. Man könnte natürlich das Problem so lösen, daß man zuerst den alten Text löscht (siehe unten), aber es gibt hier elegantere Methoden (sprich: Kommandos). Alle folgenden Kommandos führen wieder zu einem Wechsel in den Einfügemodus, der nach Eintippen des gewünschten Textes mit ESC verlassen werden muß.

Beginnen wir zunächst ganz einfach: Das Kommando **s** (**s**ubstitute) ersetzt das Zeichen, auf dem der Cursor steht, durch den eingetippten Text (egal wie lange dieser ist). Hier

ist auch eine Zählung überaus sinnvoll; dann kann man nämlich angeben, wieviele Zeichen ab Cursor vom Text ersetzt werden sollen. Das letzte Zeichen, das ersetzt wird, zeigt der vi übrigens durch ein $-Zeichen an. Größere Ersetzungen lassen sich mit **S** vornehmen: Hier wird die ganze aktuelle Zeile durch den eingefügten Text ersetzt (Zählung verwenden, falls mehrere Zeilen ersetzt werden sollen). Ebenfalls sinnvoll ist das Kommando **C** (change), mit dem der Zeilenrest (einschließlich Cursor) ersetzt wird.

Wenn Sie ein bestimmten Textbereich ersetzen wollen, ist es allerdings höchst mühsam, immer die Zeichen oder Zeilen zu zählen, die ersetzt werden sollen. Eine bessere Methode bietet das (Wirkungs-) Kommando **c**, hinter dem der Wirkungsbereich angegeben werden muß, der ersetzt werden soll. Mit den verschiedenen vi-Wirkungsbereichen befaßt sich ausführlicher der Abschnitt 8.4, hier sei nur gesagt, daß **cw** den Rest eines Wortes ab Cursor ersetzt, **cb** hingegen den Anfang eines Wortes bis vor den Cursor. Ein Sonderfall ist das Kommando **cc**, das eine ganze Zeile ersetzt und somit äquivalent zu **S** ist.

Hinweis: Bei einem zusammengesetzten (Wirkungs-) Kommando (die Kommandos **c**, **d**, **y** und **!** werden später erklärt) muß danach ein (Cursor-)Bewegungskommando angehängt werden. Außerdem gilt: Verdopplung (also **cc**, **dd** und **yy**) beziehen sich auf die ganze Zeile, Großschreibung (also **C**, **D**) auf den Rest der Zeile.

Die folgenden Ersetzungskommandos schalten um in den Eingabemodus, der dort eingegebenen Text ersetzt:

s	Zeichen an Cursorposition
n **s**	*n* Zeichen ab Cursorposition
S	aktuelle Zeile
n **S**	*n* Zeilen, beginnend mit der aktuelle Zeile
cc	aktuelle Zeile
*n***cc**	*n* Zeilen, beginnend mit der aktuelle Zeile
C	aktuelle Zeile ab Cursorposition
ESC	Zurück in Kommandomodus

Die oben genannten Kommandos führen Textersetzungen an einer einzigen Stelle der Datei, nämlich an der durch den Cursor gekennzeichneten Stelle durch. Will man dagegen global in der Datei ändern, also jedes (oder fast jedes) Vorkommen einer bestimmten Zeichenreihe durch eine andere ersetzen, muß man in den ex-Modus umschalten. In diesem steht ein globales Ersetzungskommando mit Rückfrage zur Verfügung (siehe Kapitel 12).

4.3 Textstellen löschen und Zeilen zusammenfügen

Bis jetzt wurde nur gesagt, wie man im Einfügemodus neu eingefügte Zeichen löschen kann. Was aber, wenn vorhandene Textstellen gelöscht werden sollen? Hierfür bietet vi ein paar einfache Kommandos an.

Mit **x** löschen Sie das Zeichen auf der Cursorposition, wobei der Rest der Zeile dann um ein Zeichen nach links gezogen wird, der Cursor bleibt in der gleichen Spalte. Wird allerdings das letzte Zeichen in der Zeile gelöscht, rückt der Cursor auf das neue Zeilenende. Eine Zählung vor **x** bewirkt den üblichen Wiederholungseffekt, also löscht z.B. 5x die nächsten fünf Zeichen ab Cursor; **x** löscht das Zeichen links vom Cursor.

Zum Löschen größerer Textteile auf einmal gibt es das Kommando **d** (**d**elete), das von der Bezeichnung eines Textbereiches gefolgt werden muß (siehe Abschnitt 8.4). Hier wird zunächst dieses Kommando nur für zwei Textbereiche angegeben: **dw** löscht den Rest eines Wortes ab Cursor, **db** entsprechend den Anfang bis ausschließlich Cursor. Eine ganze Zeile kann durch Tippen von **dd** gelöscht werden. Auch hier gilt für Zählungen die Wiederholungsfunktion.

Der Rest einer Zeile (ab Cursor) kann einfacher durch das Kommando **D** gelöscht werden. Im Zusammenhang mit dem Löschen sei hier auch noch den Befehl **r** (**r**eplace) erwähnt. Hinter **r** muß ein Zeichen angegeben werden, das dann das Zeichen auf der Cursorposition ersetzt. Eine Zählung hat hier den Effekt, daß die angegebene Zahl von Zeichen ab Cursor durch das gewünschte Zeichen ersetzt wird.

Unter den Punkt *Textstellen löschen* fällt auch ein Problem, das man bei der Benutzung eines Editors häufiger hat: Was macht man, wenn man zwei Zeilen zu einer zusammenfassen möchte? Die bis jetzt besprochenen Löschkommandos können nicht dazu verwendet werden, ein Zeilenende zu löschen und so Zeilen zusammenzuhängen. Für diesen Zweck gibt es das Kommando **J** (**j**oin) (nicht zu verwechseln mit dem Cursorbewegungskommando **j**), das die folgende an die aktuelle Zeile anhängt. Mit einer Zählung können auch mehrere Zeilen auf einmal zusammengehängt werden. Will man eine Zeile an einer bestimmten Stelle splitten, so verwendet man dazu z.B. die Kommandofolge **r**<CTRL>, einen eigenen Befehl dafür gibt es nicht.

Löschen folgender Textstelle:

x	Zeichen an Cursorposition
*n***x**	*n* Zeichen, beginnend mit dem Zeichen an Cursorposition
X	Zeichen links von Cursorposition
*n***X**	*n* Zeichen links von Cursorposition
dw	Rest des Wortes ab Cursorposition
db	Anfang des Wortes bis Cursorposition
dd	aktuelle Zeile
*n***dd**	*n* Zeilen, beginnend mit der aktuellen
D	Rest der Zeile ab Cursorposition

J	Zeilenende, (aktuelle Zeile mit nachfolgender verbinden)
*n***J**	*n* Zeilenendemarken, (*n* Zeilen verbinden)
r *z*	Zeichen an Cursorposition, durch *z* ersetzen
*n***r** *z*	*n* Zeichen an Cursorposition, durch *z* ersetzen

4.4 Editoroperationen wiederholen und rückgängig machen

Will man die vorangegangene Editoroperation nochmals ausführen, braucht man dazu nicht das ganze Kommando wieder einzutippen;durch Betätigen der Taste . (Punkt), das letzte Kommando, das im Kommandomodus gegeben wurde (nicht im ex-Modus).

Umgekehrt ist es manchmal wünschenswert, ein zuvor ausgeführtes Kommando wieder rückgängig zu machen (insbesondere wenn man aus Versehen etwas zu viel gelöscht hat). Dazu dient das Kommando **u** (**u**ndo), das die letzte Editoroperation zurücknimmt. **u** hat leider keinen kumulativen Effekt, d.h. ein zweimaliges **u** hat nicht etwa die Wirkung, daß auch die vorletzte Operation rückgängig gemacht wird, vielmehr wird die vorangegangene **u**-Operation rückgängig gemacht: zweimal **u** bewirkt also überhaupt nichts.

Die **u**-Operation macht immer nur eine (die letzte) Operation rückgängig. Es gibt noch ein weiteres Kommando, das alle Editoroperationen, die in der aktuellen Zeile vorgenommen wurden, wieder zurücknimmt, nämlich **U**.

.	Wiederholen des letzten Kommandos (im Kommandomodus)
u	Zurücknehmen der letzten Editoroperation
U	Zurücknehmen aller Editoroperation für die aktuelle Zeile

5 vi: Kopieren und Verschieben

5.1 Die Kopierpuffer

Das Bewegen von Text an eine andere Stelle in der Datei spielt sich beim vi grundsätzlich über die eingebauten Kopierpuffer ab. Beim vi gibt es 27 davon, und zwar einen Standardpuffer ohne Namen und 26 benannte Puffer, die mit den Buchstaben a bis z benannt werden. Darüber hinaus gibt es neun numerierte Puffer, in denen gelöschte Objekte aufbewahrt werden. In Puffer 1 befindet sich das zuletzt gelöschte Objekt, in Puffer 2 das vorletzte, in Puffer 9 das neuntletzte gelöschte Objekt.

5.2 Befehle zum Kopieren und Verschieben

Textstellen, die kopiert oder verschoben werden sollen, werden zunächst mit dem Kommando **d** oder **y** in einen Kopierpuffer transferiert und anschließend an der gewünschten Stelle wieder in den Text eingefügt. Wird bei diesen Kommandos kein Puffername

angegeben, gelangt der Text in den Standardpuffer, wobei dessen alter Inhalt überschrieben wird. Wird dem Kommando hingegen ein Puffername in der Form "*Puffername* (das Anführungszeichen muß wirklich angegeben werden!) vorangestellt (nach einem evtl. angegebenen Zählfaktor), so gelangt der Text in den entsprechenden benannten Puffer. Wenn der Puffername als Kleinbuchstabe angegeben wird, geht der vorherige Pufferinhalt verloren, wenn der Name als Großbuchstabe geschrieben wird, wird der neue Text an den alten Pufferinhalt angehängt.

Das Kommando **d** löscht den gewählten Textbereich, kopiert ihn in den Standardpuffer oder in den angegebenen benannten Puffer und zusätzlich in den Puffer 1. Der alte Inhalt von Puffer 1 wird in Puffer 2 verschoben (usw). Das Kommando **y** (yank) kopiert den gewählten Textbereich in den Kopierpuffer. Wie beim Kommando **d** wird der Textbereich hinter **y** angegeben (siehe Abschnitt 8.4). Durch eine Wiederholung des Kommandozeichens, also durch **yy** (äquivalent dazu ist **Y**) wird eine ganze Zeile transferiert. Ein Wiederholungsfaktor beim Kommando **y** bezieht sich auf den gewählten Textbereich, so kann man mit `5yy` die aktuelle Zeile und die vier folgenden in den Kopierpuffer schreiben.

Den Pufferinhalt kann man mit **p** (**p**lace buffer) und **P** wieder zurückholen; ersteres fügt den Pufferinhalt hinter dem Cursor ein, letzteres vor dem Cursor. Die Kommandos arbeiten allerdings so nur, wenn nicht ganze Zeilen im Puffer stehen. Ist dies jedoch gegeben, so arbeiten die beiden Kommandos wie folgt: **p** fügt den Pufferinhalt unterhalb der aktuellen Zeile ein (egal ob der Cursor am Zeilenanfang steht oder nicht), entsprechend macht **P** dies oberhalb der aktuellen Zeile. Eine Zählung ist in jedem Fall zugelassen und dient wieder als Wiederholungsfaktor.

Für beide Kommandos gilt: Wird kein Puffername angegeben, so bezieht sich das Kommando auf den Standardpuffer, soll es sich auf einen anderen Puffer beziehen, muß der Puffername in der Form "*Puffername* dem Kommando vorangestellt werden. Hier macht es dann keinen Unterschied mehr, ob der Name als Groß- oder Kleinbuchstabe angegeben wird.

Kopieren:

Y	aktuelle Zeile in Arbeitspuffer kopieren
y w	ab Cursorposition bis Wortende in Arbeitspuffer kopieren
y b	Wortanfang bis Cursorposition in Arbeitspuffer kopieren
"*a***Y**	aktuelle Zeile in Puffer *a* kopieren
"*a***p**	Inhalt des Puffer *a* nach aktueller Zeile einfügen
"*a***P**	Inhalt des Puffer *a* vor aktueller Zeile einfügen
p	Inhalt des Arbeitspuffers nach Cursorposition einfügen
P	Inhalt des Arbeitspuffers vor Cursorposition einfügen

Verschieben:

D	aktuelle Zeile löschen, Kopie in Arbeitspuffer
d w	von Cursor bis Wortende löschen, Kopie in Arbeitspuffer

d b	von Wortanfang bis Cursor löschen, Kopie in Arbeitspuffer
"*a*D	aktuelle Zeile löschen, Kopie in Puffer *a* kopieren
"*a*p	Inhalt des Puffer *a* nach aktueller Zeile einfügen
"*a*P	Inhalt des Puffer *a* vor aktueller Zeile einfügen
p	Inhalt des Arbeitspuffers nach Cursorposition einfügen
P	Inhalt des Arbeitspuffers vor Cursorposition einfügen
" 1 p	Inhalt des Puffer 1 nach aktueller Zeile einfügen
" 2 P	Inhalt des Puffer 2 vor aktueller Zeile einfügen

6 vi: Weitere Kommandos

6.1 Text sichern und/oder Editor verlassen

Nicht nur beim Verlassen des Editors, sondern auch während längerer Editorsitzungen ist es durchaus empfehlenswert, den Text zu sichern (obwohl unter UNIX auch bei einem Systemabsturz die Editierarbeit meist nicht verloren geht). Mit dem Kommando **:w** sichern Sie den editierten Text in die aktuelle Datei, d.h. in diejenige Datei, die beim vi-Aufruf angegeben wurde. Es besteht jedoch auch die Möglichkeit, den Text in eine andere Datei zu sichern. Hierzu müssen Sie hinter **:w** den gewünschten Dateinamen angeben. Sie können jedoch so nicht in eine Datei sichern, die bereits existiert, bei einem Versuch, dies zu tun, erhalten Sie eine entsprechende Fehlermeldung. Wollen Sie eine andere als die aktuelle Datei wirklich überschreiben, benutzen Sie das Kommando **:w!** *dateiname*. Wenn Sie nach dem Sichern den Editor verlassen wollen, verwenden Sie **:wq** (evtl. ebenfalls gefolgt von einem Dateinamen). Der Unterschied zu **Z Z** ist, daß **:wq** die Datei vor dem Verlassen des Editors in jedem Fall sichert, während **Z Z** die Datei nur zurückschreibt, wenn Sie verändert wurde.

Zum Verlassen des Editors ohne Sichern tippen Sie **:q** (gefolgt von der RETURN-Taste, da es sich um ein ex-Kommando handelt). Wenn Sie die Datei editiert haben, gibt dies allerdings eine Fehlermeldung, dann müssen Sie **:q!** verwenden.

Sollte während einer Editorsitzung das System abstürzen, ist die editierte Datei im allgemeinen nicht verloren gegangen. Das System schickt Ihnen dann eine Mail-Message, in der Ihnen mitgeteilt wird, wie Sie die Editorsitzung wieder aufnehmen können (nämlich durch Angabe der Option -r beim vi-Kommando, siehe Kapitel 10.1.).

:w	Sichern in bei Aufruf von vi angegebene Datei
:w *datei*	Sichern in neue Datei *datei*
:w! *datei*	Sichern in Datei *datei*, diese wird dabei überschrieben
:wq	Verlassen von vi mit Sichern
Z Z	Verlassen von vi, Änderungen in der Datei werden gesichert
:q	Verlassen von vi ohne Sichern (falls nicht editiert wurde)
:q!	Verlassen von vi ohne Sichern (falls editiert wurde)

6.2 Textstellen markieren

Wenn Sie immer abwechselnd an verschiedenen Stellen editieren, ist es höchst umständlich, immer mit irgendwelchen Cursorsteuerungskommandos die richtige Textstelle wiederzufinden. Aber es gibt ja auch die Möglichkeit, bestimmte Textzeilen zu markieren und diese dann per Kommando anzuspringen.

Benutzen Sie das (zusammengesetzte) Kommando **m***kleinbuchstabe*, (**m**ark) um die aktuelle Textzeile mit einem Buchstaben zu markieren. Zum Wiederfinden dieser Zeile gibt es dann zwei weitere Kommandos, nämlich ` und '. Dahinter müssen Sie jeweils den Markierungsbuchstaben der gewünschten Zeile angeben. ' bewegt den Cursor auf das erste nicht-Blank-Zeichen der entsprechenden Zeile, während ` den Cursor immer genau auf die markierte Position setzt.

m*k*	markieren einer Position (k: kleiner Buchstabe),
`*k*	Cursor auf die mit *k* markierte Position stellen
'*k*	Cursor auf Zeilenanfang der mit *k* markierten Zeile stellen

6.3 Bildschirm restaurieren

Während einer Editorsitzung kann es oftmals passieren, daß Ihnen das System oder ein anderer Benutzer (mit dem write-Kommando) etwas auf den Bildschirm schreibt, das mit Ihrer editierten Datei gar nichts zu tun hat. Nach Kenntnisnahme dieser Meldung möchte man gerne den normalen Editorbildschirm wiederherstellen. Das Kommando dafür ist `CTRL-L`, das natürlich nur im Kommandomodus die gewünschte Wirkung hat (bei manchen Terminaltypen auch `CTRL-R`).

`CTRL-L`	Wiederherstellen des Editorbildschirms

7 vi: Konfigurieren

Wenn Sie die vorangegangenen Abschnitte genau durchgelesen haben, sollten Sie nun in der Lage sein, den vi einigermaßen geschickt bedienen zu können. Die folgenden Informationen sollen Sie dazu befähigen, aus dem vi noch mehr herauszuholen. Sie sind nicht für die grundlegende Bedienung erforderlich.

7.1 Editorvariablen

Der vi kennt verschiedene Variablen (in der englischen Anleitung Optionen genannt, was aber zur Verwechslung mit den vi-Kommando-Optionen führen könnte), die das Verhalten des vi während einer Editorsitzung beeinflussen. An dieser Stelle werden nur die Kommandos zum Setzen und Löschen dieser Variablen behandelt. Die Variablen selbst sind im Abschnitt 7.2 aufgeführt.

Der vi unterscheidet zwei Arten von Variablen: Bei den einen kommt es nur darauf an, ob sie gesetzt sind oder nicht (sog. Toggles), die anderen haben einen bestimmten numerischen oder alphanumerischen Wert (Wertevariablen). Das Setzen eines Toggles geschieht mit dem Kommando

:set *variable*

Das Löschen mit dem gleichen Befehl, aber mit no vor dem Variablennamen, also mit

:set no*variable*

Wertevariablen werden mit dem Befehl

:set *variable=wert*

verändert. Es darf kein Blank zwischen Variablennamen, Gleichheitszeichen und Wert stehen!
Umgekehrt können die Werte der Variablen mit

:set

abgefragt werden. Obiges Kommando liefert allerdings nur die Werte der Variablen, die von den Standardbelegungen abweichen. Die Werte aller Variablen liefert

:set all

Es gibt auch noch eine dritte Variation hiervon:

:set *variable?*

zeigt nur den Wert der angegebenen Variablen.

7.1.1 Automatischer Zeilenumbruch

Bei der Eingabe von Texten, die nachträglich mit einem Textverarbeitungsprogramm wie nroff oder troff formatiert werden sollen, bei denen es also nicht auf das genaue Aussehen der Textzeilen ankommt, ist es vorteilhaft, den Editor vi den Zeilenumbruch selbst vornehmen zu lassen. Darunter ist zu verstehen, daß der vi während der Texteingabe bei Überschreiten einer gewissen Spalte das aktuelle Wort automatisch in die nächste Zeile plaziert, ohne daß der Benutzer `RETURN` drücken muß. Damit umgeht man die ewigen Korrekturen, wenn mal ein Wort wider Erwarten doch nicht mehr in eine Zeile paßt.

Schalten Sie den automatischen Zeilenumbruch ein, indem Sie der Variable `wrapmargin` einen numerischen Wert ungleich 0 geben. Dieser gibt dann an, ab welcher Entfernung (in Spalten) vom rechten Bildschirmrand ein Wort in die nächste Zeile gerückt werden soll. Wenn Sie z.B. ein 80-spaltiges Terminal haben, veranlaßt **:set wrapmargin=10,** daß der eingegebene Text immer mindestens 10 Spalten vom rechten Bildschirmrand entfernt ist, also niemals über Spalte 70 hinausgeht.

Der Standardwert für wrapmargin ist 0, d.h. daß diese Option abgeschaltet ist. Sie können also die Option von Hand wieder ausschalten, indem Sie die Variable auf 0 zurücksetzen.

7.1.2 Automatisches Einrücken beim Editieren von Programmen

Wenn Sie mit dem vi hauptsächlich Programme eintippen, werden Sie sicher die jetzt aufgeführten Variablen bald zu schätzen lernen. Das Problem beim Eingeben von Programmen ist ja bekanntlich die Einrückung von geschachtelten Prozeduren oder

Funktionen. Der vi setzt normalerweise beim Beginnen einer neuen Zeile im Einfügemodus den Cursor an den Zeilenanfang. Dies ist aber nicht immer erwünscht, vielmehr möchte man, daß der Cursor so weit eingerückt wird, wie es die vorherige Zeile auch war. Außerdem wäre es nützlich, Zeilen nachträglich einzurücken, ohne jede Zeile im Einfügemodus der Reihe nach mit Blanks zu ergänzen.

Die automatische Einrückung (jede Zeile wird beim Eintippen so weit eingerückt wie die vorherige Zeile) wird durch die Variable `autoindent` kontrolliert. Mit

```
:set autoindent
```

schalten Sie durch die Variable ein.

Wenn Sie dann im Einfügemodus Programmzeilen eintippen, werden Sie allerdings eine eigenartige Feststellung machen: Es ist (zunächst) nicht möglich, eine neue Zeile weniger weit einzurücken als die vorherige Zeile, denn mit der `DEL`-Taste können Sie nicht vor die Einrückposition zurückgehen. Abhilfe schafft hier `CTRL-D` (wirklich im Einfügemodus!), damit wird nämlich der Cursor um eine gewisse Spaltenzahl vor die aktuelle Position gesetzt.

Diese gewisse Spaltenzahl bestimmt sich aus dem Wert der Variablen `shiftwidth`, der standardmäßig gleich 8 ist. Dieser Wert wird auch für die folgenden Kommandos benötigt, die dazu dienen, Zeilen nachträglich ein- oder auszurücken.

- `>>` rückt die aktuelle Zeile um `shiftwidth` Spalten ein,
- `<<` rückt die aktuelle Zeile um `shiftwidth` Spalten aus.

Mit einer Zählung kann erreicht werden, daß sich diese beiden Kommandos auch auf mehr als eine Zeile beziehen. Will man gleich den ganzen Rest des Textes ab der aktuellen Cursorzeile ein- oder ausrücken, verwendet man die Kommandos >L bzw. <L.

7.1.3 Groß/Kleinschreibung bei der Suche nach Text

Standardmäßig wird bei Such- und Ersetzungsbefehlen im Suchwort Groß- und Kleinschreibung unterschieden. So wird zum Beispiel bei der Suche nach dem Begriff "Unix" der Begriff "UNIX" nicht gefunden. Dieses Verhalten wird durch die Editorvariable ignorecase beeinflußt. Mit

```
:set ignorecase
```

schaltet man die Unterscheidung zwischen Groß- und Kleinschreibung ab, mit

```
:set noignorecase
```

wieder an.

7.1.4 Suchen mit regulären Ausdrücken

Mit Hilfe von Sonderzeichen (sog. Metasymbolen) kann die Suche nach einem Suchbegriff eingeschränkt werden, z.B. auf das Vorkommen des Suchbegriffs am Zeilenanfang oder Zeilenende. Darüber hinaus kann die Suche erweitert werden, so daß nicht nur nach einem einzigen Suchbegriff, sondern nach mehreren gleichzeitig gesucht werden kann. Je nach Zustand der Editorvariablen `magic` stehen unterschiedlich viele Metasymbole zur Verfügung. Bei abgeschalteter Editorvariablen magic

(**:set nomagic**) gibt es die beiden Metasymbole **^** und **$**; sie haben folgende Bedeutung:

^*suchbegriff*	***suchbegriff*** steht am Zeilenanfang
***suchbegriff*$**	***suchbegriff*** steht am Zeilenende
^*suchbegriff***$**	Zeile enthält nur das Wort ***suchbegriff***

Im Magic-Modus, d.h. bei gesetzter Editorvariablen `magic` (Standardeinstellung), haben darüber hinaus die Zeichen **[ ] \ . *** Sonderbedeutungen, die im folgenden erläutert werden. Ist die Editorvariable `magic` nicht gesetzt, muß diesen Zeichen der Backslash ****vorangestellt werden, wenn sie in der Sonderbedeutung verwendet werden sollen.

[*Zeichenfolge***]**	ein Element der ***Zeichenfolge***
[^*Zeichenfolge***]**	kein Element der ***Zeichenfolge***
\<*suchbegriff*	***suchbegriff*** steht am Wortanfang
***suchbegriff*\>**	***suchbegriff*** steht am Wortende
*	beliebige (auch 0-malige) Wiederholung des vorangegangenen Zeichens
.	ein beliebiges Zeichen (außer NEWLINE)
v.r	ver, vor, var, v4r, ...
.*	beliebige Zeichenfolge (Länge ≥0)

Beispiel:

[Aa]	a oder A
[a-z]	kleiner Buchstabe
[0-9]	Ziffer
^END$	Zeile enthält nur das Wort END
^[Ee][Nn][Dd]$	Zeile enthält END oder end oder End oder ...
[^0-9]	keine Ziffer
[^0a]	keine 0 und kein a
ma*t	mt, mat, maat, maaat, usw.
maa*t	mat, maat, maaat, usw.
a.*t	at, ast, ararat, ...

Soll ein Metasymbol nicht in der Sonderbedeutung, sondern als gewöhnliches Zeichen verwendet werden, muß man diesem den Backslash **** voranstellen. Damit bekommt natürlich auch der Backslash eine Sonderbedeutung. Um diese wiederum auszuschalten, tippen Sie den Backslash zweimal.

Beispiel:

Sie suchen nach einer Zahl mit mindestens zwei Stellen hinter dem Dezimalpunkt:

```
/\.[0-9][0-9]
```

Wenn der Suchausdruck ein Leerzeichen enthält, muß er mit / abgeschlossen werden:

```
/2\.[0-9][0-9] DM/
```

7.1.5 vi mit EXINIT dauerhaft konfigurieren

Wenn Sie irgendwelche Editorvariablen bei jeder Editorsitzung geändert haben wollen, gibt es eine Alternative zum Eintippen diverser :set-Kommandos am Anfang jeder Sitzung. Definieren Sie sich die Environment-Variable EXINIT, in der Sie mehrere, am Anfang der Editorsitzung auszuführende ex-Kommandos(!) angeben können. Zwei Kommandos müssen Sie hier jeweils durch einen senkrechten Strich trennen. Wenn also z.B. immer die Variable autoindent gesetzt und die Variable `wrapmargin` den Wert 10 haben soll, können Sie die UNIX-Variable EXINIT setzen anstatt nach jedem vi-Aufruf die Befehle **`:set autoindent`** sowie **`:set wrapmargin=10`** einzugeben.

```
EXINIT='set autoindent | set wrapmargin=10'
```

oder kürzer (in einem set-Befehl):

```
EXINIT='set autoindent wrapmargin=10'
```

Wenn Sie diesen Befehl in Ihre Datei .profile schreiben, wird er bei jedem Login ausgeführt.

Hinweis: Das RRZN definiert im Standard-Benutzerprofile die Variable EXINIT so, daß der Editor sich im Rahmen seiner Möglichkeiten benutzerfreundlich verhält. Bei Bedarf können Sie diese Definition selbstverständlich ergänzen oder durch eine eigene ersetzen.

7.2 Übersicht über die Editorvariablen

Dieser Abschnitt enthält eine Zusammenstellung aller Editorvariablen, mit denen Sie das Verhalten des vi während der aktuellen Editorsitzung beeinflussen können.

`autoindent` Abk.: `ai` Standardwert: `noai` Typ: Toggle
Ist diese Variable gesetzt, werden alle neu eingegebenen Zeilen auf die gleiche Spalte wie die vorhergehende Zeile eingerückt.

`autoprint` Abk.: `ap` Standardwert: `ap` Typ: Toggle
Die aktuelle Zeile wird immer dann auf dem Bildschirm ausgegeben, wenn ein ex-Kommando gegeben wurde, das den Text modifizierte. Im vi-Modus ziemlich uninteressant.

`autowrite` Abk.: `aw` Standardwert: `noaw` Typ: Toggle
Automatisches Sichern des Textes vor Ausführung von bestimmten Kommandos, die Dateiinhalte ändern (könnten) oder andererseits mit UNIX direkt in Verbindung treten.

`beautify` Abk.: `bf` Standardwert: `nobf` Typ: Toggle
Ist diese Variable gesetzt, werden im Eingabemodus alle Kontrollzeichen außer <tab>, <nl> und <ff> ignoriert.

`directory` Abk.: `dir` RRZN-Einstellung: `dir=/var/tmp` Typ: String
Inhaltsverzeichnis, in dem vi seine temporäre Textdatei ablegt.

`edcompatible` Abk.: `ed` Standardwert: `noed` Typ: Toggle
substitute-Kommando ed-ähnlich.

`errorbells` Abk.: `eb` Standardwert: `noeb` Typ: Toggle
vi-Fehlermeldungen werden von einem Piepsen des Terminals begleitet.

`exrc` Standardwert: `noexrc` Typ: Toggle
Ist diese Variable gesetzt, wird die .exrc-Datei im Arbeitsverzeichnis ausgeführt, falls das Arbeitsverzeichnis nicht das HOME-Verzeichnis ist.

`flash` Abk.: `fl` Standardwert: `fl` Typ: Toggle
vi-Fehlermeldungen werden bei geeigneter Terminaldefinition anstelle von einem Piepsen von einem Flackern des Terminals begleitet.

`hardtabs` Abk.: `ht` Standardwert: `ht=8` Typ: Numerisch
Enthält die Spaltendifferenz zwischen Hardware-Tabulatoren des Terminals bzw. die Differenz zwischen Software-Tabulatoren, die vom UNIX-System in Leerzeichen umgesetzt werden.

`ignorecase` Abk.: `ic` Standardwert: noic Typ: Toggle
Bei der Verwendung von regulären Ausdrücken werden alle Großbuchstaben in Kleinbuchstaben umgewandelt.

`lisp` Standardwert: `nolisp` Typ: Toggle
Automatisches Einrücken für LISP-Programme. Die Kommandos (,), [[und]] werden entsprechend verändert, um s-Ausdrücke und Funktionen zu bearbeiten.

`list` Standardwert: `nolist` Typ: Toggle
Die Zeichen <tab> und <nl> werden explizit als Steuerzeichen angezeigt.

`magic` Standardwert: `magic` Typ: Toggle
Beim Suchen nach Zeichenketten werden außer `^`, `$`, `\` auch die Zeichen `[`, `]`, `.`, `*` als Metasymbole interpretiert; siehe Abschnitt 7.1.4

`mesg` Abk.: `mesg` Standardwert: `mesg` Typ: Toggle
Erlaubt die Ausgabe von Meldungen auf dem Bildschirm.

`modelines` Abk.: `ml` Standardwert: `noml` Typ: Toggle
Ermöglicht, daß ex-Kommandos am Anfang oder Ende einer Datei beim Einlesen der Datei in den Editor zunächst ausgeführt werden.

`novice` Standardwert: `nonovice` Typ: Toggle
Ist diese Variable gesetzt, wird edit anstatt ex genutzt.

`number` Abk.: `nu` RRZN-Einstellung: `nu` Typ: Toggle
Jede Zeile wird mit ihrer Zeilennummer angezeigt.

`optimize` Abk.: `opt` RRZN-Einstellung: `noopt` Typ: Toggle
Optimiert die Ausgabe über "dumme" Bildschirme.

`paragraphs` Abk.: `para` Standardwert: `IPLPPPQPPLIpplpipnpbp` Typ: String
Jedes Paar von Zeichen gibt ein nroff-Makro an, das vom vi als Anfang eines Absatzes interpretiert werden soll. Um Makros anzugeben, die nur aus einem Zeichen bestehen, muß in Leerzeichen für das zweite Zeichen angegeben werden.

`prompt` Standardwert: `prompt` Typ: Toggle
Wird durch Q vom vi-Modus in den ex-Modus gewechselt, wird der ex-Prompt : angezeigt.

`readonly` Standardwert: `readonly` Typ: Toggle
Datei kann nicht verändert werden.

`redraw` Abk.: `re` RRZN-Einstellung: `re` Typ: Toggle
Den Bildschirm nach jedem eingesetzten Zeichen neu schreiben.

`remap` Standardwert: `remap` Typ: Toggle
Wenn ein Makro eingesetzt wurde, wird dadurch möglicherweise ein weiteres Makro angesprochen; deshalb wird der Text wiederholt nach weiteren Makros durchsucht.

`report` Standardwert: `report=5` Typ: Numerisch
Gibt die Zahl der modifizierten Zeilen an, ab der eine entsprechende Meldung in die Infozeile geschrieben werden soll.

`scroll` Standardwert: scroll=halbe Bildschirmhöhe Typ: Numerisch
Gibt die Zahl der Zeilen an, um die der Bildschirm bei Verwendung der `CTRL-D` und `CTRL-U` gescrollt werden soll.

`sections` RRZN-Einstellung: `NHSHHHUuhsh+c` Typ: String
Jedes Paar von Zeichen gibt ein nroff-Makro an, das vom vi als Anfang eines Kapitels aufgefaßt werden soll. Makros, die nur aus einem Zeichen bestehen, müssen duch ein Leerzeichen auf zwei Zeichen aufgefüllt werden.

`shell` Abk.: `sh` Standardwert: `sh=$SHELL` Typ: String
UNIX-Kommandoname der Shell, die bei Shell-`Esc`ape-Kommandos aufgerufen werden soll.

`shiftwidth` Abk.: `sw` Standardwert: `sw=8` Typ: Numerisch
Gibt die Zahl der Leerzeichen an, um die bei `CTRL-D` und `CTRL-T` im Einfügemodus der Cursor bewegt wird bzw. um wieviele Spalten die Kommandos < und > den Text verschieben.

`showmatch` Abk.: `sm` RRZN-Einstellung: `sm` Typ: Toggle
Wird eine) oder } eingegeben, so zeigt der vi für einen Augenblick die zugehörige öffnende Klammer an, indem er den Cursor darauf setzt.

`showmode` RRZN-Einstellung: `showmode` Typ: Toggle
Wechsel in Eingabemodus wird angezeigt.

`slowopen` Abk.: `slow` Typ: Toggle
Verhindert Anzeige während der Eingabe, für "dumme" Terminals gedacht. Standardwert ist abhängig vom Terminal.

`tabstop` Abk.: `ts` Standardwert: `ts=8` Typ: Numerisch
<tab>-Zeichen werden auf Spalten ausgeweitet, die ein Vielfaches dieses Wertes darstellen.

`taglength` Abk.: `tl` Standardwert: `tl`=0 Typ: Numerisch
Maximale Anzahl von signifikanten Zeichen in einem tag; tl=0 bedeutet, daß alle Zeichen signifikant sind.

`tags` Standardwert: `/usr/lib/tags` Typ: String
Pfadname der Dateien, die tags enthalten (vgl. UNIX-Kommando `ctags`).

`term` Standardwert: `$TERM` (aus Environment) Typ: String

Definiert den Terminaltyp für die vi-Full-Screen-Verwaltung. Diese Variable sollte normalerweise nicht geändert werden, außer sie ist aus irgendwelchen Gründen falsch gesetzt. Um diese Variable zu ändern, müssen Sie (mit `Q`) in den ex-Modus gehen, dort das entsprechende set-Kommando eingeben und dann vi tippen, um wieder in den vi-Modus zu gelangen.

`terse` RRZN-Einstellung: `noterse` Typ: Toggle
vi-Fehlermeldungen werden nicht in gekürzter Form wiedergegeben.

`timeout` Standardwert: `timeout` Typ: Toggle
Ist diese Variable gesetzt, so muß der Benutzer in einer vom System vorgegebenen Zeit (ca. 1 Sekunde) eine Abkürzung, die durch das `map`-Kommando definiert wurde, eingeben, damit diese ausgewertet werden kann.

`ttytype` siehe term

`warn` Standardwert: `warn` Typ: Toggle
Veranlaßt die Ausgabe von Warnungen, wenn Shell `Esc`apes vorgenommen werden, ohne die letzten Textänderungen gesichert zu haben.

`window` Standardwert: abh. v. Ubertragungsgeschwindigkeit Typ: Numerisch
Legt die vi-Bildschirmhöhe in Zeilen fest.

`wrapscan` Abk.: `ws` Standardwert: `ws` Typ: Toggle
Ist diese Variable gesetzt, wird bei Suchoperationen, die ans Dateiende gelangen, vom Dateianfang aus weitergesucht (und umgekehrt).

`wrapmargin` Abk.: `wm` Standardwert: `wm=0` Typ: Numerisch
Der vi nimmt bei der Texteingabe automatisch einen Zeilenumbruch vor (d. h. fügt ein <nl> ein), wenn ein Leerzeichen eingegeben wird und sich der Cursor höchstens wm Spalten vom rechten Rand entfernt befindet. Hat wm den Wert 0, ist die Option ausgeschaltet.

`writeany` Abk.: `wa` Standardwert: `nowa` Typ: Toggle
Normalerweise nimmt der vi eine Reihe von Sicherheitsprüfungen vor, bevor er einen Text in eine Datei schreibt (z. B. ob die Datei bereits existiert). Durch das Setzen dieser Variable können Sie diese Prüfungen abschalten.

8 vi für Fortgeschrittene

8.1 Dateimanipulationen

Die folgende Übersicht zeigt die vi-Kommandos, die der Interaktion des vi mit physikalischen Dateien dienen. Es handelt sich dabei ausnahmslos um ex-Kommandos, daher muß zunächst mit : in den ex- Modus gewechselt werden.

:w	Text sichern auf aktuelle Datei
:wq	Text sichern und vi verlassen
:x	falls Änderungen, Text sichern, vi verlassen (wie **ZZ**)
:e *name*	Datei *name*editieren

:e!	Datei nochmals editieren, vorgenommene Änderungen sind verloren
:e \#	Alternative Datei editieren (siehe unten)
:w *name*	Text sichern auf Datei *name*
:w! *name*	Text sichern auf Datei *name*, die in jedem Fall überschrieben wird
:*n1*,*n2***w** *name*	Zeilen *n1* bis *n2* auf Datei *name* sichern
:r *name*	Datei *name* hinter Cursorzeile einfügen
:r !*unix-kommmando*	Standardausgabe des angegebenen *unix-kommmandos* hinter der Cursorzeile einfügen
:n	Nächste Datei (in vi-Argumentliste) editieren
:n!	wie **:n**, Änderungen an der aktuellen Datei gehen verloren

Das Zeichen # als Dateiname bezieht sich auf die Alternativdatei, das ist diejenige Datei, die zuletzt angesprochen wurde, und deren Name ungleich dem aktuellen Dateinamen war.

8.2 Makros und Floskeln

Insbesondere bei der häufigeren Wiederholung längerer Kommandosequenzen wäre es wünschenswert, diese abkürzen zu können. vi bietet diese Möglichkeit, und zwar nicht nur für die Abkürzung einer Kommandosequenz, sondern gleich mehrerer durch die Definition von Makros. Es gibt zwei verschiedene Arten von Makros: Einmal solche, die aus mehreren Kommandos bestehen und mit einer Tastenkombination der Form *@buchstabe* abgekürzt werden können. Sie definieren sich ein solches Makro einfach, indem Sie die Kommandos, die Sie zu einem Makro zusammenfassen wollen, in einen der (benannten) Kopierpuffer übernehmen.

Der Buchstabe, mit dem der Kopierpuffer bezeichnet ist, ist genau derjenige, den Sie auch beim Makroaufruf hinter dem @ angeben müssen. Achten Sie aber darauf, daß Sie den Inhalt eines solchen als Makro verwendeten Kopierpuffers nicht versehentlich durch Kopieroperationen mit normalem Text überschreiben, die Ergebnisse bei einem Makroaufruf eines überschriebenen Puffers sind nicht vorhersehbar und enden meist mit einem Chaos im editierten Text, im besten Fall mit einer Fehlermeldung.

Um eine beliebige Kommandofolge abzukürzen (siehe Kapitel 11), brauchen Sie keine Kopierpuffer zu verwenden; es gibt ein eigenes Kommando zur Definition solcher Makros. Tippen Sie den Befehl

:map *abkürzung kommandofolge*

Danach können Sie statt *kommandofolge* immer die Tastenfolge *abkürzung* verwenden. *abkürzung* soll ein kurzer String sein (d.h. ein oder zwei Zeichen lang), wobei möglichst nicht ein durch Kommandos anderweitig belegter Buchstabe oder ein Sonderzeichen benutzt wird. Normalerweise ist die Editorvariable timeout gesetzt, so daß man innerhalb einer vorgegebenen Zeitspanne (ca. 1 Sekunde) die komplette *abkürzung* eingetippt haben muß, wenn diese als solche erkannt werden soll. Abkürzungen mit mehr als drei Zeichen sind daher kaum benutzbar. *kommandofolge* kann ein bis zu 100 Zeichen lange Folge von vi-Kommandos sein.

Hinweis: Manche Sonderzeichen, wie <ESC> oder <CTRL>-Zeichen müssen maskiert werden, damit sie eingegeben werden können. Dies geschieht durch das Voranstellen eine <CTRL-V>. Will man z.B. in einen Text oder in einer *abkürzung* ein <CTRL-H> einfügen, so muß man die Kombination <CTRL-VH> eingeben (d.h. während man die CONTROL-Taste festhält, werden **beide** Tasten nacheinander gedrückt). Dargestellt werden diese Sonderzeichen dann durch das Zeichen ^, also ^H für <CTRL-H> und ^[für <ESC>.

Nicht maskierbar ist die RETURN-Taste. Ersatzweise stellt man diese dann durch ^M (also <CTRL-M>) dar.

Ein mit map definiertes Makro kann mit

:unmap *abkürzung*

wieder aufgehoben werden. Danach bewirkt *abkürzung* nicht mehr die Ausführung des zuvor definierten Makros. Übrigens kann ein Makro neu definiert werden, ohne zuvor das alte Makro mit der gleichen Abkürzung zu löschen; die neue Definition überschreibt dann die alte.

Es gibt noch eine zweite Form des map-Kommandos, und zwar

:map! *abkürzung string*

Dies bewirkt, daß *abkürzung* im Einfüge- und im ex-Modus stets durch *string* ersetzt wird. *abkürzung* kann hier bis zu 10 Zeichen lang sein. Dieses Kommando ist mit äußerster Vorsicht anzuwenden, da man sich hier sehr leicht in eine Situation manövrieren kann, aus der man nicht mehr herauskommt.

Wenn Sie sich die Makros ansehen wollen, die Sie bereits definiert haben, können Sie dies mit **:map** bzw. **:map!** tun.

Mit Floskeln lassen sich Abkürzungen definieren, die im Einfügemodus wirksam sind. Sie werden definiert mit

:abbreviate *abkürzung string*

Die Floskelersetzung arbeitet ähnlich wie die Makroersetzung, allerdings mit dem wesentlichen Unterschied, daß *abkürzung* nur als eigenständiges Wort (umgeben von Leerzeichen oder Sonderzeichen) durch *string* ersetzt wird, wenn . Ist *abkürzung* nur Teil eines Wortes, wird (anders als bei Makros) nichts ersetzt. Für *abkürzung* gibt es keine Einschränkung in der Länge. Sie werden eine Floskeldefinition wieder los mit

:unabbreviate *abkürzung*

und mit

:abbreviate

können Sie sich die definierten Abkürzungen listen lassen.

8.3 Kommandozählungen

Vor den meisten vi-Kommandos können Sie eine Zahl, die sogenannte Kommandozählung, angeben. Diese hat in fast allen Fällen einfach die Funktion, dem vi mitzuteilen, wie oft das betreffende Kommando ausgeführt werden soll. Die Kommandos, bei denen eine Zählung eine andere Bedeutung hat, werden im folgenden nochmals kurz behandelt.

Bei **:**, **/**, **?**, und **+** gibt die Zählung die Fenstergröße an, d.h. die Zahl der Zeilen, die nach Ausführung des Kommandos angezeigt werden sollen, wenn der Bildschirm neu erstellt werden muß. Durch nachfolgende Scrolloperationen werden dann aber nach und nach wieder so viele Zeilen angezeigt, wie am Terminal Platz haben. Die Festlegung einer kleineren Fenstergröße ist nur für langsame Terminals sinnvoll, um die Zeit, die zum Bildschirmaufbau benötigt wird, etwas zu verkürzen.

Bei CTRL-D und CTRL-U legt ein Zählfaktor die Anzahl der Zeilen fest, um die gescrollt werden soll. Diese Angabe gilt auch für alle folgenden Kommandos dieser Art, wenn kein Zählfaktor davor angegeben wird.

Bei den Kommandos **z** und **G** hat eine Kommandozählung hingegen die Bedeutung einer Zeilennummer, bei **|** die Bedeutung einer Spaltennummer. Alle anderen Kommandos, die Zählfaktoren zulassen, werden wie gesagt so oft ausgeführt, wie die Zählung angibt.

n **dw**	*n* Wörter löschen
n CTRL-D	
n CTRL-U	
*n***z** RETURN	Zeile *n* als erste Zeile des Bildschirms
n **G**	Zeile *n* wird aktuelle Zeile
n **|**	Cursor in Spalte *n*

8.4 vi-Wirkungsbereiche

Mehrere vi-Kommandos, die den Dateiinhalt oder einen vi-Puffer verändern (sog. „Wirkungs-Kommandos"), sind nach dem folgenden Muster aufgebaut:

n □ *Wirkungsbereich*

Der Multiplikationsfaktor *n* ist dabei optional. „□" steht für die Bezeichnung des Kommandos, bei der es sich um eines der folgenden Zeichen handeln kann:

□**c:** Ersetzen von Text.

□**d:** Löschen von Zeilen.

□**y:** Füllen eines vi-Puffers mit Text.

□**!:** Ersetzen von Text, wobei der zu ersetzende Text durch ein UNIX-Kommando gefiltert wird.

□**>**: Verschieben von Zeilen nach rechts.

□**<**: Verschieben von Zeilen nach links.

Für den *Wirkungsbereich* des Wirkungs-Kommandos gibt es zwei verschiedene Formate:

- Der Kommandoname wird verdoppelt (das gesamte Kommando ist dann also von der Form *n* □□). In diesem Fall bezieht sich das Kommando auf die gesamte aktuelle Zeile bzw. auf *n* ganze Zeilen ab einschließlich der aktuellen Zeile.
- Der Wirkungsbereich besteht aus einem vi-Kommando, mit dem der Cursor innerhalb der Datei bewegt wird (sog. „Bewegungskommandos"). Dieses Kommando kann sogar selbst wieder einen Multiplikator haben. Bei diesem Format erstreckt sich der geänderte Bereich (bzw. die Menge des zu puffernden Textes) im Prinzip von der aktuellen Cursorposition bis zu derjenigen Position, die der Cursor nach alleiniger Eingabe des Bewegungskommandos einnehmen würde. Der genaue Wirkungsbereich wird noch davon beeinflußt,
  - ob sich das Wirkungskommando generell auf ganze Zeilen auswirkt (im Falle von **<** und **>**),
  - ob sich das Bewegungskommando primär auf eine Zeile bezieht (z.□B. wie bei *n***G**) oder auf eine Position innerhalb einer Zeile (z.□B. bei *n***w** und *n***W**)
  - und ob das Bewegungskommando nach Vorwärts zum Dateiende hin gerichtet ist (wie z.□B. bei *n***w**□) oder nach Rückwärts zum Dateianfang hin (wie z.□B. bei *n***b**). Manche Bewegungskommandos können dabei in Abhängigkeit von der aktuellen Cursorposition nach Vorwärts *oder* Rückwärts gerichtet sein (wie z.□B. bei **%**).

Bezieht sich das Wirkungskommando auf ganze Zeilen oder das Bewegungskommando primär auf eine Zeile, so handelt es sich beim Wirkungsbereich um die aktuelle Zeile bis zu einschließlich derjenigen Zeile, die durch das Bewegungskommando erreicht werden würde. In den restlichen Fällen erstreckt sich der Wirkungsbereich bis zu einschließlich der End-Position des Bewegungskommandos; Anfangspunkt des Bereiches ist bei vorwärts gerichteten Bewegungskommandos die aktuelle Cursorposition bzw. bei rückwärts gerichteten Kommandos die aktuelle Position *vor* dem Cursor.

In den meisten Fällen wird eines der folgenden Bewegungskommandos verwendet:

*n***w**, *n***W**
*n***b**, *n***B**
*n***e**, *n***E**
*n***G**
'x
%

8.5 Korrekturen im Einfügemodus

Im Einfügemodus gibt es außer der DEL-Taste (die zum Löschen des zuletzt eingegebenen Zeichens dient) noch einige weitere Tasten mit Sonderfunktionen: Mit CTRL–U löschen Sie alles, was Sie auf der aktuellen Zeile getippt haben, mit CTRL–W das zuletzt eingegebene Wort. Das Drücken der RETURN-Taste bewirkt im Einfügemodus das Erzeugen einer neuen Zeile.

Schon bekannt ist die Funktion von `CTRL-D`: Damit können Sie bei eingeschaltetem autoindent eine Tabulatorposition hinter die Einrückung zurückgehen. Wollen Sie ganz an den Zeilenanfang zurück, verwenden Sie **0**`CTRL-D`. Sehr hilfreich ist auch `^CTRL-D`. Diese Tastenkombination entspricht in der umittelbaren Wirkung `CTRL-D`, allerdings wird die nächste Zeile dann wieder so weit eingerückt, wie es die vorige ohne `CTRL-D` gewesen wäre.

Sehr wichtig ist `CTRL-V`. Damit wird die Eingabe von Steuerzeichen in den Text ermöglicht, die der Editor normalerweise ignorieren oder anders interpretieren würde. Stellen Sie einfach `CTRL-V` dem gewünschten Steuerzeichen voran. Es erscheint dann ein `^` auf dem Bildschirm, auf dem der Cursor stehen bleibt. Nach dem Drücken der gewünschten Steuertaste erscheint diese in der Form `^-Taste` auf dem Bildschirm. Hierbei handelt es sich tatsächlich nur um ein Zeichen, was Sie merken, wenn Sie den Cursor darüber bewegen: Der Cursor überspringt nämlich immer das `^`.

8.6 Zurückholen von gelöschten Zeilen

Irren ist menschlich - und so kann es durchaus mal vorkommen, daß man aus Versehen einige Zeilen löscht, die man eigentlich noch gebraucht hätte. Kein Problem – das Kommando **u** macht die letzte Operation rückgängig und ermöglicht so auch die Wiederherstellung von versehentlich gelöschten Zeilen. Das funktioniert aber nur, wenn man sofort nach Ausführung der Löschoperation merkt, daß diese eigentlich nicht gewollt war. Unangenehmer wird es, wenn man den Verlust der Zeilen erst irgendwann später feststellt.

Aber auch für dieses Problem bietet der Editor eine Abhilfe an: Die letzten neun gelöschten Blocks von Zeilen werden in spezielle Puffer abgespeichert, die von 1 bis 9 durchnumeriert sind. Der Puffer *n* enthält dann den an *n*.-letzter Stelle gelöschten Text. Diese speziellen Puffer können beim Zurückholen von Text genauso behandelt werden wie die normalen Kopierpuffer, also fügt z.B. **"***n***p** den Inhalt des *n*-ten Puffers hinter die aktuelle Zeile in den Text ein. Wenn Sie nicht genau wissen, in welchem Puffer Ihr gelöschter Text steht, müssen Sie die Puffer durchprobieren, wobei Sie sich der speziellen Konvention bedienen können, daß das Kommando **.** (Punkt), das normalerweise einfach das letzte Kommando (außer **u**) wiederholt, hier außerdem die Puffernummer hochzählt. Das bedeutet, daß eine Kommandosequenz wie

```
"1pu.u.u.
```

wenn u. lange genug ausgeführt wird, schließlich Ihren versehentlich gelöschten Text wieder zurückbringt.

9 UNIX-Kommandos in vi nutzen

9.1 Shell Escapes

Der vi (oder vielmehr der ex) bietet die Möglichkeit, während einer Editorsitzung gewöhnliche UNIX-Kommandos ausführen zu lassen, und zwar durch Eingabe von

:!*kommando***.**

Wurde der gerade editierte Text noch nicht gesichert, erscheint die Warnung `"[No write sincelast change]"`, das Kommando wird aber trotzdem ausgeführt. Nach der Beendigung des UNIX-Kommandos werden Sie zum Drücken der `RETURN`-Taste aufgefordert, damit der Bildschirm, der vielleicht durch die Ausgaben des UNIX-Kommandos etwas in Unordung gekommen ist, wieder restauriert werden kann.

Der genauere Ablauf dieses Vorgangs, den man `Shell Escape` nennt, sieht so aus, daß zunächst eine neue Shell erzeugt wird und diese dann das gewünschte Kommando ausführt. Daher erhalten Sie auch bei irgendwelchen Schwierigkeiten die üblichen Shell-Fehlermeldungen.

Es ist auch möglich, die Standardausgabe eines UNIX-Kommandos in die gerade bearbeitete Datei umzulenken, sie wird dann hinter der aktuellen Cursorzeile eingefügt. Das entsprechende Kommando lautet:

:r !*kommmando*

:!*kommando*	Ausführen des UNIX-Kommandos *kommando*, zurück in vi
:r **!***kommmando*	Ausführen des UNIX-Kommandos *kommando*, Ausgabe hinter Cursorzeile einfügen

9.2 Bearbeiten von Textteilen mit UNIX-Kommandos

Der vi bietet Ihnen die Möglichkeit, Textstellen mit verschiedenen UNIX-Kommandos zu bearbeiten. Dies erreichen Sie durch den Befehl

!}*UNIX-Kommando*

der mit der `RETURN`-Taste abgeschlossen werden muß. Dieser Befehl bewirkt, daß die Textzeilen ab Cursorzeile bis vor die nächste Leerzeile gelöscht und dem UNIX-Kommando als Standardeingabe übergeben werden. Die Standardausgabe des Kommandos wird dann in den Text an die gleiche Stelle wieder eingesetzt. Aus diesem Grund ist es sinnvoll, hier nur die UNIX-Kommandos mit Filterwirkung zu verwenden. Andere Kommandos können zwar auch angegeben werden, was aber den Nachteil hat, daß der Text von der Cursorzeile bis vor die nächste Leerzeile gelöscht wird, z.B. hat also **!}pwd** die Wirkung, daß statt des gelöschten Textes der Name des Arbeitskataloges in den Text eingesetzt wird.

10 vi-Kommandoübersicht

In diesem Abschnitt werden alle verfügbaren vi-Kommandos zusammengefaßt und kurz beschrieben. Es werden folgende Abkürzungen verwendet:

[*option*]	Bezeichnet einen Teil des vi-Kommandos der nicht unbedingt angegeben werden muß.
[*n*]	Bezeichnet eine Zählung, die vor einem Kommando angegeben werden kann.
{*variabler Teil*}	Kennzeichnet einen Kommandoteil, der zwar angegeben werden muß, aber variieren kann.
<*zeichen* [-*zeichen*]>	Hier muß ein Zeichen aus dem angegebenen Bereich getippt werden.
	Sondertasten werden wie folgt abgekürzt:
ESC	ESCAPE-Taste
CR	RETURN-Taste
LF oder NL	LINE FEED-Taste
BS	BACKSPACE-Taste, entspricht CTRL-H
TAB	TAB-Taste
BELL	entspricht CTRL-G
FF	FORM FEED, entspricht CTRL-L
<sp>	SPACE-Taste
DEL	DELETE-Taste

10.1 Aufruf und Verlassen des vi

Zunächst noch ein paar Worte zum Aufruf und Verlassen des Editors vi: Der Aufruf geschieht mit dem UNIX-Befehl

vi [*option* [...*option*]] [*dateiname* [...*dateiname*]] ...

Folgende Optionen sind zugelassen:

-R	Die Datei wird im Nur-Lese-Modus editiert. Ein Zurückschreiben des editierten Textes auf die gleiche Datei ist dann nicht möglich. Dies entspricht dem Aufruf des UNIX-Befehls view.
-r	Wiederaufnahme von Editorsitzungen, die nicht ordnungsgemäß beendet wurden (z. B. durch Systemcrash). Man erhält in diesem Fall eine Mail und muß vi genau in der dort angegebenen Form aufrufen.
-l	Editoren von LISP-Programmen. Wird diese Option angegeben, sind einige speziell auf LISP-Programme abgestellte Kommandos zugänglich.
-c*kommando*	Die Editorsitzung wird mit dem angegebenen ex-Kommando *kommando* begonnen.

Beim Aufruf des vi kann nicht nur ein Dateiname, sondern eine ganze Liste von zu editierenden Dateien (bzw. eine UNIX-Dateispezifikation auch mit Wildcards) angegeben werden. Die Editorsitzung beginnt dann mit der ersten Datei, die weiteren Dateien können mit dem vi-Kommando **:n** editiert werden.

Zum Verlassen des vi geben Sie die Kommandos **ZZ** oder **:x**, um gleichzeitig den editierten Text abzuspeichern, oder **:q** bzw. **:q!** um am Text vorgenommene Änderungen zu verwerfen. Stellen Sie aber vor dem Tippen eines dieser Kommandos sicher, daß Sie sich im Kommandomodus befinden (ggf. durch Drücken der ESC-Taste), sonst erhalten Sie nicht die gewünschte Wirkung.

:q! RETURN	Verlassen **ohne** Sichern der editierten Datei
:q RETURN	Verlassen von vi, editierte Datei muß vorher gesichert sein
:wq!	Verlassen von vi, mit Sichern der editierten Datei
ZZ	Verlassen **mit** Sichern, falls Datei editiert wurde
:x	wie **ZZ**

10.2 Cursorbewegungen

Die folgenden Kommandos dienen dazu, den Cursor am Bildschirm und im editierten Text an verschiedene Stellen zu bewegen. Die meisten Kommandos zur Cursorbewegung können auch als vi-Objekte verwendet werden.

[*n*] BS [*n*] **h** [*n*] ¨	Bewegt den Cursor um ein Zeichen nach links. Der Cursor kann damit nicht über den linken Rand bewegt werden.
[*n*] CTRL-N [*n*] **j** [*n*] LF [*n*] **Ø**	Bewegt den Cursor um eine Zeile nach unten, wobei ggf. der Bildschirm um eine Zeile nach oben gescrollt wird.
[*n*] CTRL-P [*n*] **k** [*n*] ≠	Bewegt den Cursor um eine Zeile nach oben, wobei ggf. der Bildschirm um eine Zeile nach unten gescrollt wird.
[*n*] <sp> [*n*] **l** [*n*] □	Bewegt den Cursor um ein Zeichen nach rechts. Es ist nicht möglich, den Cursor über das Zeilenende hinaus auf den Anfang der nächsten Zeile zu setzen.
[*n*] **-**	Cursor an den Anfang der vorigen Zeile setzen. Falls nötig, wird dabei gescrollt.
[*n*] **+** [*n*] CR	Cursor an den Anfang der nächsten Zeile setzen und ggf. scrollen.
[*n*] **$**	Cursor ans Zeilenende setzen. Ist eine Zählung angegeben, wird der Cursor auf das Ende der (n-1)-ten Zeile hinter der aktuellen Zeile gesetzt.
? ?	Setzt den Cursor an den Anfang des ersten Wortes in der Zeile.
0	Setzt den Cursor an den physikalischen Zeilenanfang, d. h. in die erste Spalte.
[*n*] **|**	Setzt den Cursor in die n-te Spalte der aktuellen Zeile.
[*n*] **w**	Der Cursor wird an den Anfang des nächsten Wortes gesetzt.
[*n*] **W**	Der Cursor wird an den Anfang des nächsten Wortes gesetzt, dem ein Leerzeichen vorausgeht. Satzzeichen werden ignoriert.

[*n*] **b**	Setzt den Cursor an den Anfang des vorhergehenden Wortes.
[*n*] **B**	Setzt den Cursor an den Anfang des vorhergehenden Wortes, wobei Satzzeichen ignoriert werden.
[*n*] **e**	Setzt den Cursor auf das Wortende.
[*n*] **E**	Wie vorher, aber ohne Berücksichtigung von Satzzeichen.
[*zeile*] **G**	Cursor auf das erste Wort der Zeile mit der angegebenen Zeilennummer setzen. Wird hier keine Zählung verwendet, bezieht sich das Kommando auf die letzte Zeile.
[*n*] CTRL-D	Scrollt den Bildschirm um n *Zeilen* nach oben, wobei der Cursor in der aktuellen Bildschirmzeile bleibt. Wird keine Zählung angegeben, so wird die zuletzt bei einem solchen Kommando angegebene Zählung verwendet. Anfangswert ist eine halbe Bildschirmseite. Kann nicht als vi-Objekt verwendet werden.
[*n*] CTRL-U	Scrollt den Bildschirm um *n* Zeilen nach unten. Sonst wie oben.
[*n*] CTRL-F	Setzt den Cursor an den Anfang der nächsten (Bildschirm-) Seite. Kann nicht als vi-Objekt verwendet werden.
[*n*] CTRL-B	Setzt den Cursor an den Anfang der vorherigen (Bildschirm-) Seite. Kann nicht als vi-Objekt verwendet werden.
[*n*] **)**	Cursor an den nächsten Satzanfang setzen. Ein Satz ist für den vi ein Textstück, das endet mit **.**, **!** oder **?**, gefolgt von zwei Leerzeichen, einem Leerzeichen und NL oder zwei NL.
[*n*] **(**	Cursor zurück an Satzanfang setzen.
[*n*] **}**	Cursor an den Anfang des nächsten Absatzes setzen. Ein Absatz wird für gewöhnlich durch eine Leerzeile begrenzt. Beim Editieren von nroff-Dokumenten werden auch die Kommandos .IP, .LP, .PP und . QP in der -ms Makrobibliothek sowie das gewöhnliche nroff-Kommando .bp als Begrenzungen eines Absatzes erkannt.
[*n*] {	Cursor zurück an den Anfang eines Absatzes setzen.
]]	Setzt den Cursor an den Anfang des nächsten Kapitels. Ein Kapitel ist normalerweise begrenzt durch eine FF NL-Sequenz oder eine Zeile, die mit anfängt. Beim Editieren von nroff-Dokumenten können auch die -ms-Makros . NH, . SH und . H als Kapitelbegrenzung verwendet werden.
[[	Setzt den Cursor zurück an den Kapitelanfang.
? ?	Steht der Cursor auf einer öffnenden oder schließenden Klammer (gleich ob runde, eckige oder geschweifte), wird er durch dieses Kommando auf die zugehörige andere Klammer gesetzt. Falls der Cursor auf einem anderen Zeichen steht, wird er so lange nach vorne bewegt, bis eine Klammer erreicht wird und dann so verfahren, wie wenn der Cursor von Anfang an auf dieser Klammer gestanden hätte.
[*n*] **H**	Der Cursor wird auf die erste Bildschirmzeile, bzw. bei Vorhandensein einer Zählung auf die *n*. Bildschirmzeile gesetzt.

[*n*]**L**	Der Cursor wird auf die letzte (bzw. auf die *n*.-letzte) Bildschirmzeile gesetzt.
M	Setzt den Cursor auf die Zeile in der Bildschirmmitte.
m<a-z>	Markiert die Cursorposition mit dem angegebenen Buchstaben. Mit den folgenden Kommandos kann der Cursor von einer beliebigen Stelle wieder an diese Position gesetzt werden.
'<a-z>	Setzt den Cursor auf den Zeilenanfang der mit dem angegebenen Buchstaben markierten Zeile.
`<a-z>	Setzt den Cursor exakt auf die mit dem angegebenen Buchstaben markierte Textstelle.
''	Setzt den Cursor auf den Anfang der Zeile zurück, in der er sich vor der letzten nicht-relativen Cursorbewegung befand. Eine nicht-relative Cursorbewegung ist z. B. eine Such- oder Sprungoperation, nicht aber ein zeilenweises Bewegen des Cursors oder Scrollen.
``	Wie oben, allerdings wird der Cursor auf seine exakte Position zurückgesetzt.

10.3 Cursorbewegung durch Suche nach Text

Die folgenden Kommandos dienen zur Suche eines bestimmten Buchstabens oder einer bestimmten Textstelle.

[*n*]**f**{*chr*}	Sucht auf der aktuellen Zeile das nächste Vorkommnis des Zeichens *chr*. Der Cursor wird, falls die Suche erfolgreich ist, auf das gefundene Zeichen gesetzt.
[*n*]**b**{*chr*}	Sucht auf der aktuellen Zeile das vorige Vorkommnis der Zeichens *chr*. Der Cursor wird bei erfolgreicher Suche auf das gefundene Zeichen gesetzt.
[*n*]**t**{*chr*}	Sucht auf der aktuellen Zeile das nächste Vorkommnis des Zeichens *chr*. Der Cursor wird bei erfolgreicher Suche vor das gefundene Zeichen gesetzt.
[*n*]**T**{*chr*}	Sucht auf der aktuellen Zeile das vorige Vorkommnis des Zeichens *chr*. Der Cursor wird bei erfolgreicher Suche hinter das gefundene Zeichen gesetzt.
[*n*]**;**	Wiederholung der letzten f, F, t oder T-Operation in der gleichen Richtung.
[*n*]**,**	Wiederholung der letzten f, F, t oder T-Operation, aber in der entgegengesetzten Richtung.
[*n*]**/**[*string*]**/**NL	Sucht nach dem nächsten Vorkommnis von *string* im Text. Wird der String im Rest des Textes nicht gefunden, wird vom Textanfang aus gesucht. Der abschließende / kann auch weggelassen werden.

[*n*]**?**[*string*]**?**NL	Sucht nach dem vorigen Vorkommnis von *string* im Text. Wird der String bis zum Textanfang nicht gefunden, wird die Suche vom Textende aus fortgesetzt. Das abschließende ? kann auch weggelassen werden.
n	Wiederholt das letzte / oder ?-Kommando in der gleichen Richtung.
N	Wiederholt das letzte / oder ?-Kommando, aber in der anderen Richtung.

10.4 Text einfügen, löschen und ersetzen

a*{text}*ESC	Text hinter der Cursorposition einfügen.
A*{text}*ESC	Text hinter dem Ende der aktuellen Zeile einfügen.
i*{text}*ESC	Text vor der Cursorposition einfügen.
I*{text}*ESC	Text am Anfang der aktuellen Zeile einfügen.
o*{text}*ESC	Eine neue Zeile wird unterhalb der aktuellen Zeile begonnen und dort der Text eingefügt.
O*{text}*ESC	Eine neue Zeile wird oberhalb der aktuellen Zeile begonnen und dort der Text eingefügt.
[*n*]x	Löscht das Zeichen auf der aktuellen Cursorposition. Der Rest der Zeile wird nachgezogen.
[*n*]**X**	Löscht das Zeichen unmittelbar vor der aktuellen Cursorposition. Der Rest der Zeile (einschließlich Cursor) wird nachgezogen.
D	Löscht den Zeilenrest ab der Cursorposition.
[*n*]**d***{WB}*]	Löschen eines Wirkungsbereiches WB ab Cursor. Siehe hierzu Abschnitt 8.4
d d	Löschen der aktuellen Textzeile, in der Cursor steht.
[*n*]**r***{chr}*	Ersetzt das Zeichen auf der Cursorposition durch *chr*.
[*n*]**R***{text}*ESC	Allgemeines Kommando zum Ersetzen von Text. Die durch das Eintippen des neuen Textes betroffene Textstelle wird ab der Cursorposition vom neuen Text überschrieben (das gilt auch für Folgezeilen, wenn mehrere Zeilen getippt werden).
[*n*]**s***{text}*ESC	Es werden *n* Zeichen (ab Cursor) durch den eingetippten Text ersetzt. Zur Information darüber, wo der zu ersetzende Bereich endet, erscheint ein $-Zeichen auf der letzten Position, die von der Ersetzung betroffen ist.
[*n*]**S***{text}*ESC	Ersetzt die aktuelle Zeile (bzw. *n* Zeilen ab der aktuellen Zeile) durch den eingegebenen Text.

[*n*]**c***{WB}{text}*ESC	Der angegebene Wirkungsbereich WB wird ab Cursor durch den eingetippten Text ersetzt. Es erscheint wieder ein $-Zeichen auf dem Ende des ersetzten Bereichs.

10.5 Text bewegen

Die folgenden Kommandos können nicht nur zum Verschieben, sondern u. a. auch zum Kopieren von Textstellen verwendet werden.

["<*a-z*>[*n*]]**y***{WB}*	Der spezifizierte Wirkungsbereich WB wird in den mit dem angegebenen Buchstaben bezeichneten Puffer kopiert, oder, falls keine Pufferbezeichnung angegeben ist, in den Standardpuffer (diese Konvention gilt auch für die folgenden Kommandos).
"<a-z>[*n*]**Y**	Kopiert den Inhalt der Cursorzeile in den spezifizierten Puffer.
"<a-z>[*n*]**p**	Der Inhalt des spezifizierten Puffers wird hinter die Cursorposition kopiert. Enthält der Puffer eine oder eine Folge von ganzen Zeilen, wird hinter die Cursorzeile kopiert.
"<a-z>[*n*]**P**	Der Inhalt des spezifizierten Puffers wird vor die Cursorposition kopiert. Enthält der Puffer eine oder eine Folge von ganzen Zeilen, wird vor die Cursorzeile kopiert.
[*n*]□**>***{WB}*]	Alle Zeilen des angegebenen Wirkungsbereiches WB werden um den Wert der Variablen shiftwidth nach rechts verschoben.
[*n*]□**>>**	Verschiebt die aktuelle Zeile um shiftwidth Spalten nach rechts.
[*n*]□**<***{WB}*]	Alle Zeilen des angegebenen Wirkungsbereiches WB werden um shiftwidth Spalten nach links verschoben, so weit dies möglich ist (also nur, bis in der ersten Spalte kein Leerzeichen mehr steht).
[*n*]□**<<**	Verschiebt die aktuelle Zeile um shiftwidth Spalten nach links.
[*n*]□**=***{WB}*	Formatiert den angegebenen Wirkungsbereich WB gemäß en Konventionen für LISP-Ausdrücke. Der Bereich sollte also tatsächlich aus einem LISP-Ausdruck bestehen, da sich sonst etwas wunderlich anmutende Resultate ergeben könnten.

10.6 Andere Kommandos

In diesem Abschnitt werden verschiedene Kommandos besprochen, die sich nicht oder nicht eindeutig anderen Abschnitten zuordnen lassen.

Z Z	Verlassen des vi. Wurden Änderungen am Text vorgenommen, die noch nicht gesichert wurden, erfolgt automatische Sicherung auf die aktuelle Datei.
CTRL-L	Neuerstellen des Bildschirms. Dieses Kommando ist sehr nützlich, wenn der Bildschirm durch Ausgaben von Hintergrundprozessen oder Systemmeldungen etwas in Unordnung geraten ist (Bei manchen Terminals CTRL-R).
.	Wiederholt das letzte Editorkommando, das eine Änderung am Text bewirkt hat.
u	Macht die letzte textändernde Editoroperation rückgängig.
U	Macht alle textändernden Editoroperationen auf der aktuellen Zeile rückgängig. Es darf aber keine Cursorbewegung gegeben haben, mit der die Zeile verlassen wurde.
[*n*] **J**	Zusammenhängen der aktuellen und der folgenden Zeile. Die folgende Zeile wird an das Ende der aktuellen Zeile angefügt. In den meisten Fällen wird noch ein Leerzeichen dazwischen geschaltet, wenn die aktuelle Zeile mit einem Punkt abgeschlossen war, zwei Leerzeichen.
Q	Dauerhaftes Umschalten in den ex-Modus. Sie bleiben dann so lange im ex-Modus, bis Sie dort durch das Kommando vi zurück in den vi-Modus wechseln. Zum ex-Modus siehe auch den Kapitel 12 mit einer Übersicht über die ex-Kommandos.
! *{WB}kommando* NL	Der angegebene Wirkungsbereich WB wird im Text gelöscht und als Standardeingabe für das UNIX-Kommando verwendet. Dessen Standardausgabe wird statt der gelöschten Textstelle in den Text eingefügt. Es empfiehlt sich, hier nur UNIX-Filterkommandos zu verwenden.
z *{n}* NL	Die aktuelle Fenstergröße wird auf *n* Zeilen gesetzt und dann der Bildschirm neu erstellt.
CTRL-G	Zeigt Informationen über die aktuell editierte Datei an.

11 Der Editor ex

Dieser Abschnitt soll Ihnen, wenn Sie im vi schon einigermaßen geübt sind, die (den vi in vielfacher Weise ergänzenden) Möglichkeiten des Editors ex (bzw. des ex-Modus des Editors vi) aufzeigen.

11.1 Aufruf von ex

11.1.1 Direktaufruf

Von UNIX aus können Sie den ex mit dem Kommando

`ex dateiname(n)`

aufrufen. Zusätzlich können Sie bestimmte Optionen angeben. Diese sind genau die gleichen wie beim Aufruf von vi (beschrieben in Abschnitt 10.1.). Eine spezielle Option ist - (Minuszeichen). Damit werden alle Bildschirmausgaben zur Information des Benutzers unterdrückt. Diese Option ist sinnvoll beim Einsatz von ex in Shell Scripts.

Beim Aufruf von ex prüft dieser zunächst einmal nach, ob die Variable EXINIT gesetzt ist. Ist dies der Fall, führt ex zunächst die Kommandos aus, die in dieser Variable stehen. Im anderen Fall sieht ex nach, ob Sie eine Datei namens .exrc in Ihrem Homedirectory haben, und führt ggf. die darin stehenden Kommandos aus. Dies ist nützlich, um z. B. gewisse Variablen gleich am Anfang jeder Editorsitzung zu setzen bzw. zu verändern.

11.1.2 Aufruf des ex-Modus aus dem vi

Vom vi-Kommandomodus aus können Sie den ex-Modus durch **:** für ein einziges ex-Kommando aufrufen; nach Abarbeitung dieses Kommandos gelangen Sie automatisch wieder in den vi-Kommandomodus. Wollen Sie mehrere Kommandos im ex-Modus absetzen, wechseln Sie mit **Q** in den ex-Modus, aus dem Sie dann durch Aufruf des Kommandos **vi** wieder in den vi-Kommandomodus zurückkommen. Der Cursor wird auf diejenige Zeile positioniert, die von ex zuletzt bearbeitet wurde.

11.2 Dateimanipulation

ex (wie auch vi) merkt sich den Namen der gerade aktuellen Datei als aktuellen Dateinamen. Alle Editoroperationen werden nicht direkt in der Datei vorgenommen, sondern in einem Puffer, in den zu Beginn der Editorsitzung der Inhalt der zu editierenden Datei gelesen wird. Demzufolge haben Änderung im Text so lange keine Wirkung auf die Datei selbst, bis der Inhalt des Puffers explizit mit write auf die aktuelle Datei (oder eine andere) zurückgeschrieben wird (dadurch geht natürlich der alte Dateiinhalt verloren). Wird eine neue Datei editiert, so wird der Name dieser Datei als aktueller Dateiname übernommen und der Dateiinhalt wird in den Editierpuffer eingelesen.

Immer dann, wenn sich der aktuelle Dateiname ändert, wird sein vorheriger Wert als alternativer Dateiname gemerkt. Der alternative Dateiname wird auch dann neu gesetzt, wenn eine Datei angesprochen wird, die nicht die aktuelle Datei ist.

Wenn Sie bei irgendwelchen ex-Kommandos Dateinamen angeben müssen, können Sie das in der von UNIX gewohnten Weise tun. Zusätzlich wird das Zeichen **%** in einem Dateinamen durch den aktuellen Dateinamen, das Zeichen **#** durch den alternativen Dateinamen ersetzt. Damit sparen Sie sich das Tippen dieser beiden Dateinamen.

11.3 Editormodi

Der Editor ex hat im wesentlichen zwei Editormodi. Einerseits den gewöhnlichen Kommandomodus, der durch den Prompt `:` vor jeder Eingabezeile angezeigt wird. Im Kommandomodus können Sie ex-Kommandos eingeben, die Sie mit der `RETURN`-Taste zur Ausführung bringen.

Der zweite wichtige Modus ist der Einfügemodus, der normalerweise daran erkenntlich ist, daß kein Prompt angezeigt wird (siehe Editorvariable `showmode`). In diesem Modus werden eingegebene Zeilen direkt in den editierten Text eingefügt. Den Einfügemodus verlassen Sie wieder durch Eingabe eines `.` (Punkt) in einer neuen Zeile (und abschließendes Drücken von `RETURN`).

11.4 Struktur der ex-Kommandos

Die meisten ex-Kommandos sind englische Wörter, die Sie allerdings nicht auszuschreiben brauchen, sondern auch abkürzen können (siehe Kapitel 12 für eine Übersicht über alle ex-Kommandos).

Vor den meisten Kommandos können Sie explizit einen Zeilenbereich angeben, in dem die Kommandos arbeiten sollen. Dieser Zeilenbereich besteht entweder aus einer einzigen Zeilennummer (dann arbeitet das Kommando auf dieser Zeile allein) oder aus zwei durch Komma getrennten Zeilennummern, die erste bzw. letzte Zeile des Bereichs angeben. Der ex merkt sich die letzte bearbeitete Zeile als aktuelle Zeile; diese wird dann verwendet, wenn kein Zeilenbereich angegeben wurde. Eine genauere Diskussion der Möglichkeiten, einen Zeilenbereich vorzugeben (auch Adressierung genannt), erfolgt im nächsten Abschnitt.

Bei einigen Kommandos ist es möglich, eine Zählung anzugeben, die festlegt, auf wieviele Zeilen sich das Kommando beziehen soll. Diese Zählung muß - im Unterschied zu der beim vi gebräuchlichen Weise - hinter dem Kommando angegeben werden. Einige Kommandos existieren in zwei Formen. Die Standardform erhalten Sie einfach durch Eingabe des gewöhnlichen Kommandonamens, die alternative Form, indem Sie werden.

Nicht wenige Kommandos verlangen spezielle Parameter, z. B. Dateinamen oder reguläre Ausdrücke. Diese sind dann in der Kommandoübersicht genauer besprochen.

Einige Kommandos existieren in zwei Formen. Die Standardform erhalten Sie einfach durch Eingabe des gewöhnlichen Kommandonamens, die alternative Form, indem Sie den Kommandonamen mit einem Ausrufezeichen ! abschließen.

Hinter vielen Kommandos können Sie die Zeichen (sog. Flags) #, p oder l angeben (wobei p und l eine Leerstelle vorausgehen muß). Diese Zeichen sind Abkürzungen für Kommandos, die ex nach Beendigung des vorherigen Kommandos ausführt, z. B. zeigt p die aktuelle Zeile auf dem Bildschirm an (es ist allerdings überflüssig, p am Ende eines Kommandos anzugeben, da der Editor ex nach jeder Änderung am Text sowieso die aktuelle Zeile anzeigt).

Für Editor-Kommandoprozeduren eignet sich die Möglichkeit, Kommentarzeilen, die der Editor nicht weiter beachtet, zu generieren. Ein Kommentar wird durch die doppelten Anführungsstriche " eingeleitet und kann entweder am Zeilenanfang positioniert werden (was dazu führt, daß die ganze Zeile ignoriert wird), oder aber hinter ein Kommando gestellt werden. Bitte beachten Sie, daß es in manchen Fällen nicht möglich ist, hinter ein Kommando noch einen Kommentar zu stellen, nämlich dann, wenn der Editor ex das einleitende Anführungszeichen als zum Kommando gehörigen Text auffaßt (vor allem bei substitute- und map-Kommandos).

Schließlich haben Sie auch noch die Möglichkeit, mehrere Kommandos in eine Kommandozeile zu schreiben. Dazu werden die Kommandos mit dem senkrechten Strich `|` getrennt. Bei allen globalen Kommandos (siehe Kapitel 12) ist dies jedoch nicht möglich, ebenso wie bei Shell `Esc`apes, die immer allein in einer Kommandozeile stehen müssen.

11.5 Adressierung der Kommandos

Für die Adressierung der ex-Kommandos, d.h. die Vorgabe von Zeilenbereichen, gibt es folgende Möglichkeiten:

`.`	Bezeichnet die aktuelle Zeile. Die meisten ex-Kommandos stellen die letzte bearbeitete Zeile als neue aktuelle Zeile ein. Da sich die meisten Kommandos ohne weitere Bereichsvorgaben sowieso auf die aktuelle Zeile beziehen, wird . als alleinige Adresse wohl eher selten Anwendung finden.
n	Die *n*-te Zeile im editierten Text. Die Zeilen werden bei 1 beginnend durchnumeriert.
`$`	Die letzte Zeile im editierten Text.
`%`	Der ganze editierte Text (ist eine Abkürzung für 1,$).
+n bzw. *-n*	Es wird die Zeile relativ zur aktuellen Zeile bezeichnet. Bezeichnungen der Form .+3, +3 und +++ sind äquivalent.
/pat/	Es wird vorwärts nach einer Zeile gesucht, die den regulären Ausdruck *pat*, enthält; diese bildet dann die Zeile, auf die das Kommando wirkt. Wird *pat* nicht angegeben, so wird nach dem zuletzt angegebenen regulären Ausdruck gesucht. Bitte beachten Sie, daß die Suche standardmäßig zyklisch ist, d.h. wenn der Ausdruck bis zum Textende nicht gefunden wird, wird am Textanfang weitergesucht.
`?`*pat*`?`	Wie oben, allerdings in der Rückwärtsrichtung.
`''`	Vor jeder nicht-relativen Änderung der Position der aktuellen Zeile wird diese mit der Marke `''` (zweimal das Zeichen `'`) versehen. Sie brauchen nur diese Marke anzugeben, wenn Sie sich auf diese Zeile beziehen wollen.
`'x`	Bezieht sich auf die zuvor mit `mark` gekennzeichnete Zeile.

Die oben angegebenen Adressierungsarten können Sie auch kombinieren, da eine Kommandoadresse aus einer Liste von (durch Komma oder Semikolon getrennten) Adressen

besteht. Wenn zwei Adressen durch ein Semikolon getrennt werden, so wird die aktuelle Zeile auf den Wert der vor dem Semikolon stehenden Adresse gesetzt, bevor die nächste Adresse interpretiert wird. In der Liste der Adressen können Sie auch die Adresse weglassen; es wird dann die aktuelle Zeile dafür eingesetzt. So ist z.B. `,100` gleichwertig mit `.,100`.

11.6 Reguläre Ausdrücke

Wie beim vi kann bei Suchkommandos und beim Ersetzungskommando (substitute) im Suchbegriff ein regulärer Ausdruck verwendet werden. Für die Bildung der regulären Ausdrücke gelten dieselben Regeln wie beim vi (siehe Abschnitt 7.1.4), inbesondere was die Verfügbarkeit und Bedeutung der Metasymbole betrifft. Der zuletzt angegebene Suchbegriff wird von ex gespeichert und immer dann eingesetzt, wenn der leere reguläre Ausdruck verwendet wird, also bedeutet

`//` erneute Suche mit altem Suchbegriff in Suchrichtung
`??` Umkehren der Suchrichtung und erneute Suche mit altem Suchbegriff

Beim Ersetzen von Text mit dem substitute-Kommandos werden zwei Begriffe angegeben: der Such- und der Ersetzungsbegriff. In diesen können nun (im Magic-Modus) die folgenden Metasymbole verwendet werden:

`&` letzter Suchbegriff
`~` letzter Ersetzungsbegriff

Mit dem Klammerpaar `\(...\)` können reguläre Ausdrücke im Suchstring eingeklammert werden. Diese können im Ersetzungstext ihrer Reihenfolge (im Suchstring) entsprechend mit `\1`, `\2` usw. angesprochen werden.

Beispiel:

`1,$s/end/ende/`	ersetze in der gesamten Datei `end` durch `ende`
`1,$s/~/ENDE/`	ersetze in der gesamten Datei `ende` durch `ENDE`
`1,$s/END/&IF/`	ersetze in der gesamten Datei `END` durch `ENDIF`

12 Übersicht über die ex-Kommandos

Alle ex-Kommandos genügen der folgenden Syntax

```
adresse kommando ! parameter zählung flags
```

Hierbei ist zu beachten, daß alle Teile eines Kommandos optional sind; wird z.B. überhaupt nichts angegeben (leeres Kommando), gibt ex einfach die nächste Zeile der Datei auf dem Bildschirm aus.
Das Flag am Ende eines ex-Kommandos kann nur aus den Zeichen #, l oder p bestehen. Diese Flags bewirken, daß die durch das ex-Kommando berührten Zeilen auf dem Bildschirm angezeigt werden, und zwar mit:

Angabe der Zeilennummer
l Darstellung des Zeilenendes durch ein $–Zeichen

p die reine Ausgabe der Zeile
(Der Sinn dieses Flags ist eher historisch zu verstehen, da ex-Kommandos normalerweise stets die bearbeiteten Zeilen ausgeben)

In der folgenden Befehlsübersicht sind die Ersatzwerte für Adressen in Klammern angegeben, diese Klammern sind aber kein Teil des Kommandos.

abbreviate *ak wort* Abkürzung: **ab**

Definiert eine Abkürzung für den vi-Einfügemodus. Wird dort *ak* als ein Wort eingegeben, wird dieses durch *wort* ersetzt. Siehe Abschnitt 8.2.

adr **append** Abkürzung: **a**
text
.

Der angegebene *text* wird hinter der adressierten Zeile eingefügt. Nach Beendigung des Kommandos adressiert. die letzte eingefügte Zeile. Wird die adr `0` angegeben, wird der Text vor der ersten Zeile eingefügt. Wird das Kommando mit einem Ausrufezeichen gegeben, so bewirkt dies die Umschaltung des autoindent-Flags für die Dauer der Texteinfügung.

args

Gibt die Argumentliste des ex-Kommandos aus. Das aktuelle Argument (der Name der editierten Datei) wird dabei in eckige Klammern gesetzt.

adr1,*adr2* **change** *zählung* Abkürzung: **c**
text

Die angegebenen Zeilen werden durch den eingegebenen *text* ersetzt. Nach der Ausführung dieses Kommandos wird die letzte eingegebene Zeile zur aktuellen Zeile. Wurde kein Text eingegeben, entspricht die Wirkungsweise dieses Kommandos dem Kommando `delete`. Ein Ausrufezeichen bei diesem Kommando bewirkt die Umschaltung des `autoindent`-Flags für die Dauer der Texteingabe.

adrs1,*adrs2* **copy** *adrs flags* Abkürzung: **co**

Der angegebene Zeilenbereich wird hinter die Zeile *adrs* kopiert (hier kann auch `0` angegeben werden, was bewirkt, daß der Zeilenbereich an den Textanfang kopiert wird). Die letzte Zeile der Kopie wird zur aktuellen Zeile.

adrs1,*adrs2* **delete** *puffer zählung flags* Abkürzung: **d**

Löscht den angegebenen Zeilenbereich aus dem Text. Dieser kommt in den Standardpuffer, falls kein *puffer* angegeben ist. Die Zeile hinter der letzten gelöschten Zeile wird zur aktuellen Zeile; wird am Textende gelöscht, wird die letzte Textzeile zur aktuellen Zeile. Ist *puffer* durch einen Buchstaben angegeben, wird der Zeilenbereich in den benannten Puffer übernommen.

edit *dateiname* Abkürzung: **e**

Editieren der angegebenen Datei. Bevor das Kommando ausgeführt wird, prüft ex, ob der gerade editierte Text noch nicht gesichert wurde. Ist dies der Fall, so wird das Kommando nicht ausgeführt (verwenden Sie e!, wenn Sie nicht gesicherte

Änderungen verwerfen wollen). Im anderen Fall lädt ex die zu editierende Datei in den Textspeicher, wobei der alte Text verloren geht. Als Bestätigung, daß der Ladevorgang erfolgreich abgeschlossen wurde, wird der Name der Datei und die Zahl der Zeilen und Zeichen in der Datei auf dem Bildschirm ausgegeben. Diese Datei wird nun zur aktuellen Datei. Die aktuelle Zeile ist die letzte Zeile der Datei.

file Abkürzung: **f**

Gibt den Namen der aktuellen Datei und verschiedene Informationen über diese Datei aus: Ob sie bearbeitet wurde ([Modified]), ob sie nur gelesen werden kann ([Read only]), die aktuelle Zeile, die Gesamtzahl der Zeilen und die prozentuale Position der aktuellen Zeile im Verhältnis zur Gesamtzeilenzahl.

file *dateiname*

Der aktuelle Dateiname wird auf *dateiname* gesetzt, wobei diese Datei in den Zustand "nicht-editiert" versetzt wird.

1,$ global */pat/ cmds* Abkürzung: **g**

Dieses Kommando markiert zunächst jede Zeile im angegebenen Bereich, die einen Text enthält, auf den der reguläre Ausdruck *pat* paßt. Danach werden die Kommandos *cmds* für jede gefundene Zeile (die als aktuelle Zeile gesetzt wird) ausgeführt. *cmds* ist eine Kommandoliste, die aus der hinter *pat* stehenden Eingabe besteht. Diese Kommandoliste kann sich über mehrere Zeilen erstrecken, wenn alle Zeilen außer der letzten mit einem Backslash abgeschlossen werden. Ist *cmds* leer, so wird jede Zeile auf dem Bildschirm ausgegeben, auf die *pat* paßt. Wird hinter **g** ein Ausrufezeichen angegeben, so dreht sich die Wirkung um: Die Kommandos *cmds* werden dann über alle Zeilen ausgeführt, auf die *pat* nicht paßt.

adr **insert** Abkürzung: **i**
text
.

Der angegebene *text* wird vor der angegebenen Zeile eingefügt. Die letzte eingegebene Zeile wird zur aktuellen Zeile. Wird das Ausrufezeichen angegeben, so wird die Variable autoindent für die Zeit der Texteingabe umgeschaltet.

adr1,adr2 **join** *zählung flags* Abkürzung: **j**

Hängt die angegebenen Zeilen zu einer einzigen Zeile zusammen. Beim Zusammenfügen werden zwei Leerstellen eingefügt, wenn eine Zeile mit einem Punkt endet; es wird kein Leerzeichen eingefügt, wenn das erste Zeichen der folgenden Zeile eine Klammer zu ist. In allen anderen Fällen wird der Umbruch durch eine Leerstelle ersetzt. Bei Verwendung des Kommandos mit dem Ausrufezeichen werden keine Leerzeichen an den Umbruchstellen eingefügt.

adr **k** *x*

Ist ein Synonym für das `mark`-Kommando. Zwischen dem Kommandozeichen und dem folgenden Buchstaben braucht keine Leerstelle zu stehen.

adr1,adr2 **list** *zählung flags*

Listet den angegebenen Zeilenbereich auf dem Bildschirm, wobei Zeilenenden

durch `$`, `TAB` durch `^I` dargestellt werden. Die aktuelle Zeile ergibt sich aus der letzten gelisteten Zeile.

map *ls rs*

Mit diesem Kommando definiert man Makros für den vi-Modus. Bitte schlagen Sie im Abschnitt 8.2 eine genauere Beschreibung dieses Kommandos nach.

adr **mark** *x*

Markiert die angegebene Zeile mit dem Kleinbuchstaben *x*. Der Buchstabe muß durch ein Leerzeichen vom Kommandonamen abgesetzt sein. Nach diesem Kommando kann die markierte Zeile durch `'x` adressiert werden. Die aktuelle Zeile wird durch dieses Kommando nicht verändert.

adr1,*adr2* **move** *adrs* Abkürzung: **m**

Dieses Kommando verschiebt einen Zeilenbereich hinter die Zeile *adrs*. Die aktuelle Zeile ergibt sich aus der ersten verschobenen Zeile.

next Abkürzung: **n**

Editiert die nächste Datei aus der Argumentliste des ex-Kommandos. Es werden die gleichen Sicherheitsüberprüfungen vorgenommen wie beim edit-Kommando, und Sie müssen ebenfalls ein Ausrufezeichen angeben, um diese abzuschalten.

n *dateinamenliste*

Die angegebene Liste wird nach den Regeln der Shell für Dateinamen-Expansion expandiert und ersetzt dann die Argumentliste des ex-Kommandos. Anschließend wird die erste Datei in dieser Liste editiert.

adr1,*adr2* **number** *zählung flags* Abkürzung: **#** oder **nu**

Gibt den angegebenen Zeilenbereich auf den Bildschirm aus, wobei jeder Zeile ihre Nummer vorangestellt wird. Die aktuelle Zeile ergibt sich aus der letzten ausgegebenen Zeile.

adr1,*adr2* **print** *zählung* Abkürzung: **p** oder **P**

Ausgabe des angegebenen Zeilenbereichs auf dem Bildschirm. Die letzte ausgegebene Zeile wird zur aktuellen Zeile.

adr **put** *puffer* Abkürzung: **pu**

Fügt zuvor gelöschte oder in *puffer* übernommene Zeilen wieder in den Text ein. Dieses Kommando wird meist in Verbindung mit delete zum Verschieben von Zeilen bzw. `yank` zum Kopieren von Zeilen benutzt. Wird kein *puffer* angegeben, so wird der zuletzt gelöschte oder kopierte Text zurückgeholt, ansonsten wird der Inhalt von *puffer* hinter die adressierte Zeile eingefügt.

quit Abkürzung: **q**

Beenden von ex ohne Zurückschreiben des editierten Textes. Um den Editor zu verlassen, ohne einen editierten, aber nicht gesicherten Text zurückzuschreiben, muß **quit!** bzw. **q!** verwendet werden.

adr **read** *dateiname* Abkürzung: **r**

Der Inhalt der angegebenen Datei wird hinter die angegebene Zeile eingefügt.

Falls der *dateiname* weggelassen wird, wird der Name der aktuellen Datei verwendet. Wie üblich bewirkt hier Adresse `0`, daß der Dateiinhalt an den Textanfang gestellt wird. Die aktuelle Zeile ist die zuletzt eingelesene Zeile.

adr **read** **!***unix-kommando* Abkürzung: **r**

Die Standardausgabe des angegebenen *unix-kommandos* wird hinter der angegebenen Zeile der Datei eingefügt.

rewind Abkürzung: r**ew**

Die ex-Argumentliste wird wieder auf die erste Datei zurückgesetzt. Diese wird dann editiert; **rewind!** bzw. **rew!** prüft nicht, ob irgendwelche ungesicherten Änderungen am Text vorgenommen wurden.

set *parameter*

Dient zum Setzen von Editorvariablen. Siehe Abschnitt 7.1.

shell Abkürzung: **sh**

Erzeugt eine neue Shell. Die Editorsitzung wird fortgesetzt, wenn diese Shell beendet wird.

source *dateiname* Abkürzung: **so**

Lesen und Ausführung der in der Datei *dateiname* stehenden ex-Kommandos.

stop

Stoppt den Editor und hat somit die gleiche Wirkung wie `CTRL-Z`. Die Editorsitzung kann dann durch Eingabe von fg in der UNIX-Shell wieder fortgesetzt werden. Ist die Variable autowrite gesetzt, wird bei Vorhandensein ungesicherter Änderungen erst der Text gesichert, außer wenn ein Ausrufezeichen hinter dem Kommando angegeben wird.

adr1,*adr2* **substitute** /*pat*/*repl*/*optionen zählung flags* Abkürzung: **s**

Ersetzt im Zeilenbereich jeweils das zeilenweise erste Auftreten von *pat* durch *repl*. Um alle Vorkommnisse von *pat* in einer Zeile zu ersetzen, müssen Sie die Option **g** angeben. Die Option **c** bewirkt, daß vor dem Ersetzen einer Textstelle eine Sicherheitsabfrage vorgenommen wird, die Sie mit **y** beantworten müssen, um die Ersetzung durchzuführen. Nach der Ausführung des Kommandos wird die letzte Zeile, in der eine Ersetzung vorgenommen wurde, zur aktuellen Zeile. Werden *pat* und *repl* weggelassen, wird das vorhergehende substitute-Kommando wiederholt. In diesem Fall ist das ein Synonym für das Kommando `&`.

adr1,*adr2* **t** *adrs flags*

Synonym für copy.

unabbreviate *wort* Abkürzung: **una**

wort wird im folgenden nicht mehr als Abkürzung verwendet. Siehe Abschnitt 8.2.

undo Abkürzung: **u**

Macht die Änderungen rückgängig, die das letzte Editorkommando verursachte.

unmap *ls*

Die Makrodefinition für *ls* wird rückgängig gemacht. Siehe Abschnitt 8.2.

adr1,adr2 **v** */pat/cmds*

Synonym für das g**!**-Kommando.

version Abkürzung: **v**

Gibt die aktuelle Versionsnummer des Editors sowie das Datum der letzten Änderung aus.

adr **visual** Abkürzung: **vi**

Geht in den vi-Modus, wobei die angegebene Zeile als erste am Bildschirm angezeigt wird. Um wieder in den ex-Modus zu kommen, müssen Sie das Kommando Q geben.

adr1,adr2 **write** *dateiname* Abkürzung: **w**

Schreibt den Zeilenbereich in die Datei mit dem Namen *dateiname*. Wird *dateiname* nicht angegeben, so wird der Name der aktuellen Datei verwendet. Verwenden Sie die Form mit dem Ausrufezeichen in folgenden Fällen: Um eine existierende Datei zu überschreiben, die nicht die aktuelle Datei ist bzw. um einen Teilbereich auf die aktuelle Datei zu schreiben.

adr1,adr2 **write>>***dateiname* Abkürzung: **w>>**

Schreibt den Zeilenbereich an das Ende der angegebenen Datei.

adr1,adr2 **w !***unix-kommando*

Verwendet den angegebenen Zeilenbereich als Standardeingabe von *unix-kommando*. Beachten Sie bitte das Leerzeichen zwischen **w** und **!**.

wq *dateiname*

Wirkt wie write und anschließend quit.

xit *dateiname* Abkürzung: **x**

Verlassen des Editors. Der editierte Text wird nur dann gesichert, wenn noch nicht gesicherte Änderungen am Text vorliegen.

adr1,adr2 **yank** *puffer zählung* Abkürzung: **y**

Der angegebene Zeilenbereich wird in *puffer* kopiert (für eine spätere Weiterverarbeitung mit put). Wird kein *puffer* angegeben, wird der Zeilenbereich in den Standardpuffer geschrieben.

z *zählung*

Gibt die nächsten *zählung* Zeilen auf dem Bildschirm aus.

!*unix-kommando*

Ausführung des angegebenen *unix-kommandos* in einer separaten Shell (Shell-Escape). Die Zeichen # und % werden wie in Dateinamen expandiert; das Zeichen ! wird durch den Kommandotext der letzten Shell-Escape ersetzt.

adr1,adr2 **!***unix-kommando*

Der angegebene Zeilenbereich wird als Standardeingabe des *unix-kommando*

verwendet; dessen Standardausgabe ersetzt den Zeilenbereich. Hier muß ausnahmsweise ein Adreßbereich angegeben werden.

`adr =`
Gibt die Nummer der adressierten Zeile auf dem Bildschirm aus.

adr1,adr2 **>***zählung flags*

adr1,adr2 **<***zählung flags*
Verschiebt die Zeilen im angegebenen Bereich nach links (<) bzw. rechts (>). Siehe die entsprechenden vi-Kommandos im Abschnitt 7.1.2.

adr1,adr2
Ausgabe der adressierten Zeilen auf dem Bildschirm.

adr1,adr2 **&** *optionen zählung flags*
Wiederholt das letzte substitute-Kommando.

adr1,adr2 **~** *optionen zählung flags*
Ersetzt den vorhergehenden regulären Ausdruck durch die Ersetzungsmaske des letzten substitute-Kommandos.

13 Tutorium

Eine gute Möglichkeit, den vi kennenzulernen bietet ein (englischsprachliches) Tutorium. Dieses führt an allen wichtigen vi-Kommandos vorbei, so daß Sie nach Bearbeiten dieses Tutoriums sicher sein können, die wesentlichen Möglichkeiten des vi zu beherrschen.

Sie können es am RRZN auf der UNICS und auf dem Workstation-Cluster aufrufen mit:

```
/opt/local/tutorium/vi-tutorial/vitutor
```

In Ihrem HOME-Verzeichnis wird eine Datei `tutor.vi` angelegt und der vi für diese Datei aufgerufen.

Es empfiehlt sich, das umfangreiche Tutorium nicht "in einem Rutsch", sondern in mehreren Abschnitten durchzuarbeiten.Wenn Sie sich die Stelle merken wollen, an der Sie die Arbeit vorläufig beenden, können Sie wie folgt vorgehen:

Mit `CTRL-G` lassen Sie sich anzeigen, in welcher Zeile Sie sich gerade befinden (im folgenden mit *`zeile`* bezeichnet) und verlassen den vi mit **`ZZ`**. Beim nächsten Mal rufen Sie mit

```
vi tutor.vi
```

die Datei wieder auf und springen mit

```
zeileG
```

an die Stelle, die Sie zuletzt bearbeitet haben.

Eine andere Möglickeit besteht darin, den Sprung auf die gewünschte Zeile beim Aufruf des vi anzufordern durch :

```
vi -c zeile+ tutor.vi
```

(siehe Abschnitt 10.1). Der Cursor steht dann in *`zeile`*`+1,` also am Beginn der ersten von Ihnen noch nicht bearbeiteten Zeile.

Viel Erfolg beim Bearbeiten des Tutoriums!