FH Schmalkalden
FB Elektrotechnik

Einführung in vi

Inhaltsverzeichnis

1 Einleitung

Das Programm „vi“ ist ein Texteditor, der auf den meisten UNIX-Betriebssystemen vorhanden ist.
Der vi kann sowohl im Terminalfenster als auch in einer Text-Konsole verwendet werden.
Der überwiegende Teil dieser Anleitung bezieht sich auf den Standard-vi und die Mehrzahl der gängigen vi-Clones, an einigen Stellen wird auf Features des Programmes „vim“ gesondert eingegangen.

2 Den vi starten

Das Programm wird durch Eingabe von

`vi` *Datei(en)*

gestartet (siehe Abb. 1).

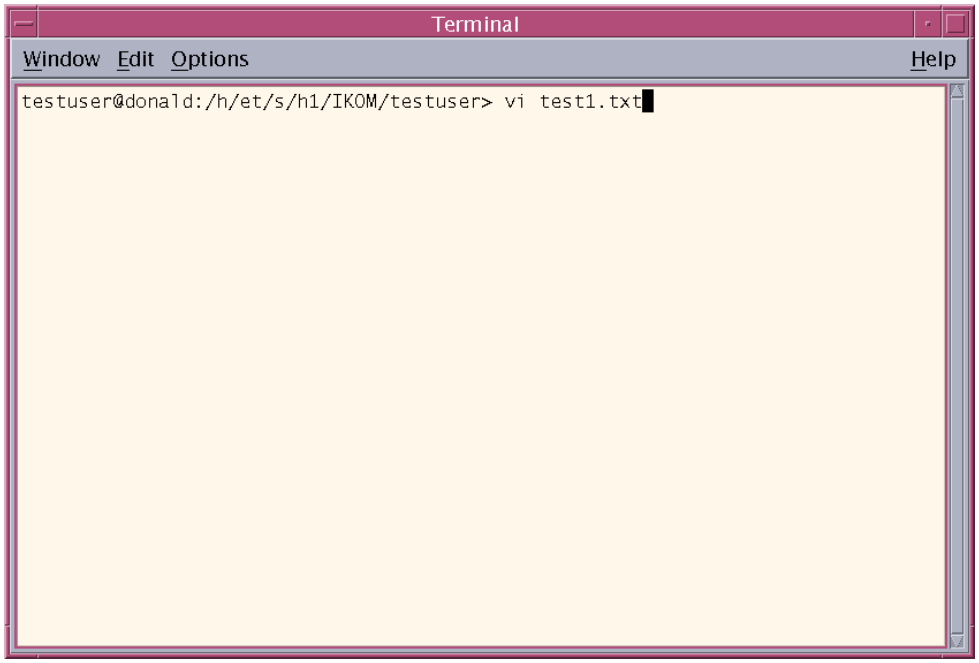

Abbildung 1: Aufruf des vi

Wurde mindestens ein Dateiname angegeben, wird die erste angegebene Datei geöffnet.
Andernfalls wird mit einem leeren Textpuffer gestartet wie in Abb. 2 zu sehen.

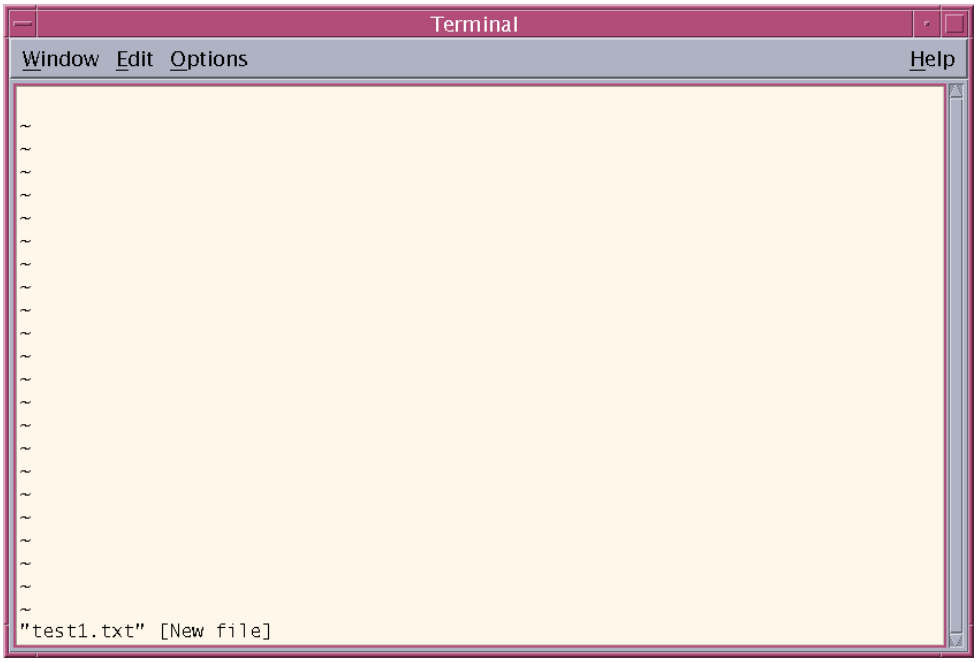

Abbildung 2: vi mit leerem Textpuffer

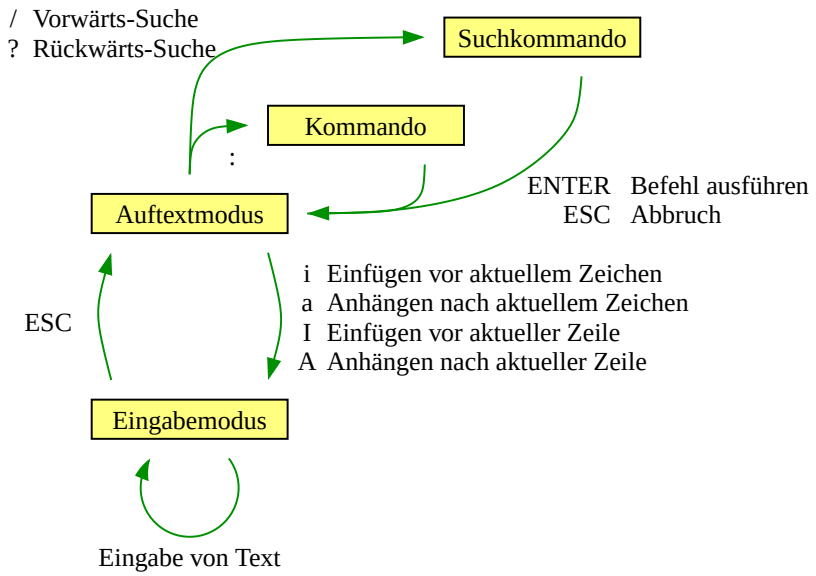

Abbildung 3: Arbeitsmodi des vi

3 Arbeitsweise

Der vi kennt verschiedene Arbeitsmodi:

- Auftextmodus.
  In diesem Modus befindet sich der vi unmittelbar nach dem Start.
  Mit den Cursortasten kann durch die Datei navigiert werden.

- Einfügemodus.
  Aus dem Auftextmodus gelangt man durch die Eingabe von `a`, `i`, `A`, `I` oder `o` in den Einfügemodus.

Zeichen	Bedeutung
a	Anhängen nach der aktuellen Cursor-Position
i	Einfügen vor der aktuellen Cursor-Position
A	Anhängen nach der aktuellen Zeile
I	Einfügen vor der aktuellen Zeile
o	Einfügen in einer neuen Zeile nach der aktuellen Zeile

  Wie der Name bereits sagt, werden in diesem Modus die eingegebenen Daten in den Textpuffer eingefügt.
  Einige Kontrollzeichen können im Eingabemodus nicht eingegeben werden, z.B. `CTRL-j`, `CTRL-q` und `CTRL-s`.
  Andere Kontrollzeichen, wie z.B. `CTRL-d` können eingegeben werden, unmittelbar vor der Eingabe des Kontrollzeichens muß `CTRL-v` vorangestellt wer-

den.
Durch Drücken der Esc-Taste gelangt man aus dem Eingabemodus zurück in den Auftextmodus.

- Kommandomodus.
  In den Kommandomodus gelangt man, indem man im Auftextmodus die Taste : drückt. Der Cursor springt in die letzte Bildschirmzeile und zeigt den Doppelpunkt an.
  Es kann jetzt ein vi-Kommando eingegeben werden.
  Wird die Eingabe mit ENTER abgeschlossen, wird das Kommando ausgeführt.
  Mit Esc gelangen Sie zurück in den Auftextmodus ohne daß ein Kommando ausgeführt wird.

- Suchmodus.
  Durch Eingabe von / bzw. ? gelangen Sie in den Suchmodus.
  Dabei steht / für eine Vorwärts-Suche (im Dokument nach unten gehen) und ? für eine Rückwärts-Suche (im Dokument nach oben gehen). Der Cursor springt in die letzte Zeile und zeigt den Slash bzw. das Fragezeichen an. Sie können nun das Suchmuster in Form eines regulären Ausdruckes angeben (siehe unten).
  Wenn die Eingabe des Suchmusters mit ENTER abgeschlossen wird, erfolgt die Suche, wenn das Suchmuster gefunden wurde, wird der Cursor am Beginn des Suchmusters positioniert.
  Mit Esc kommen Sie zurück in den Auftextmodus ohne eine Suche auszuführen.

4 Der Auftextmodus

4.1 Navigieren im Text

Um den Cursor im Text zu bewegen, können Sie die Cursortasten benutzen. Zusätzlich stehen folgende Tasten bzw. Tastenkombinationen zurVerfügung:

Taste	Bedeutung
CTRL-f	eine Seite vorwärts
CTRL-b	eine Seite zurück
^	zum Zeilenanfang gehen
0	zu erster Position in der Zeile gehen
$	zum Zeilenende gehen
G	zur letzten Zeile gehen
*n*G	zu Zeile Nummer *n* gehen
{	zur nächstgelegenen Leerzeile weiter oben im Text gehen
}	zur nächstgelegenen Leerzeile weiter unten im Text gehen
zENTER	aktuelle Zeile wird erste Zeile im Fenster
z.	aktuelle Zeile wird mittlere Zeile im Fenster
z-	aktuelle Zeile wird unterste Zeile im Fenster
*n*zENTER	macht die n-te Zeile des Textes zur ersten Zeile des Fensters
*n*z.	macht die n-te Zeile des Textes zur mittleren Zeile des Fensters
*n*z-	macht die n-te Zeile des Textes zur untersten Zeile des Fensters

4.2 Text suchen

4.2.1 Neuen Suchvorgang beginnen

Durch Drücken der Taste / bzw. ? gelangt man in denSuchmodus.
Der Cursor springt in die letzte Zeile und zeigt / bzw. ? an. Es kann jetzt ein Suchmuster eingegeben werden. Mit ENTER wird die Suchmuster-Eingabe beendet und die Suche gestartet. Wird eine passende Textstelle gefunden, wird der Cursor am Anfang der Textstelle positioniert.
Der Slash veranlaßt dabei eine Suche vorwärts (im Text nach unten gehend), das Fragezeichen eine Suche rückwärts (im Text nach oben gehend).
Das Suchmuster kann als einfacher regulärer Ausdruck angegeben werden, siehe Abschnitt 7 auf Seite 60. Im Beispiel in Abb. 4 wird nach einer beliebig langen Folge von Buchstaben (sowohl Klein- als auch Großbuchstaben), die mit „tor“ abschließt, gesucht.

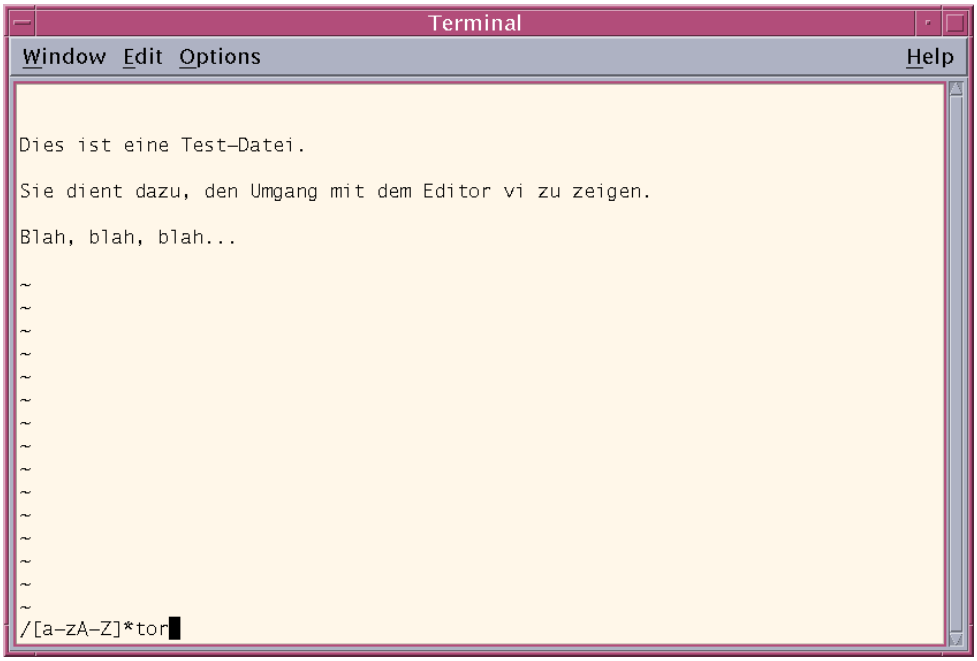

Abbildung 4: Suchmuster eingeben

4.2.2 Letzten Suchvorgang wiederholen

Durch Drücken der Taste n wird der letzte Suchvorgang wiederholt.

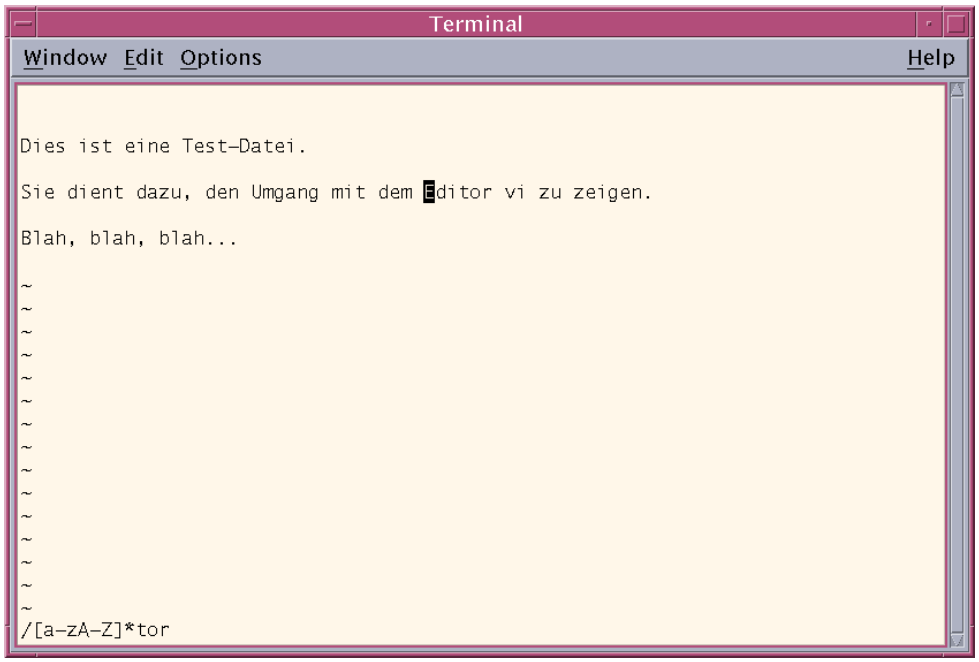

Abbildung 5: Cursor an passender Textstelle positioniert

4.3 Text-Markierungen

4.3.1 Markierung setzen

Im Text können Markierungen gesetzt werden, die mit den Kleinbuchstaben „a“. . . „z“ benannt werden können.
Durch Eingabe von `ma` wird die Markierung *a* auf die aktuelle Cursorposition gesetzt. Mit `mb` wird die Markierung *b* gesetzt. . .

4.3.2 Zu einer Markierung springen

Durch Eingabe von `` `a `` wird zu Markierung *a* gesprungen, mit `` `b `` zu Markierung *b*. . .

4.4 Kopierpuffer

Zum Kopieren/Ausschneiden und anschließenden Einfügen stehen Kopierpuffer zur Verfügung, die mit Kleinbuchstaben „a“. . . „z“ benannt werden. Um eine Textstelle in einen Kopierpuffer zu bringen, wird zunächst der Cursor auf das erste Zeichen nach der Textstelle gesetzt. Auf diese Position setzt man eine Markierung, indem man beispielsweise `ma` eingibt. Anschließend wird der Cursor auf das erste Zeichen der Textstelle gesetzt. Mit `¨ay`a` wird dann der Kopierpuffer *a* (`¨a`) mit Text gefüllt (`y` - yank) und zwar von der aktuellen Cursorposition bis zur Markierung *a* (`` `a ``).
Um einen Kopierpuffer an der aktuellen Cursorposition einzufügen (beispielsweise den Kopierpuffer *a*) kann `¨ap` genutzt werden.
Allgemein haben die Kommandos für den Umgang mit Kopierpuffern die Form ¨, Puffername (Kleinbuchstabe), Aktion. Wird für die Aktion ein `y` (yank) eingegeben, muß das Bereichsende definiert werden.
Dies kann auf unterschiedliche Weise erfolgen, u.a.:

- bis zu einer Markierung (`` `a ``)
- bis zu einer Zeile, die ein bestimmtes Suchmuster enthält (`/suchtext`)
- Angabe einer Zeilenzahl.

4.5 Text löschen

4.5.1 Aktuelles Zeichen löschen

Durch Drücken der Taste `x` wird das aktuelle Zeichen (d.h. das Zeichen, auf dem der Cursor sich gerade befindet) gelöscht.

4.5.2 Aktuelle Zeile löschen

Durch Eingabe von `dd` (zweimaliges Drücken der Taste `d`) wird die aktuelle Zeile (die Zeile, in der der Cursor sich gerade befindet) gelöscht.

4.5.3 Zeilenende löschen

Mit `D` wird der Text von der aktuellen Cursorposition bis zum Zeilenende gelöscht.

4.5.4 Weitere Löschoperationen

Mit `d` und anschließender Angabe einer Endadresse (z.B. Markierung, Suchmuster, Zeilenzahl) sind ebenfalls Löschoperationen möglich.

5 vi-Kommandos

Die folgenden Kommandos können im Kommandomodus eingesetzt werden, d.h. es muß mit Eingabe eines Doppelpunktes vom Auftextmodus in den Kommandomodus gewechselt werden, bevor das Kommando eingegeben wird.

5.1 Text speichern

5.1.1 Datei speichern

Das Kommando

```
w
```

veranlasst, daß die Datei unter ihrem aktuellen Namen gespeichert wird (siehe Abb. 6).

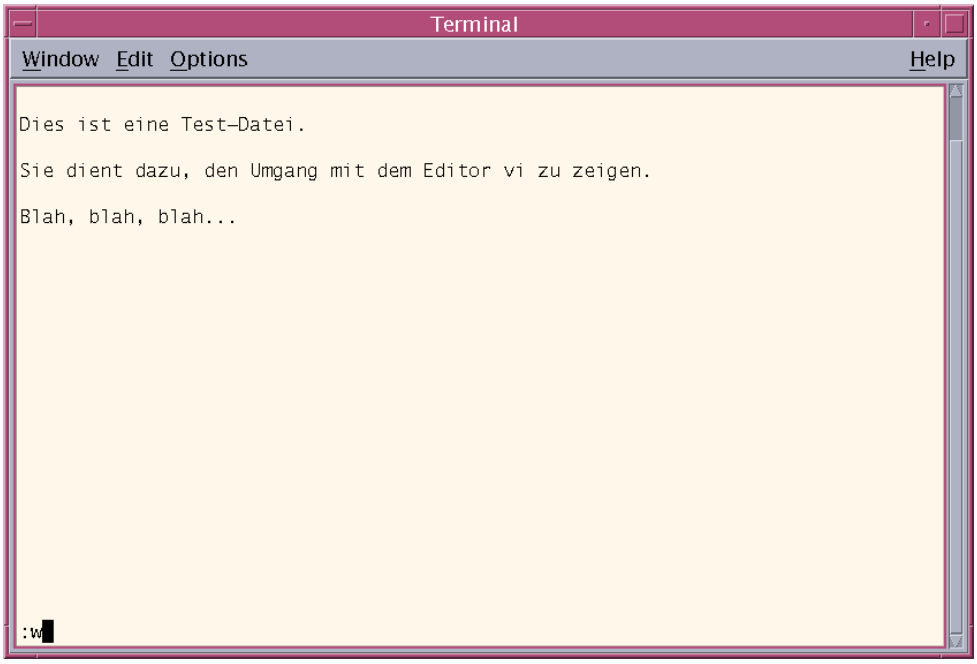

Abbildung 6: Datei speichern

5.1.2 Datei unter bestimmten Namen speichern

Mit

`w` *Dateiname*

wird die Datei unter dem vorgegebenen Namen gespeichert.

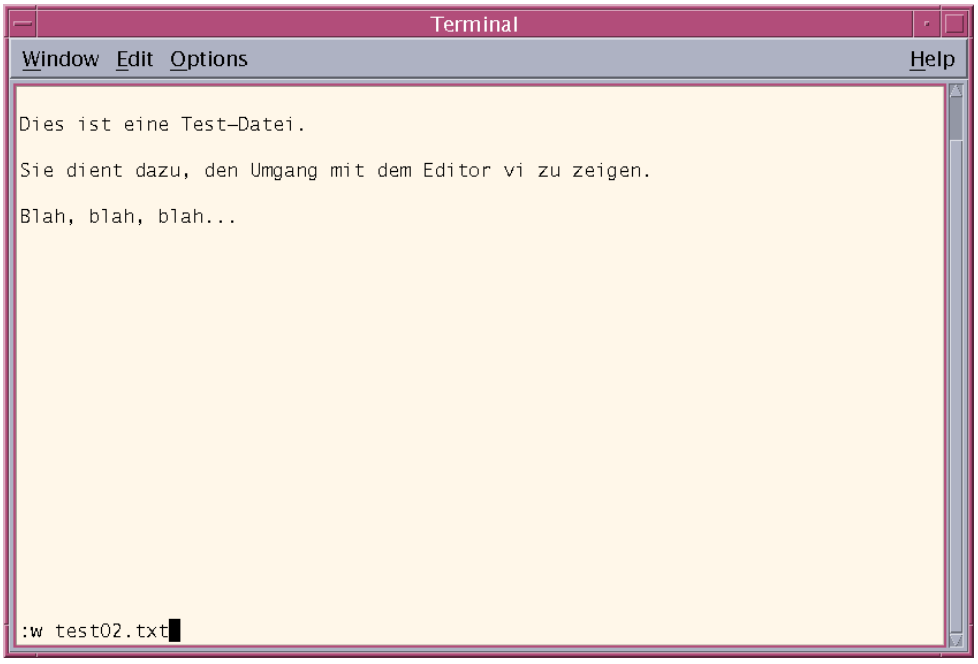

Abbildung 7: Datei unter anderem Namen speichern

5.1.3 Letzten gespeicherten Zustand wiederherstellen

Mit

```
e!
```

wird der letzte gespeicherte Zustand der Datei erneut geladen.

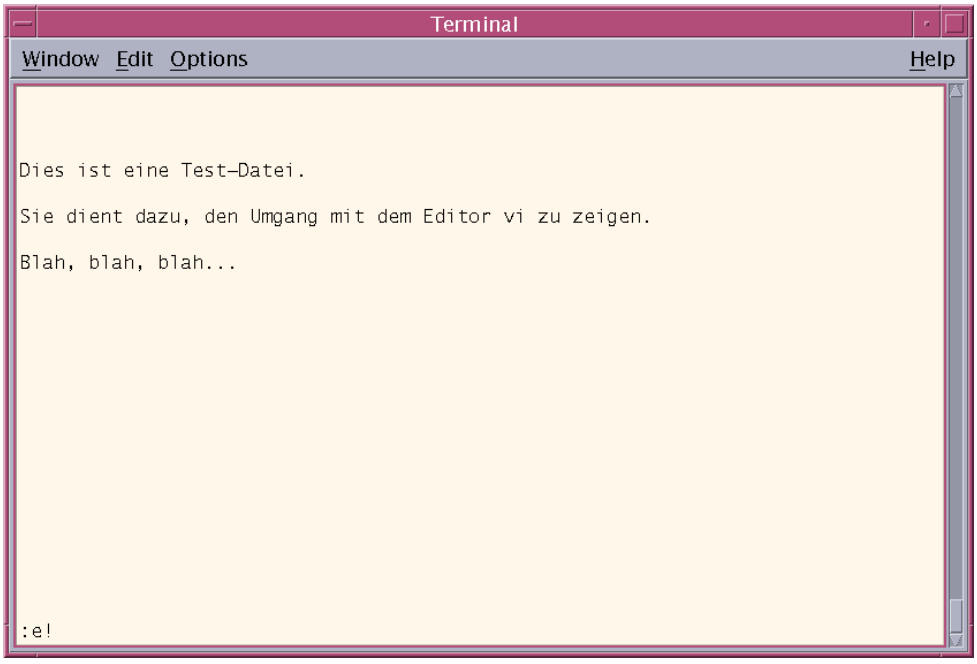

Abbildung 8: Letzten gespeicherten Zustand wiederherstellen

5.2 vi beenden

Mit

q

kann der vi verlassen werden, wenn die letzten Änderungen bereits gespeichert worden sind. Andernfalls erfolgt eine Fehlermeldung.

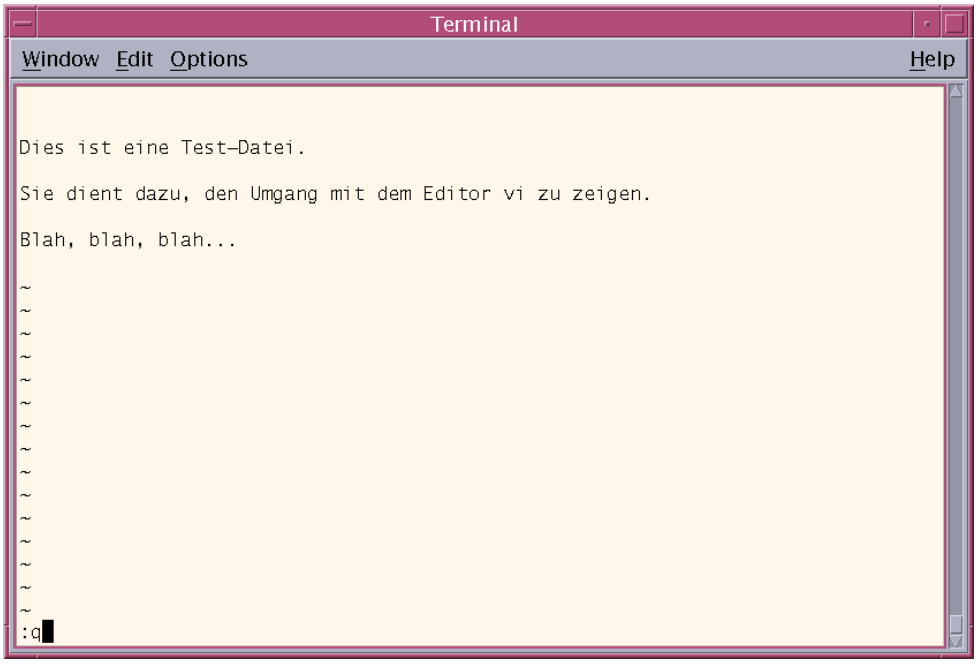

Abbildung 9: Editor verlassen

Mit

```
wq
```

wird die gerade in Bearbeitung befindliche Datei gespeichert und der Editor verlassen.

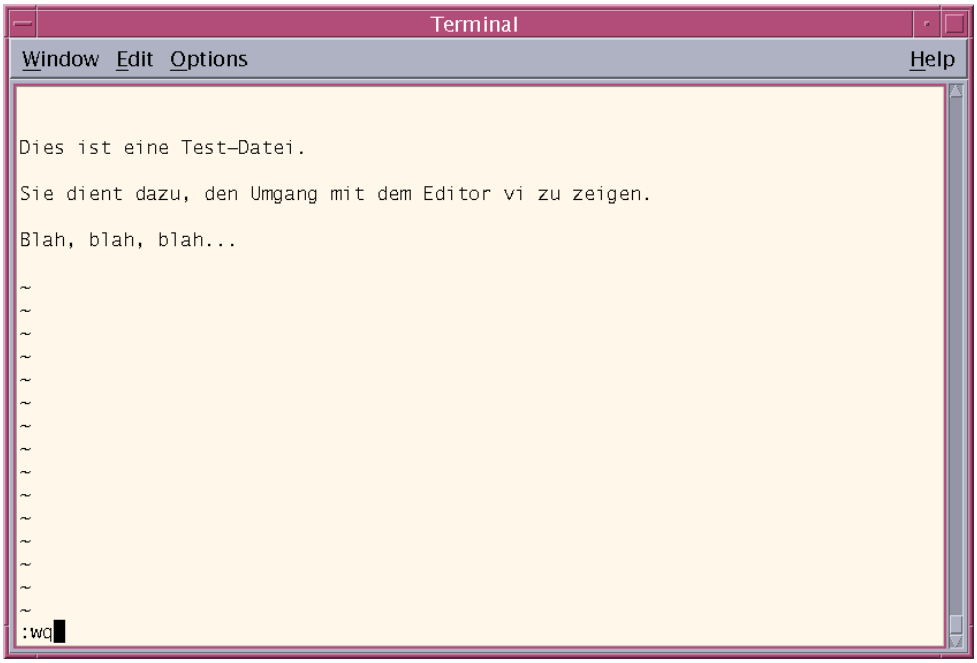

Abbildung 10: Datei speichern, Editor verlassen

Mit

```
q!
```

wird der Editor verlassen, eventuell vorhandene noch nicht gesicherte Änderungen werden verworfen.

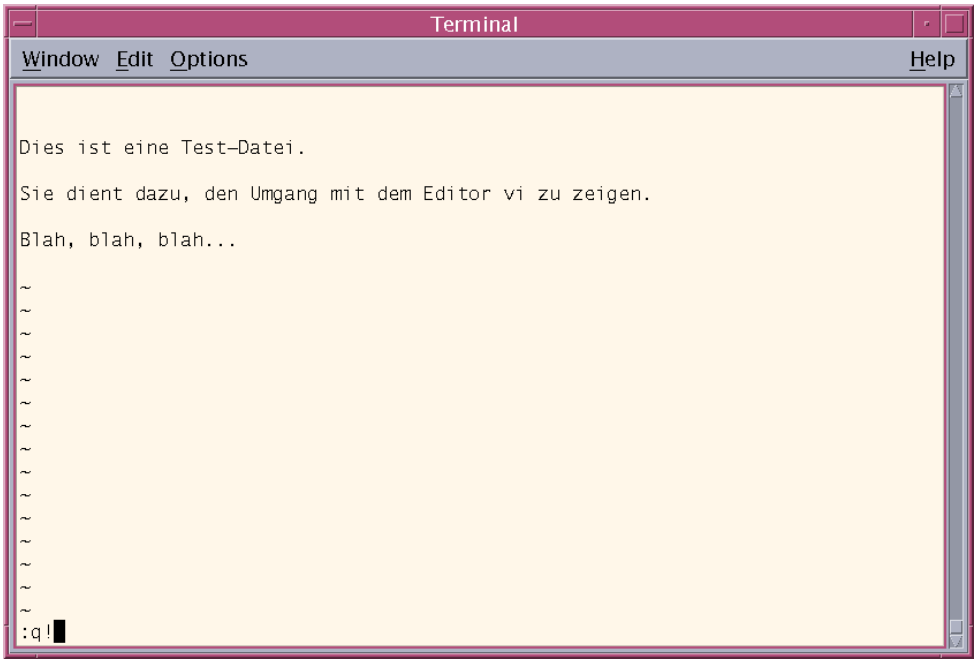

Abbildung 11: Editor verlassen, Änderungen verwerfen

5.3 Datei öffnen

5.3.1 Nächste Datei bearbeiten

Wurden beim Aufruf des Programmes mehrere Dateinamen angegeben, wird mit

n

die zur nächsten Datei übergegangen.

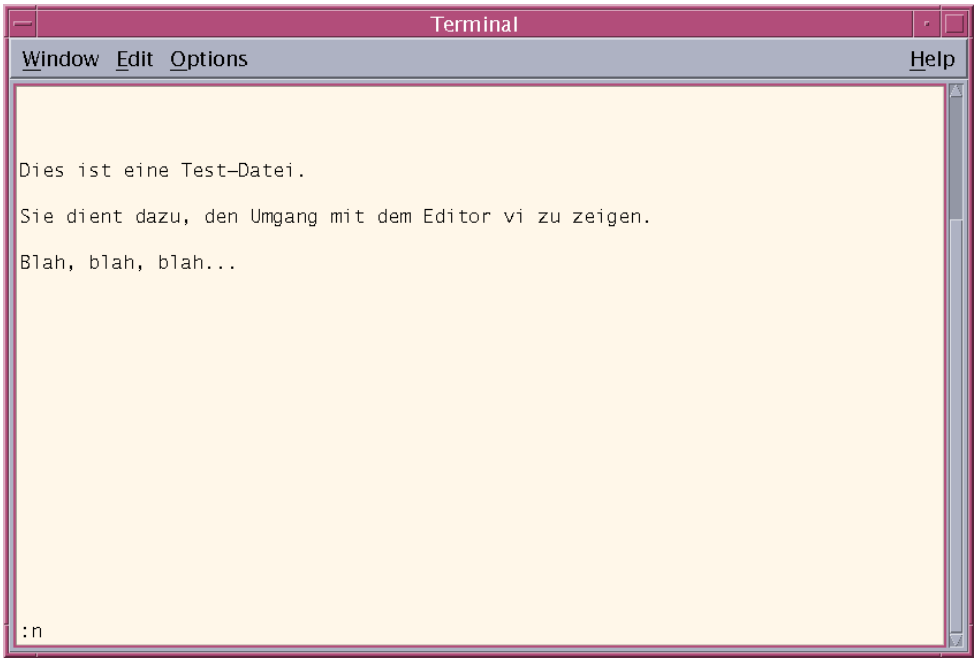

Abbildung 12: Zur nächsten Datei übergehen

Bevor zu einer anderen Datei gewechselt werden kann, muß die aktuelle Datei gespeichert werden, andernfalls wird eine Fehlermeldung ausgegeben und der Dateiwechsel nicht ausgeführt.

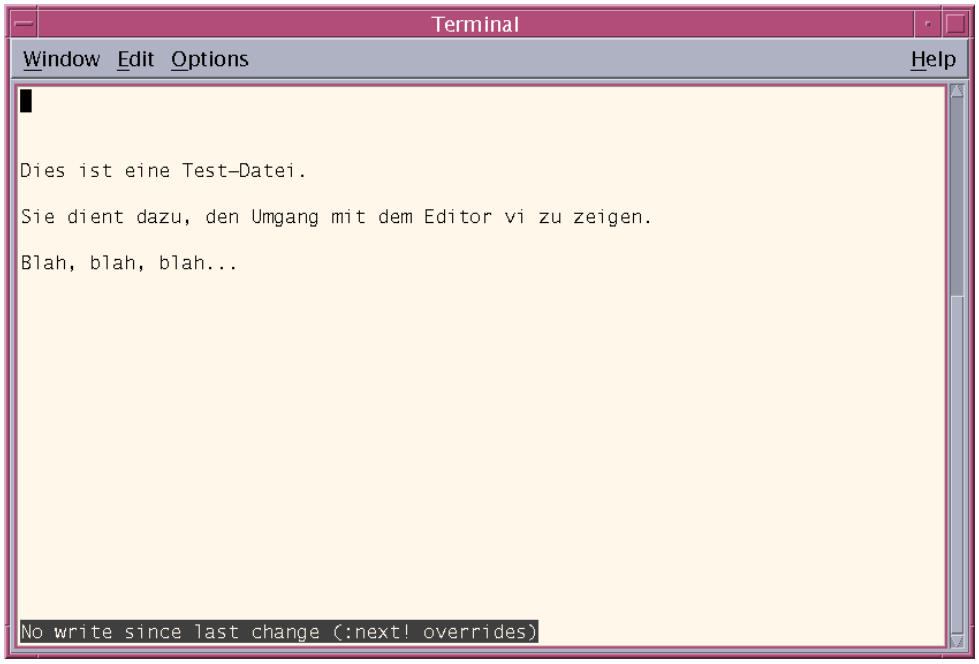

Abbildung 13: Fehlermeldung wegen nicht abgespeicherten Änderungen

Mit

```
n!
```

wird unbedingt zur nächsten Datei übergegangen, dabei werden eventuelle Änderungen, die nach dem letzten Speichern an der aktuellen Datei vorgenommen wurden, verworfen.

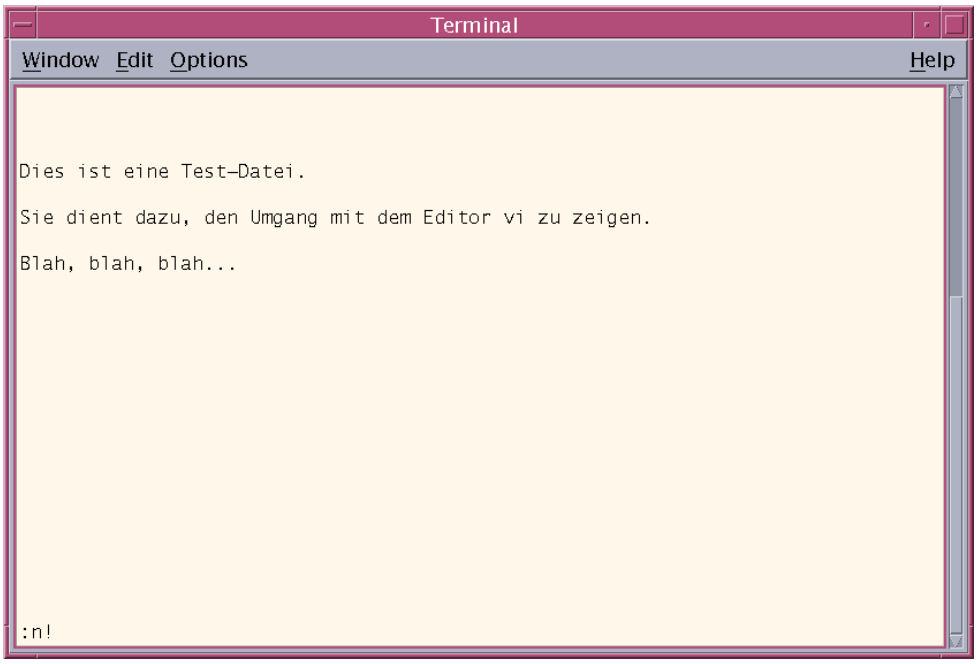

Abbildung 14: Änderungen verwerfen, zur nächsten Datei wechseln

5.3.2 Vorangegangene Datei nochmals bearbeiten

Mit

e#

wird die vorangegangene Datei nochmals bearbeitet.

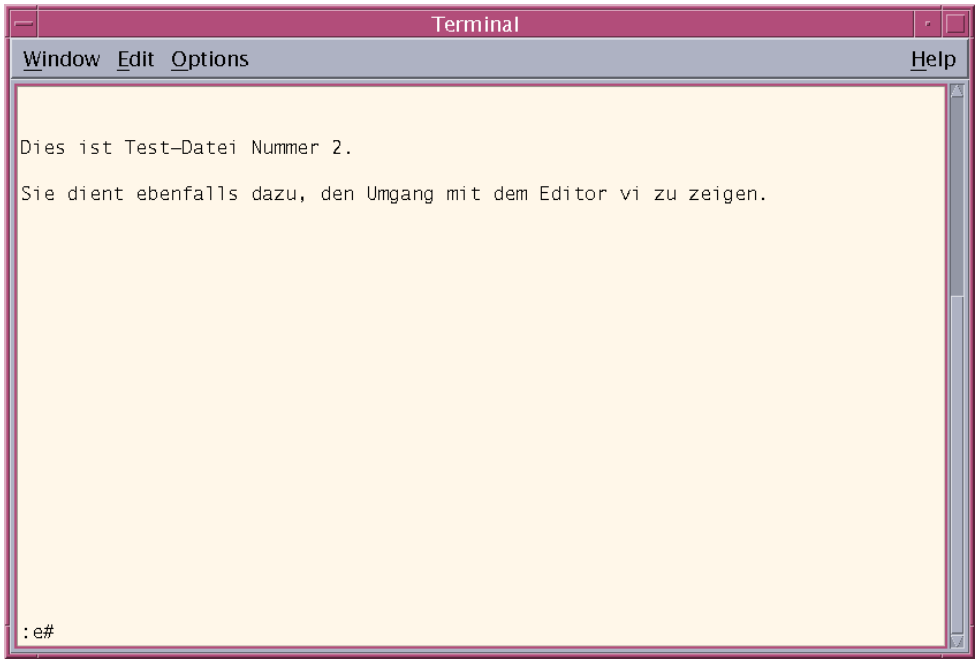

Abbildung 15: Zurück zur vorangegangenen Datei

Mit

e!#

wird der Test, ob ein Speichern der aktuellen Datei nötig ist, unterdrückt.

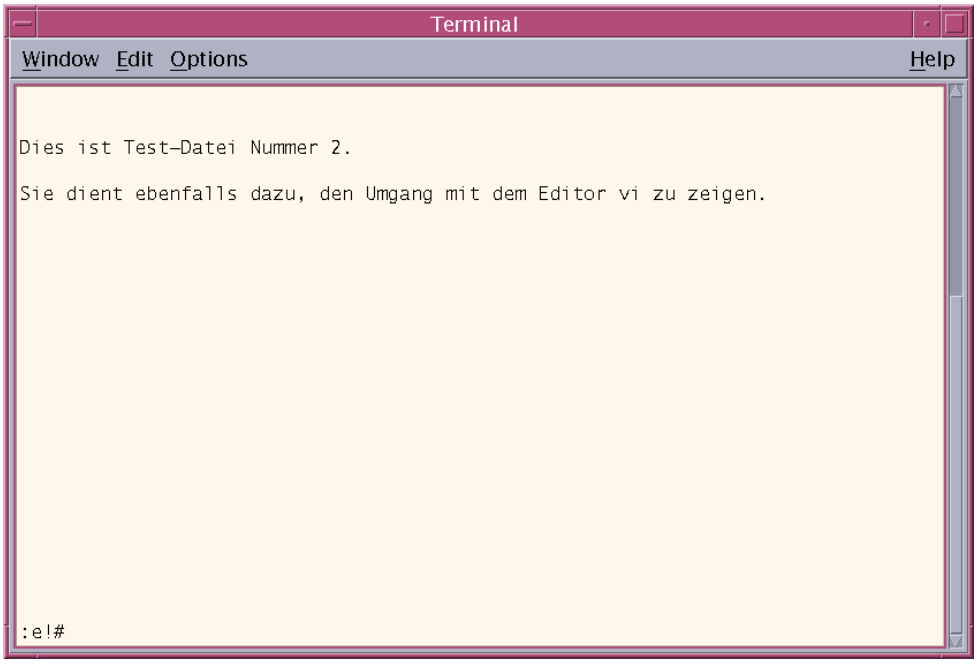

Abbildung 16: Änderungen verwerfen, zurück zur vorherigen Datei

5.3.3 Andere Datei öffnen

Mit

e *Datei*

wird die angegebene Datei bearbeitet.

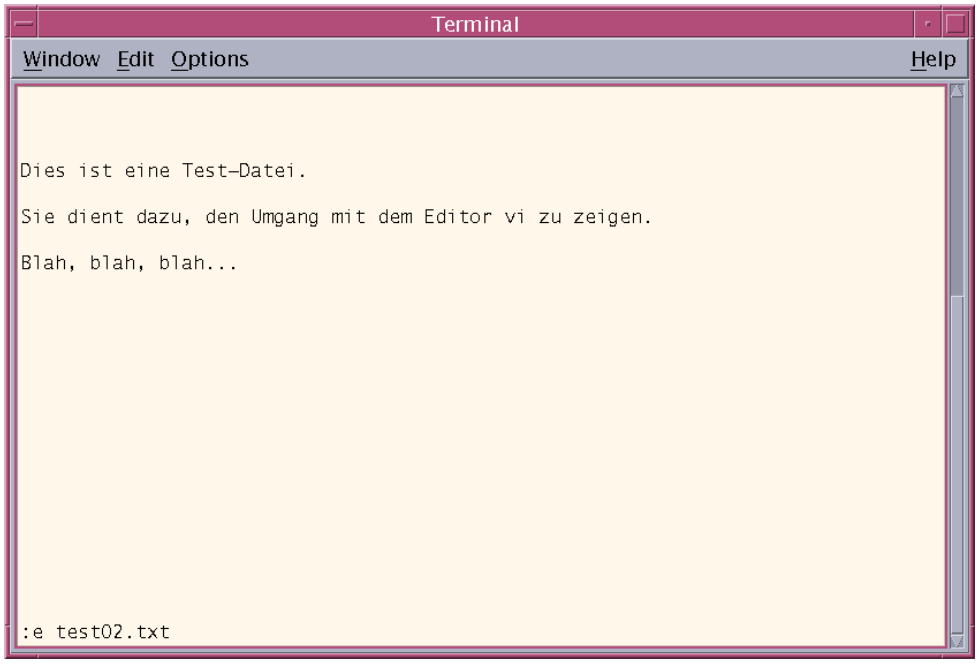

Abbildung 17: Andere Datei öffnen

Mit

e! *Datei*

wird der Test, ob ein Speichern der aktuellen Datei nötig ist, unterdrückt.

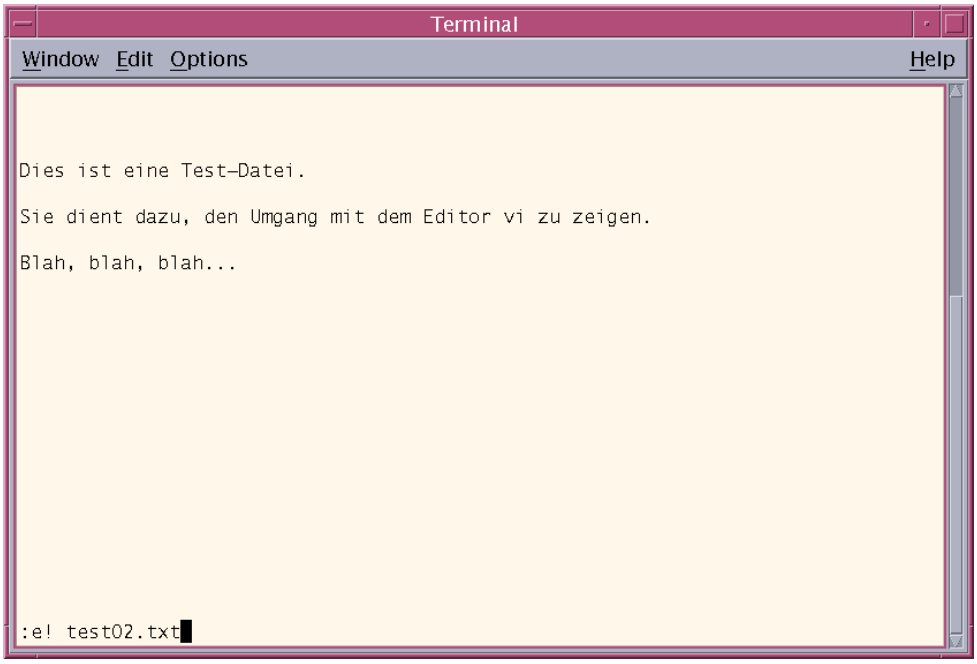

Abbildung 18: Änderungen verwerfen, andere Datei öffnen

5.4 Text importieren

5.4.1 Datei einfügen

Das Kommando

r *Datei*

fügt den Inhalt der angegebenen Datei an der aktuellen Cursorposition ein.

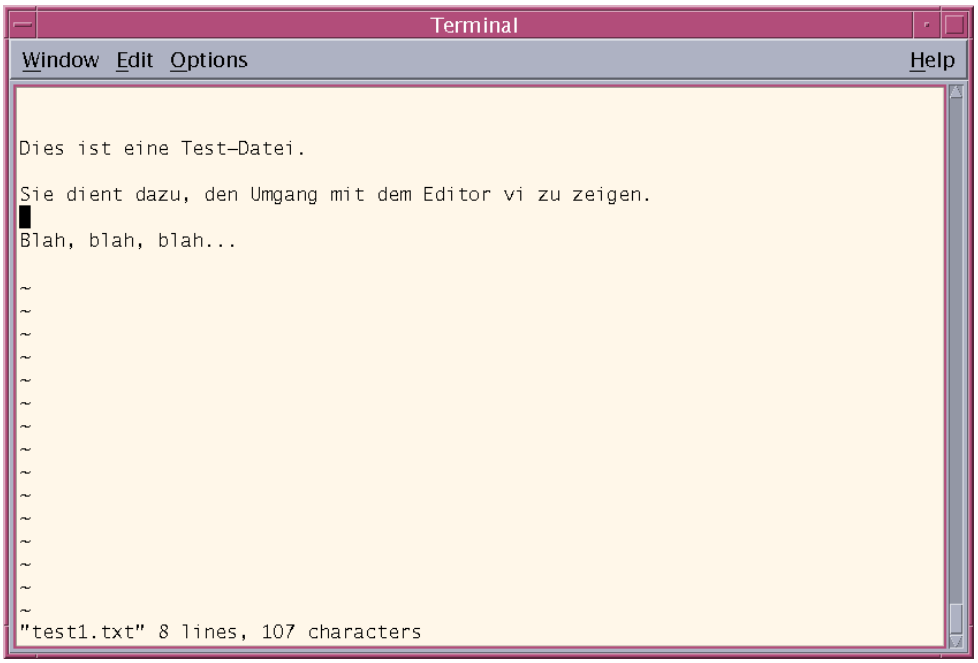

Abbildung 19: Datei vor Einfügen einer anderen Datei

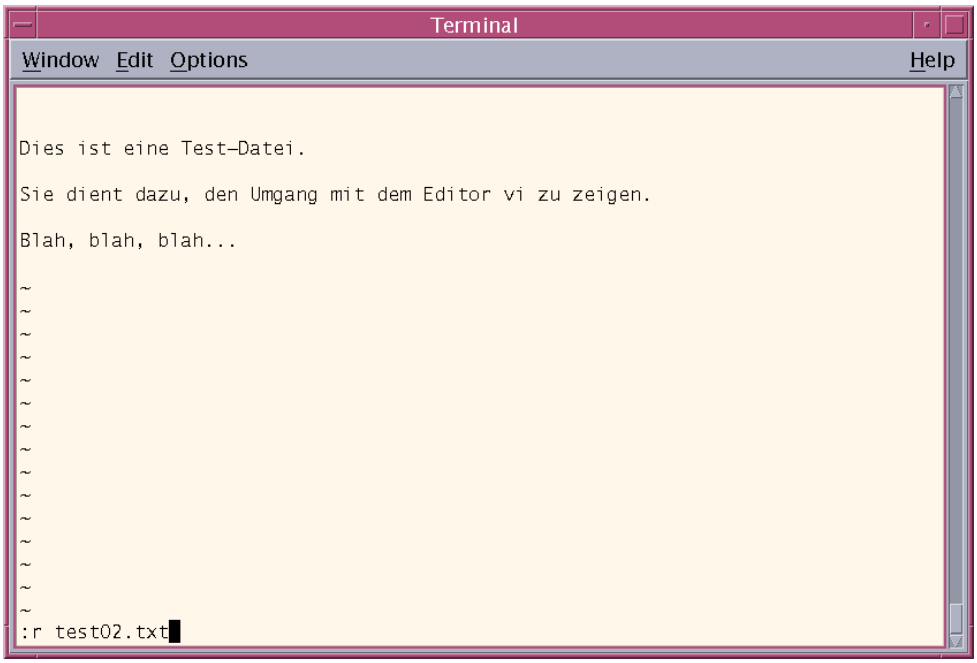

Abbildung 20: Einfügen einer anderen Datei an aktuelle Position

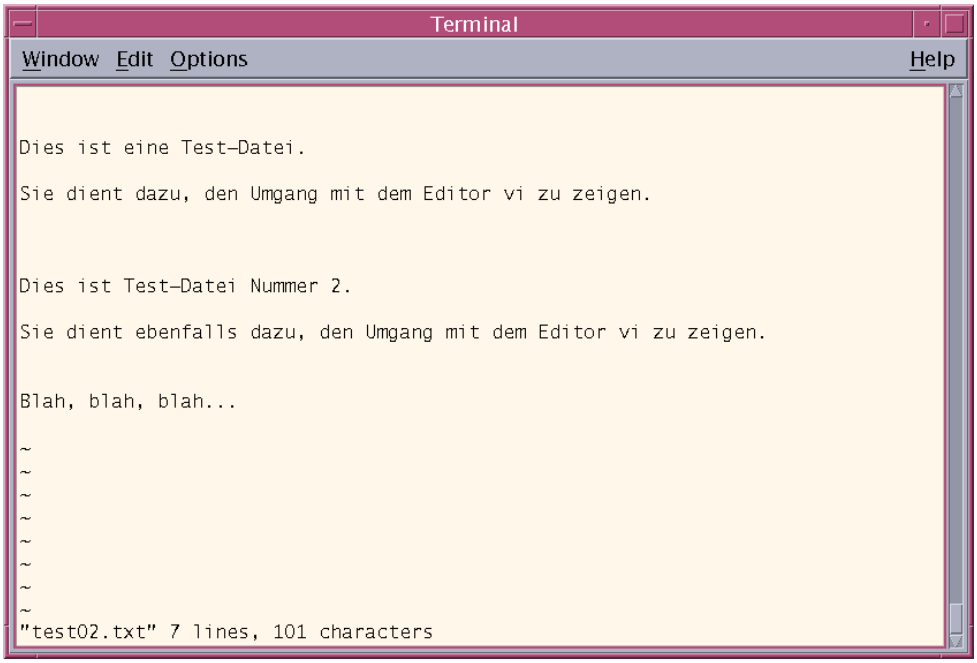

Abbildung 21: Datei nach dem Einfügen

5.4.2 UNIX-Befehl ausführen

Mit

! *Befehl*

wird der genannte UNIX-Befehl ausgeführt.
Nachdem der Befehl ausgeführt wurde muß ENTER gedrückt werden, um mit dem Bearbeiten der Datei fortzufahren.
Im Beispiel wird das Programm ls gestartet, um die Dateien im aktuellen Verzeichnis anzuzeigen.

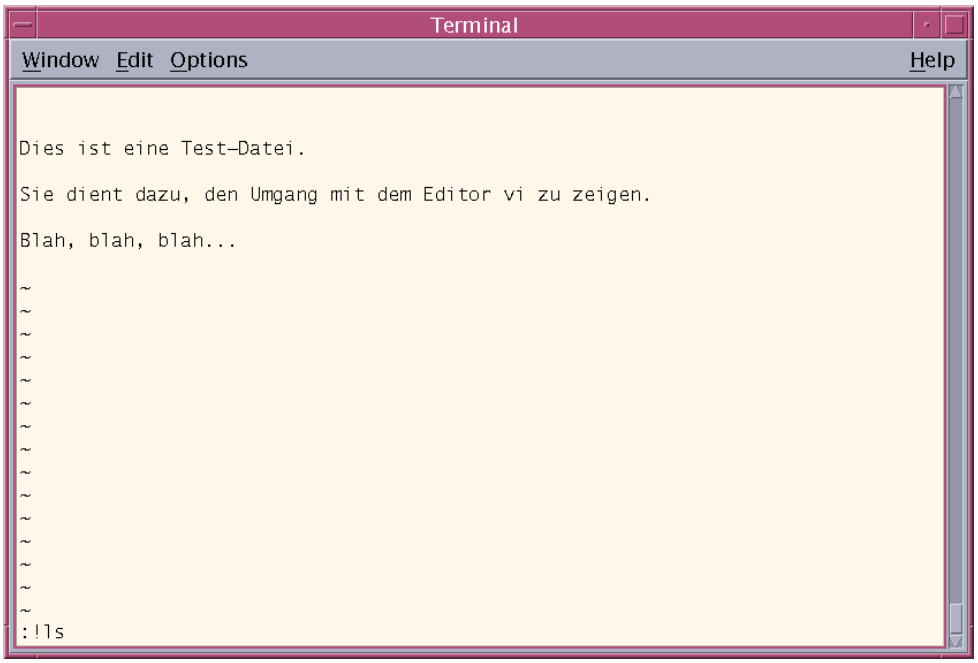

Abbildung 22: UNIX-Befehl ausführen

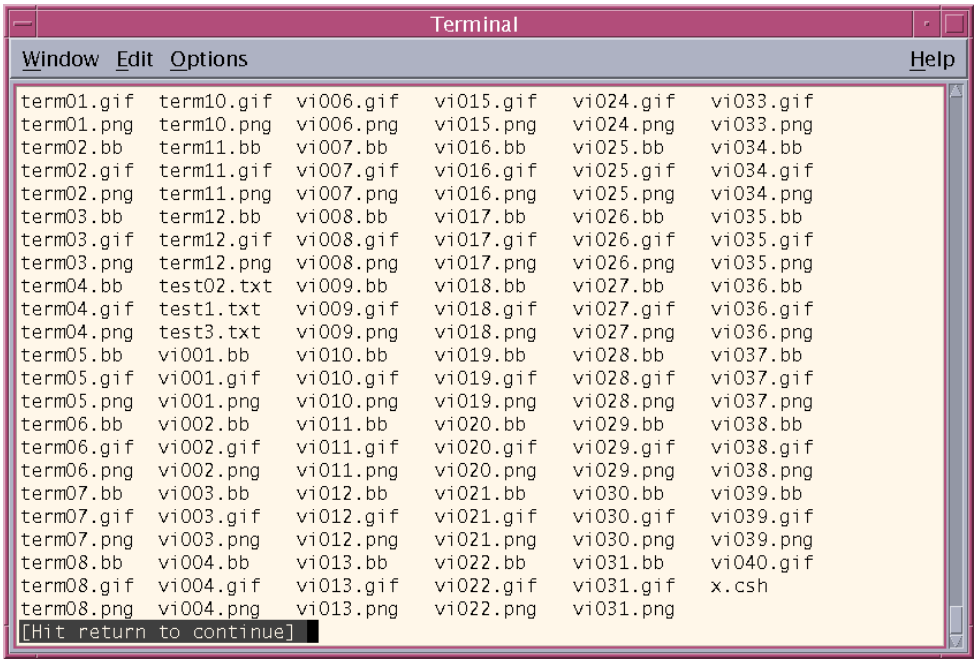

Abbildung 23: Nach Ausführung eines UNIX-Befehles

5.4.3 Ausgabe eines UNIX-Befehles einlesen

Soll die Ausgabe eines UNIX-Befehles an der aktuellen Cursorposition eingefügt werden, erreicht man dies mit

`r !` *Befehl*

Im Beispiel wird die Ausgabe des Programmes date eingelesen, das die aktuelle Zeit liefert.

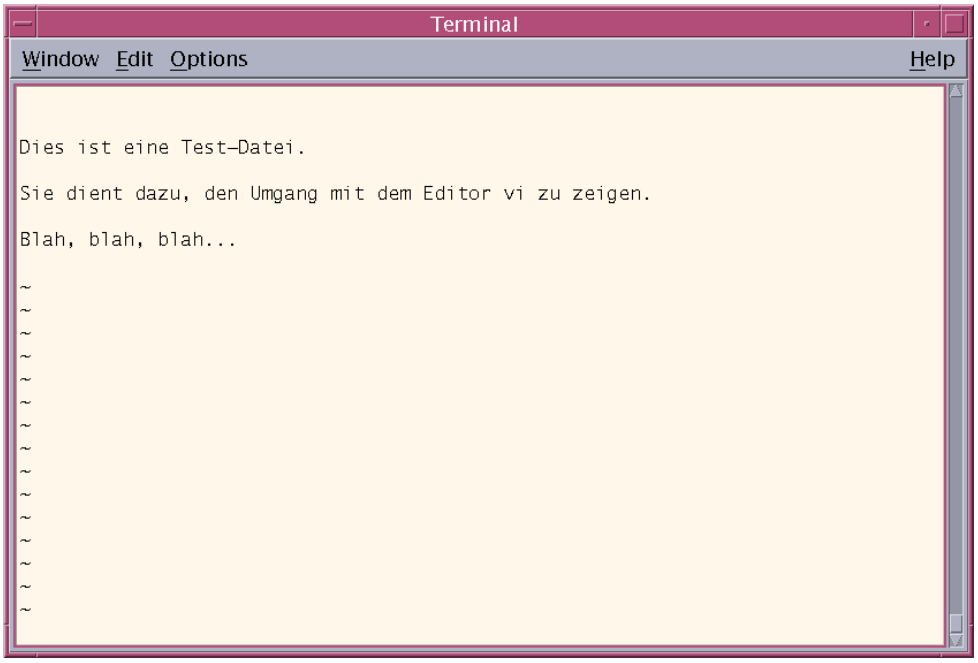

Abbildung 24: Datei vor Einlesen der date-Ausgabe

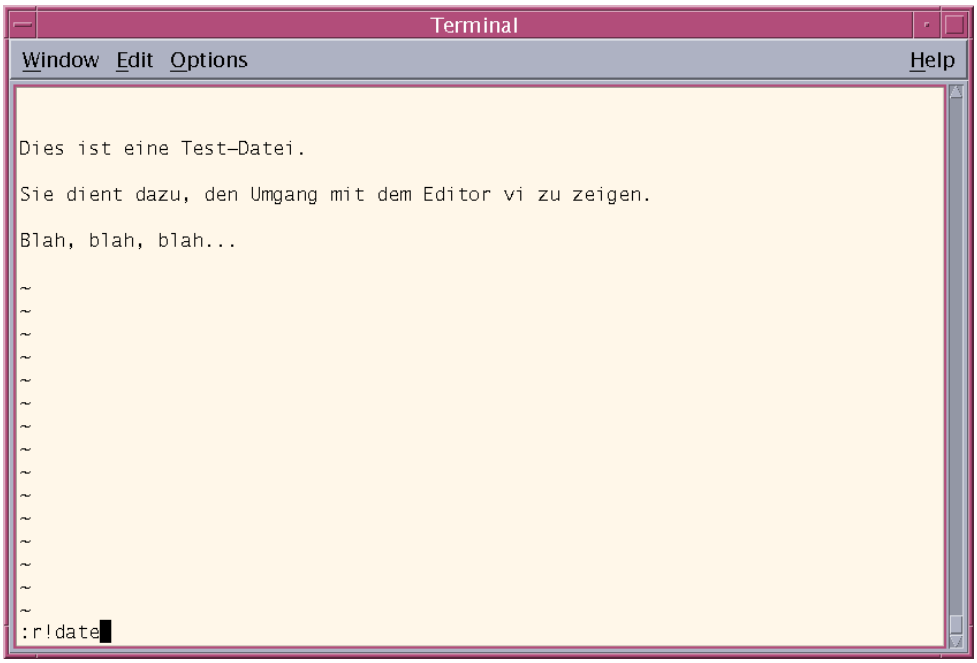

Abbildung 25: Der Einlese-Befehl

Abbildung 26: Datei nach Einlesen der date-Ausgabe

5.5 Text löschen

Um einen Textbereicht zu löschen, wird das Kommando *Bereich*`d` genutzt.
Der Bereich wird entweder durch Angabe einer Zeile oder durch Angabe von Anfangs- und Endzeile ausgewählt. Wird kein Bereich angegeben, wird die aktuelle Zeile gelöscht.
Werden Anfangs- und Endzeile angegeben, werden die beiden Adressen durch ein Komma getrennt.
Zeilen können durch Zeilennummern angegeben werden, wobei zusätzlich der Punkt (`.`) für die aktuelle Zeile und das Dollar-Zeichen (`$`) für die letzte Zeile verwendet werden kann.
Alternativ dazu können die Zeilen auch über Suchmuster ausgewählt werden, die in Slashes (`/`) eingeschlossen sind.
Beispiele:

Löschbefehl	Bedeutung
`%d`	löscht alle Zeilen
`3,5d`	löscht die Zeilen 3 bis 5
`1,.d`	löscht von der ersten Zeile bis zur aktuellen Zeile
`22,$d`	löscht von Zeile 22 bis einschließlich zur letzten Zeile
`/bbb/,/hhh/d`	löscht von der ersten Zeile nach der aktuellen, die `bbb` enthält bis einschließlich zur ersten Zeile, die `hhh` enthält.

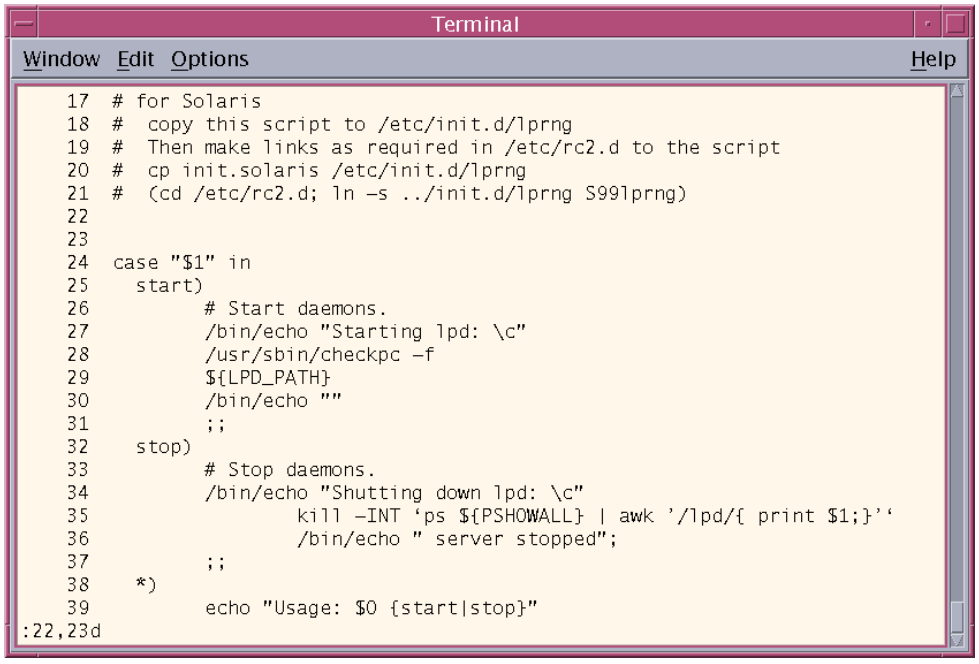

Abbildung 27: Löschen von Zeile 22 bis Zeile 23

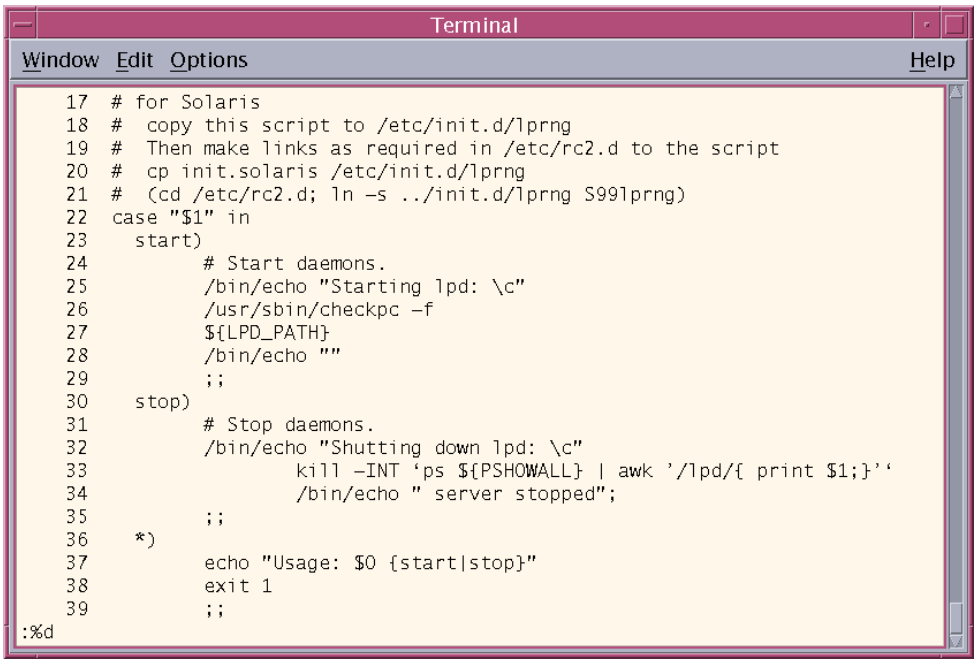

Abbildung 28: Alle Zeilen löschen

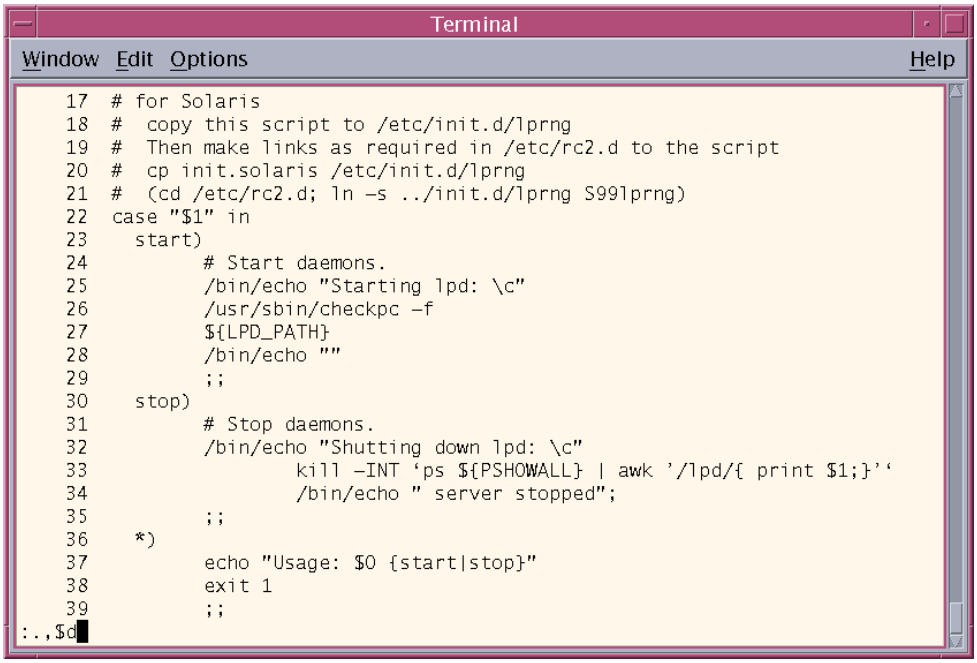

Abbildung 29: Von der aktuellen Zeile bis zur letzten Zeile löschen

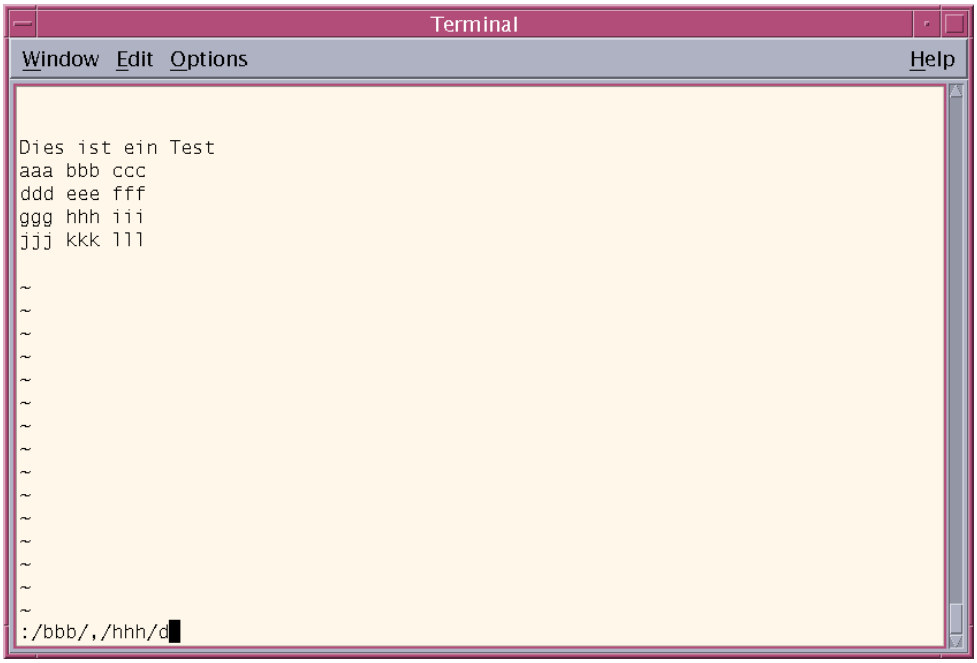

Abbildung 30: Von der ersten Zeile mit bbb bis zur ersten Zeile mit hhh löschen

5.6 Text ersetzen

Mit dem Befehl *Bereich*`s/`*alt*`/`*neu* (substitute) können Textmuster durch neuen Text ersetzt werden.
Der Bereich, in dem die Ersetzungen vorgenommen werden sollen, kann wie auch beim Löschen durch Zeilennummern oder Suchausdrücke angegeben werden.
Der zu ersetzende Text (*alt*) kann als regulärer Ausdruck (siehe 7 auf Seite 60) angegeben werden. Durch einen Slash vom alten Text getrennt, schließt sich der neue Text (*neu*) an.
Der oben gezeigte Befehl führt die Ersetzung in jeder in Frage kommenden Zeile nur für das erste gefundene Suchmuster aus.
Mit *Bereich*`s/`*alt*`/`*neu*`/g` werden in den ausgewählten Zeilen alle Vorkommen des Suchmusters (global) ersetzt.

5.7 Makros

5.7.1 Befehlsmakros

Befehlsmakros werden mit

`map` *#name Eingabe*

definiert.
Wird über die Tastatur der Name eines Befehlsmakros eingegeben, verhält sich vi, als hätte er statt dessen die angegebenen Eingaben erhalten.
Es wird empfohlen, daß die Namen der Befehlsmakros mit # beginnen, da der Großteil aller anderen Tasten im Auftextmodus schon anderweitig belegt ist.
Das Beispielmakro

```
map #x I <CTRL-v><Esc><CTRL-v><Enter>
```

bewirkt, daß bei Eingabe von `#x` die aktuelle Zeile um zwei Leerzeichen eingerückt wird und der Cursor in der nächsten Zeile positioniert wird.

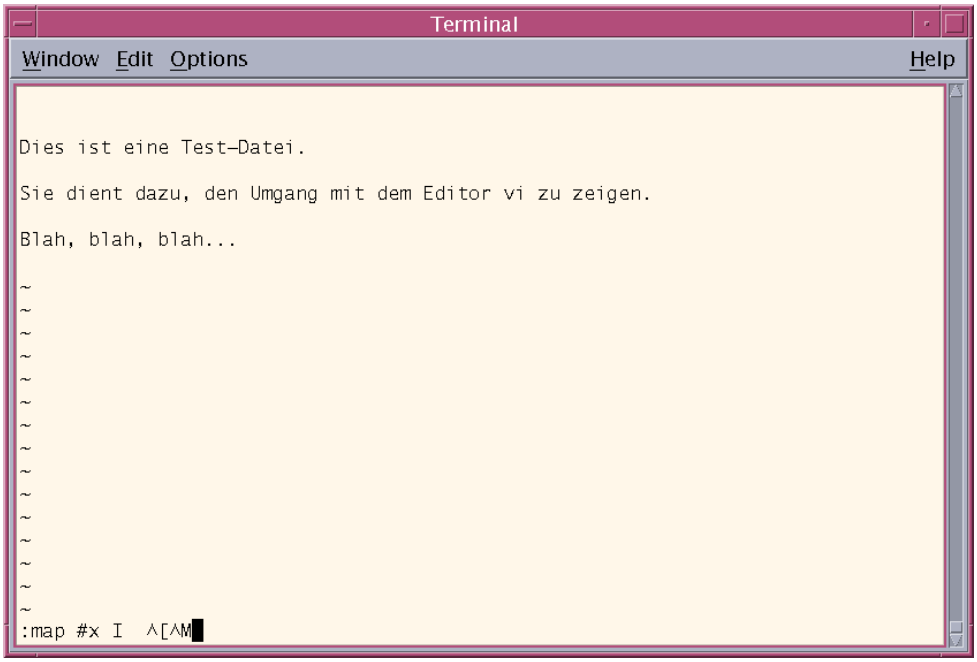

Abbildung 31: Befehlsmakro definieren

5.7.2 Textmakros

Bei Textmakros handelt es sich um Abkürzungen. Immer wenn eine solche Abkürzung für sich allein eingegeben wird (d.h. ein Leerzeichen vor der Abkürzung, eines danach) wird die Abkürzung durch den vollen Text ersetzt. Mit

```
ab HSDS Heute scheint die Sonne.
```

wird eine Abkürzung `HSDS` mit dem Volltext
„Heute scheint die Sonne.“
definiert.

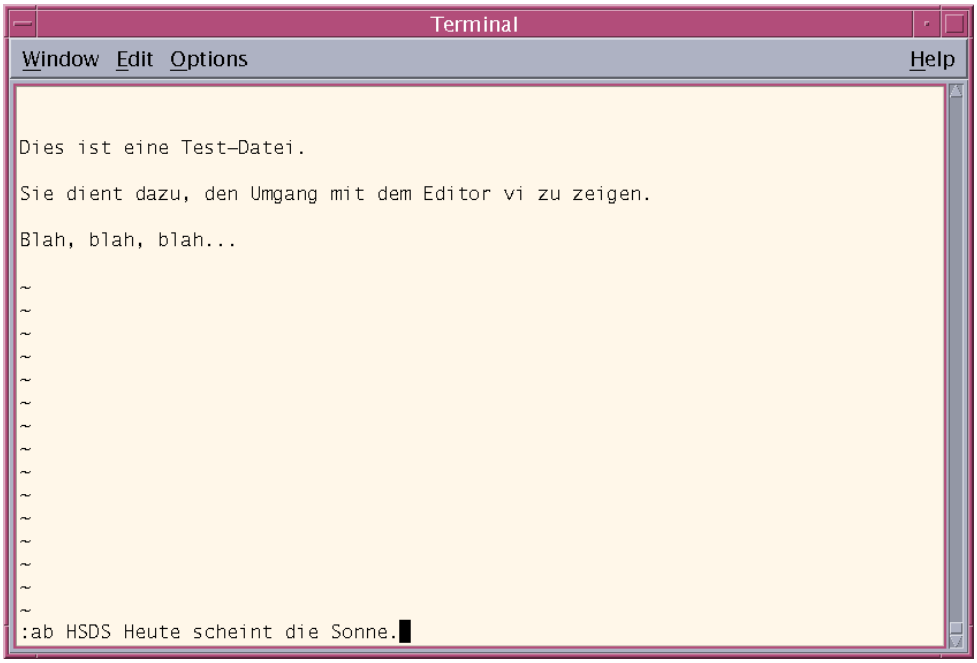

Abbildung 32: Abkürzung definieren

Mit

```
unab HSDS
```

wird die Abkürzung wieder aufgehoben.

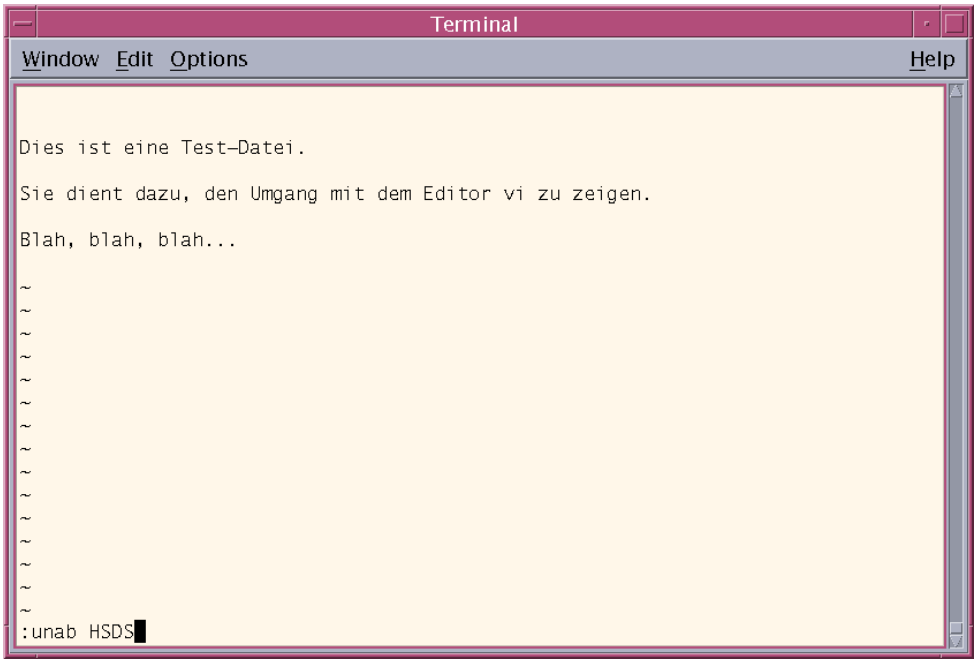

Abbildung 33: Abkürzung wieder aufheben

5.8 Einstellungen festlegen

5.8.1 Groß- und Kleinschreibung beim Suchen

```
set ignorecase
```

bewirkt, daß bei allen nachfolgenden Suchoperationen keine Unterscheidung zwischen Groß- und Kleinschreibung erfolgt. Abb. 35 auf der nächsten Seite zeigt, wie

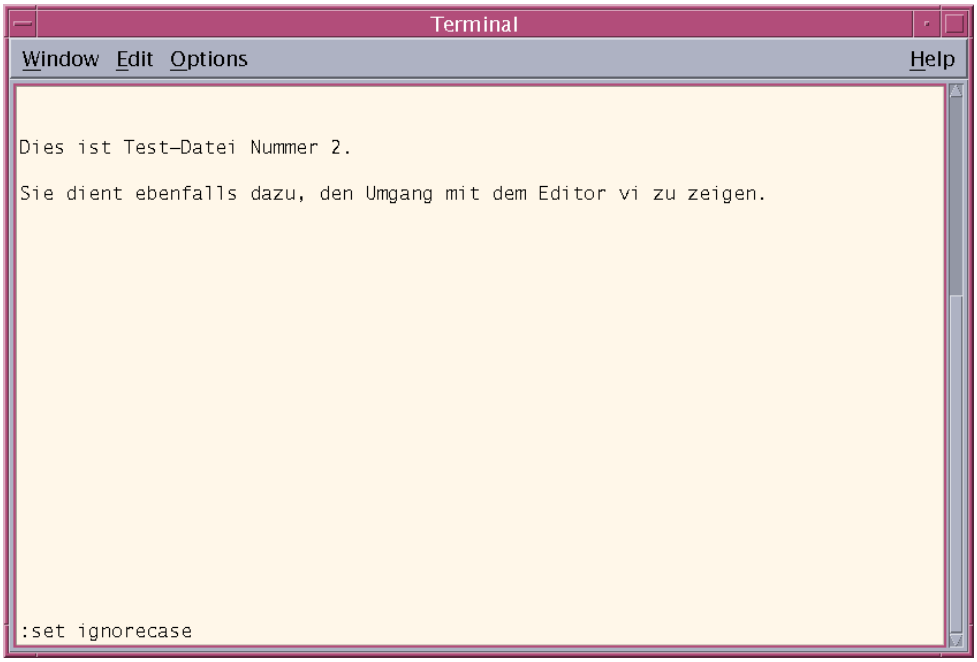

Abbildung 34: Groß- und Kleinschreibung ignorieren

„Nummer“ gefunden wird, wenn als Suchmuster „nummer“ eingegeben wird.

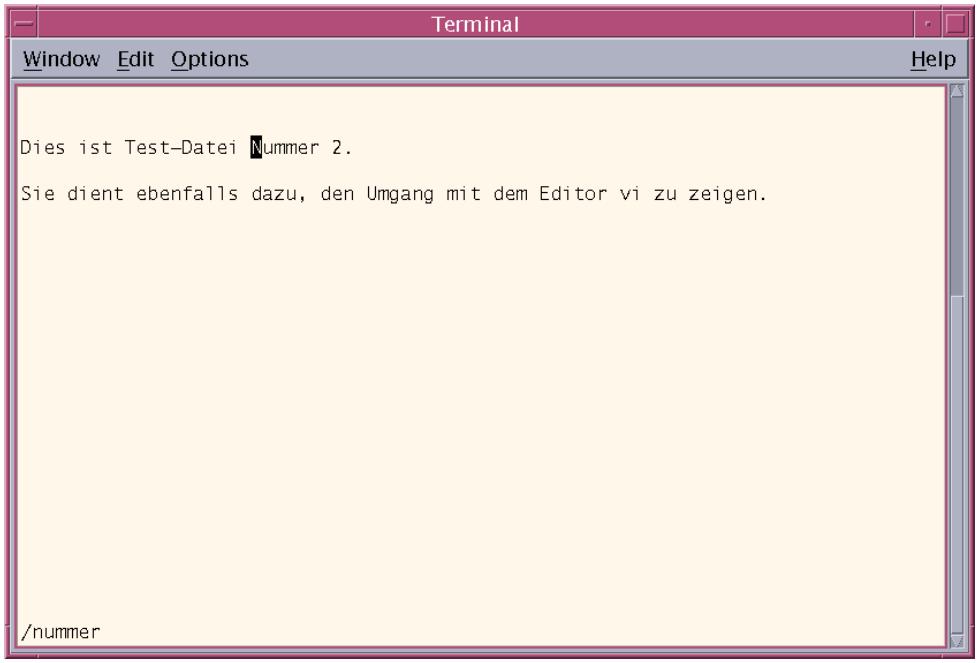

Abbildung 35: Suche mit ignorierter Groß- und Kleinschreibung

```
set noic
```

führt zu Unterscheidung zwischen Groß- und Kleinschreibung. Wird die Unterschei-

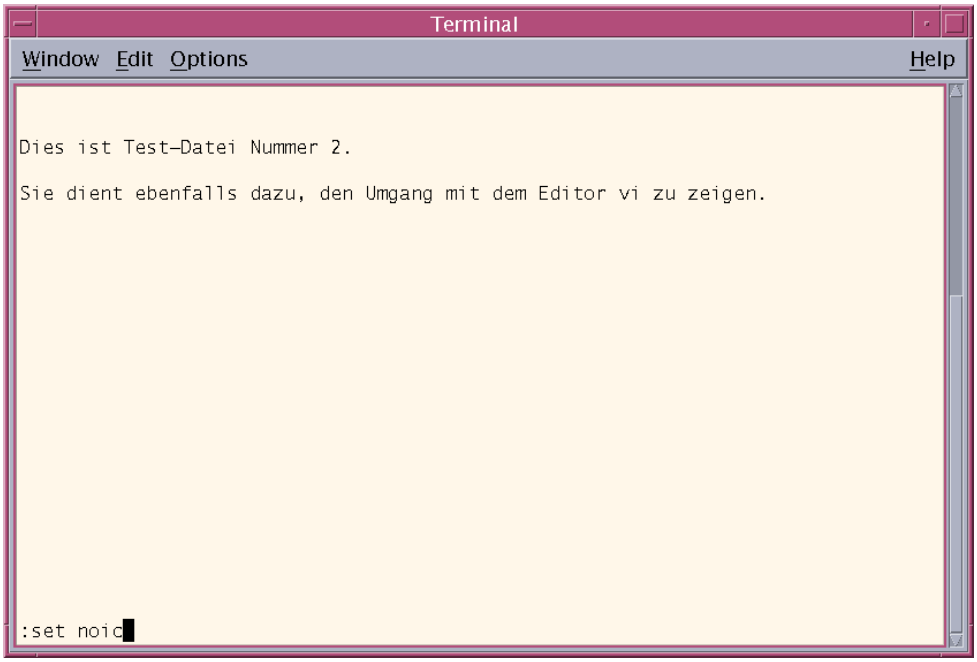

Abbildung 36: Groß- und Kleinschreibung unterscheiden

dung eingeschalten, findet das Suchmuster „nummer“ den Text „Nummer“ nicht, siehe Abb. 37 auf der nächsten Seite.

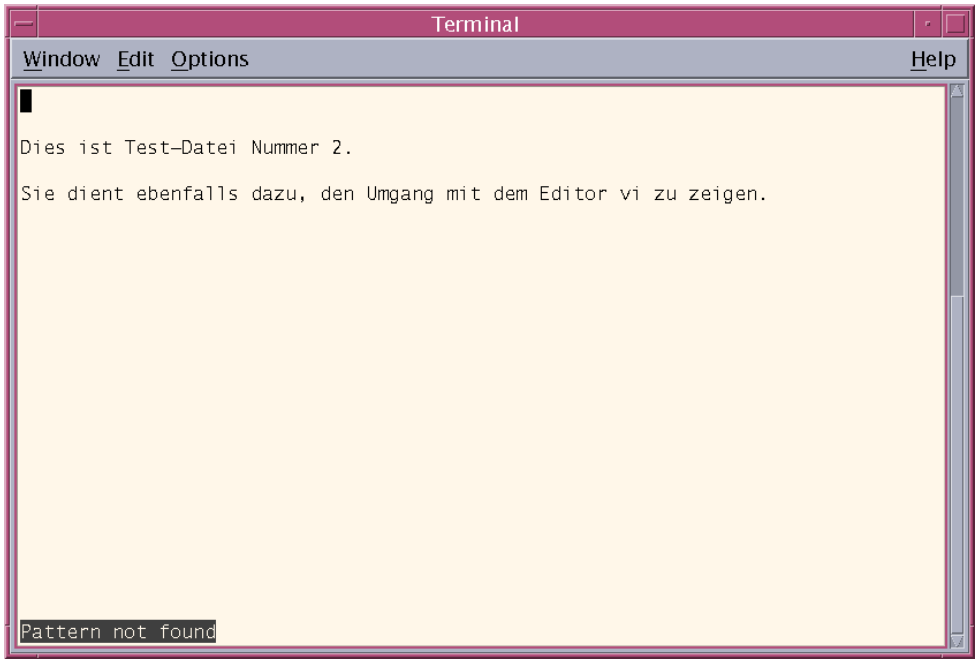

Abbildung 37: Fehlgeschlagene Suche

5.8.2 Automatische Einrückung

Mit

```
set autoindent
```

wird die automatische Einrückung eingeschaltet. Automatische Einrückung bedeutet,

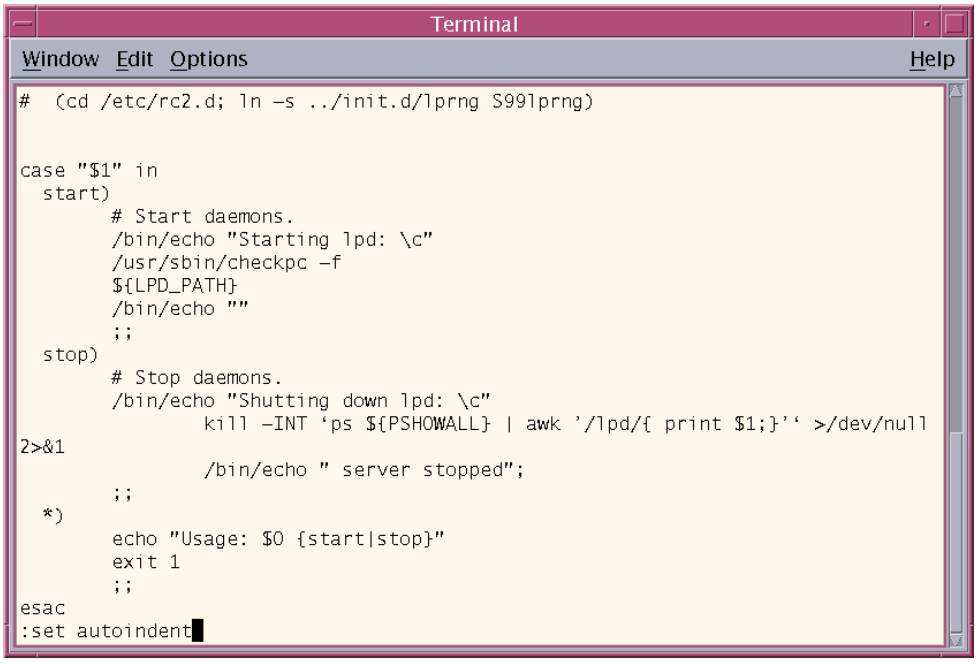

Abbildung 38: Automatische Einrückung einschalten

daß beim Anhängen einer neuen Zeile an eine vorhandene automatisch soviel Leerraum am Zeilenbeginn der neuen Zeile erzeugt wird, wie in der Vorgängerzeile vor dem ersten Textzeichen vorhanden ist.
Dies ist insbesondere beim Programmieren und beim Schreiben von Shellscripts sinnvoll.
Um eine Einrückungsebene zurück zu gelangen, kann `CTRL-d` genutzt werden.

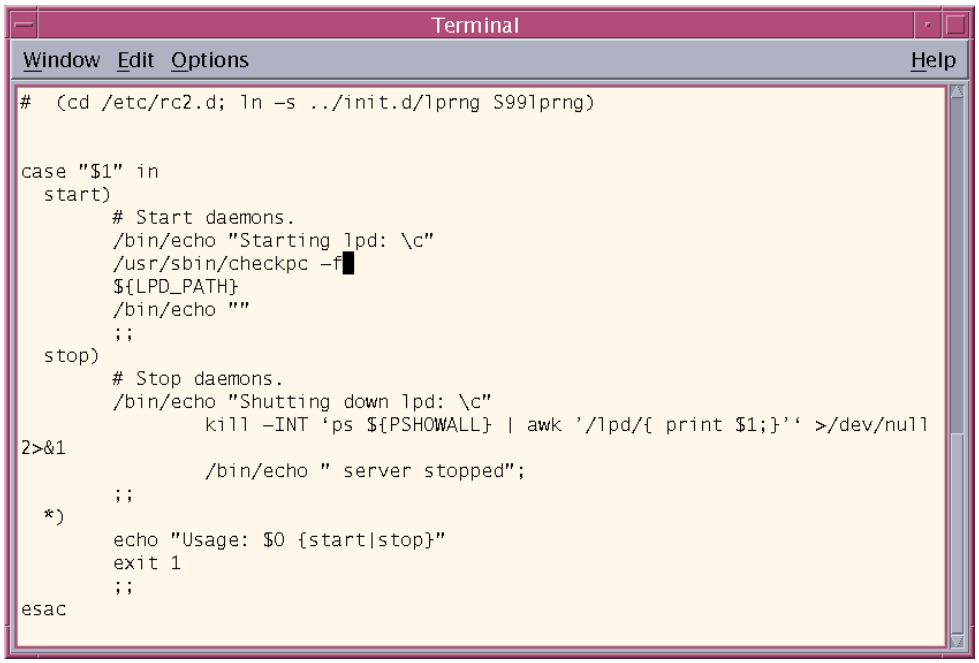

Abbildung 39: Vor Anhängen einer neuen Zeile

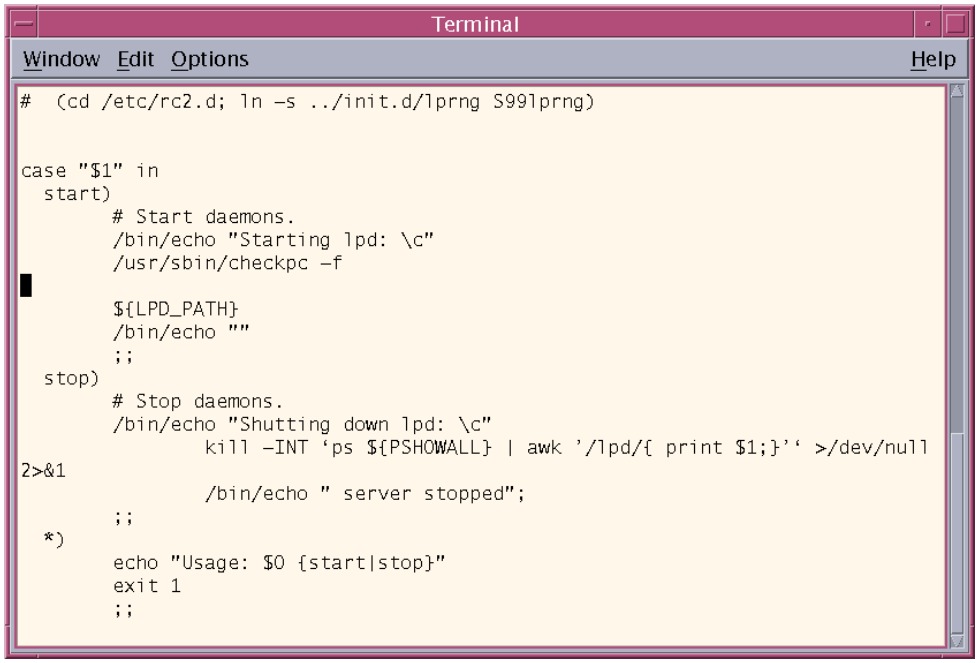

Abbildung 40: Neue Zeile, wenn automatische Einrückung aus

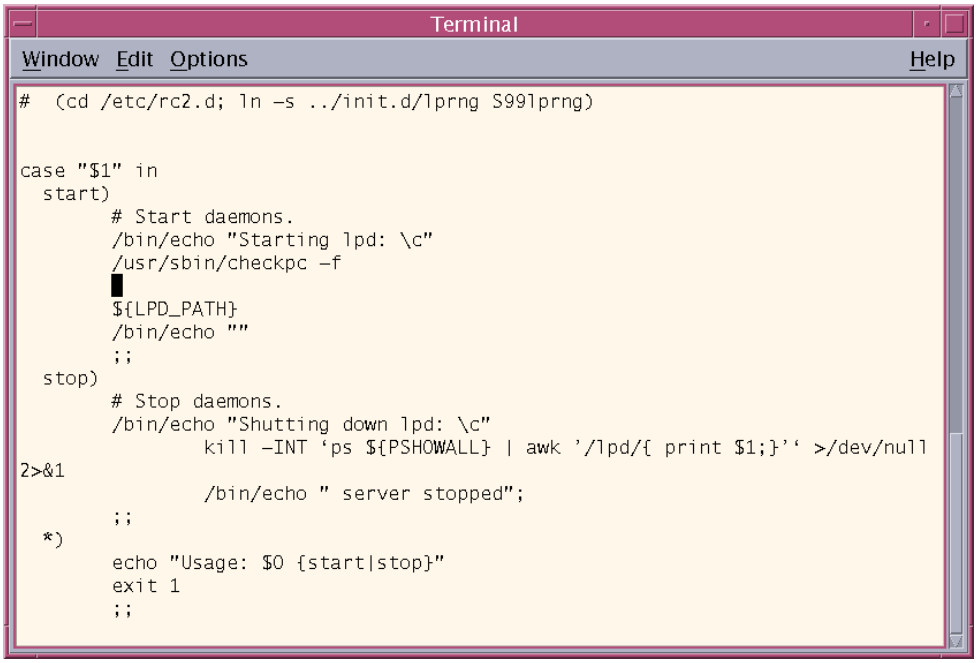

Abbildung 41: Neue Zeile, wenn automatische Einrückung ein

Mit

```
set noai
```

wird die automatische Einrückung wieder ausgeschaltet.

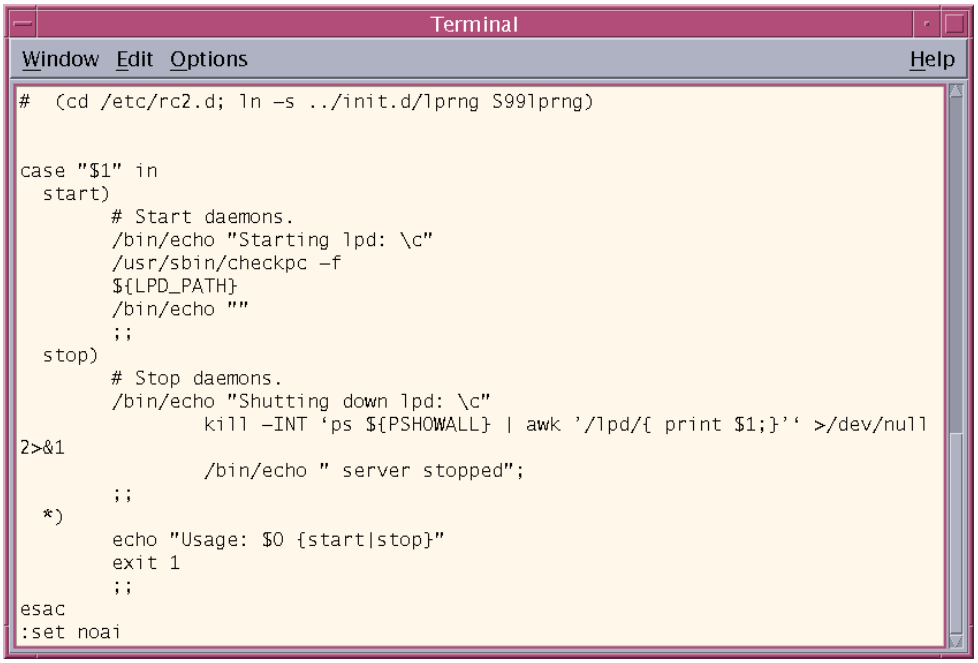

Abbildung 42: Automatische Einrückung ausschalten

5.8.3 Tabulator-Weite festlegen

Mit

`set ts=`*Anzahl*

wird die Tabulator-Schrittweite festgelegt, beispielsweise setzt

`set ts=4`

diese auf 4.

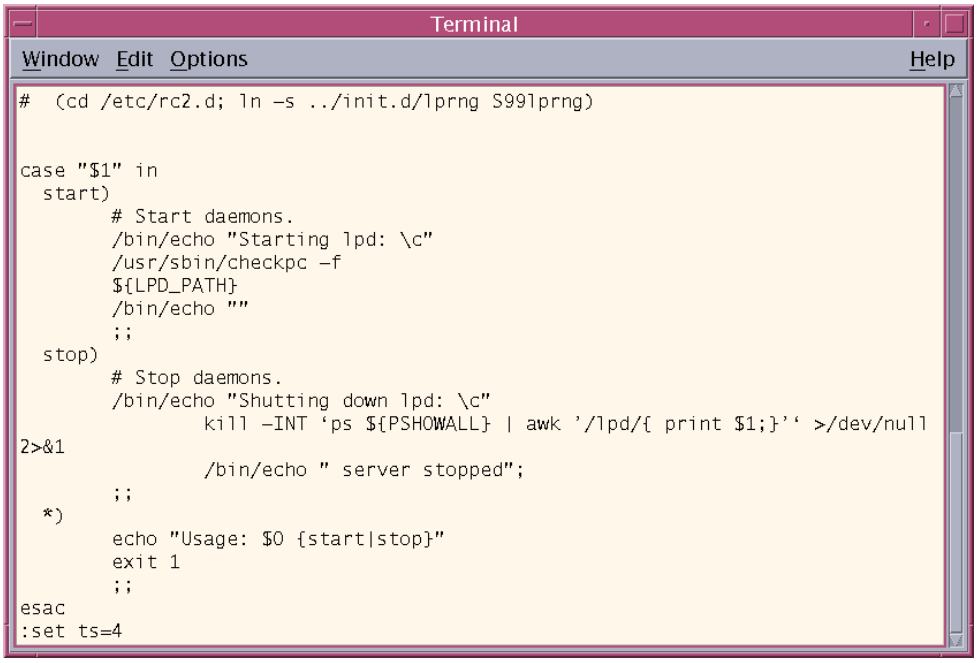

Abbildung 43: Festlegung der Tabulator-Schrittweite

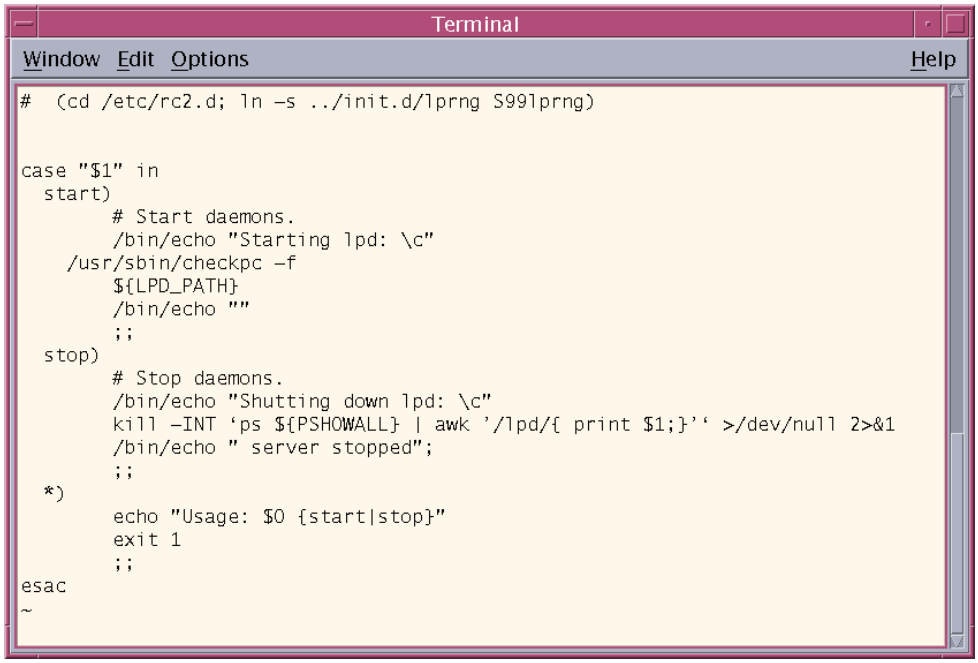

Abbildung 44: Datei mit geänderter Tabulator-Schrittweite

5.8.4 Zeilennummern-Anzeige

Mit

```
set number
```

wird die Anzeige von Zeilennummern vor den Textzeilen eingeschaltet,

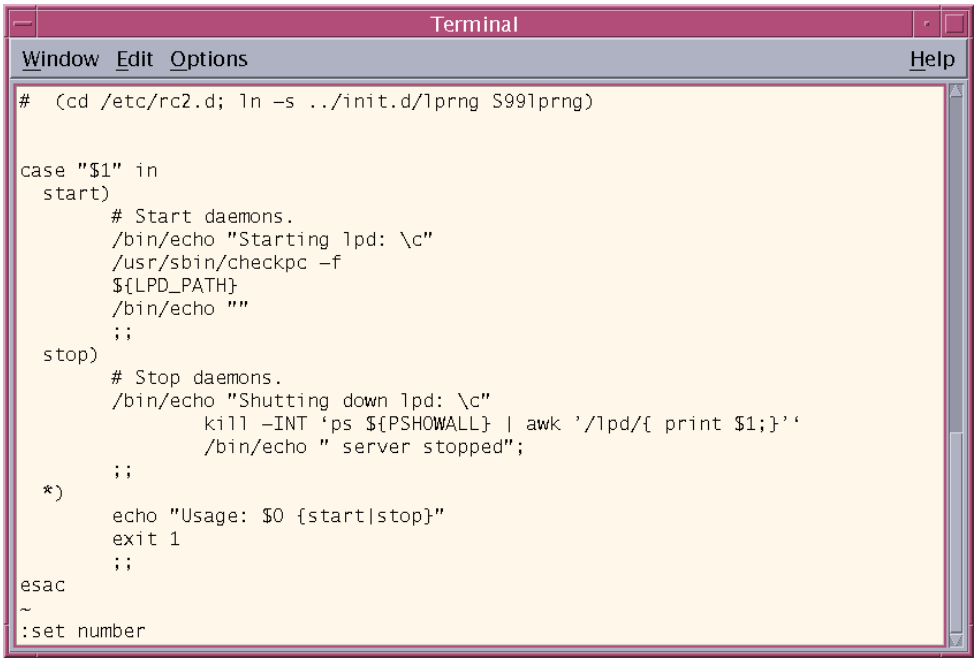

Abbildung 45: Einschalten der Zeilennummerierung

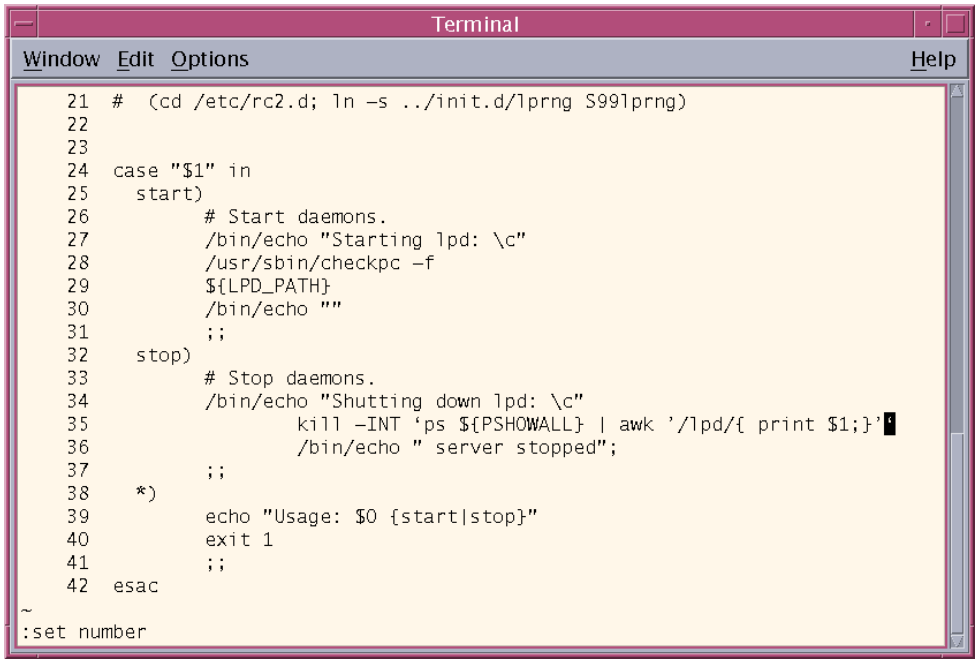

Abbildung 46: Text mit angezeigten Zeilennummern

`set nonu`

schaltet die Anzeige aus.

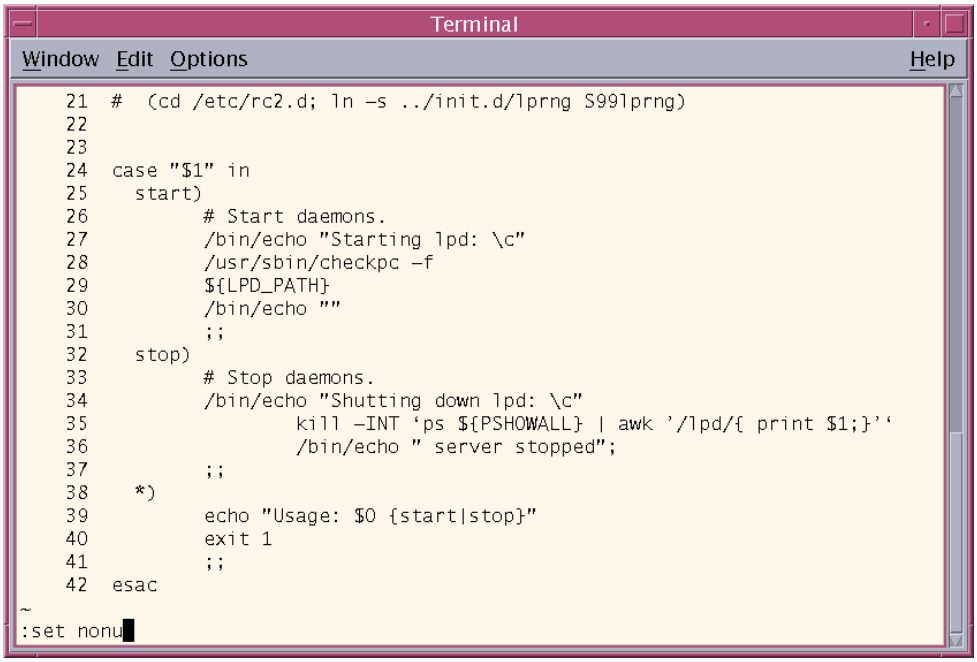

Abbildung 47: Ausschalten der Zeilennummerierung

5.8.5 Modus anzeigen

Mit

```
set showmode
```

erreicht man, daß im Einfügemode in der Statuszeile der Hinweis INSERT bzw. APPEND MODE erscheint.

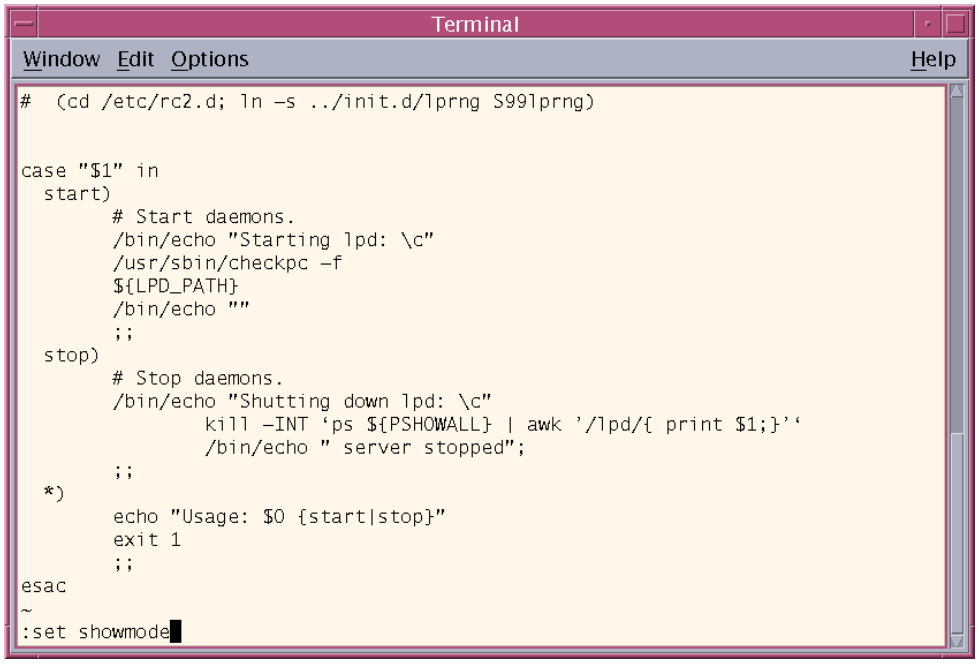

Abbildung 48: Modus-Anzeige einschalten

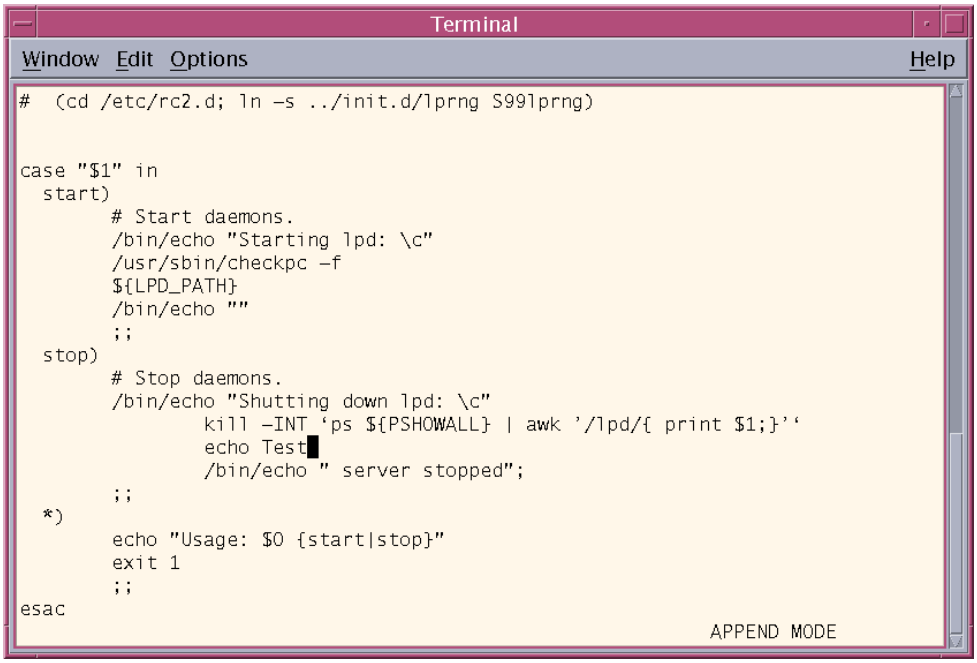

Abbildung 49: Eingeschaltene Modus-Anzeige im Anhänge-Modus

`set noshowmode`

schaltet die Modus-Anzeige ab.

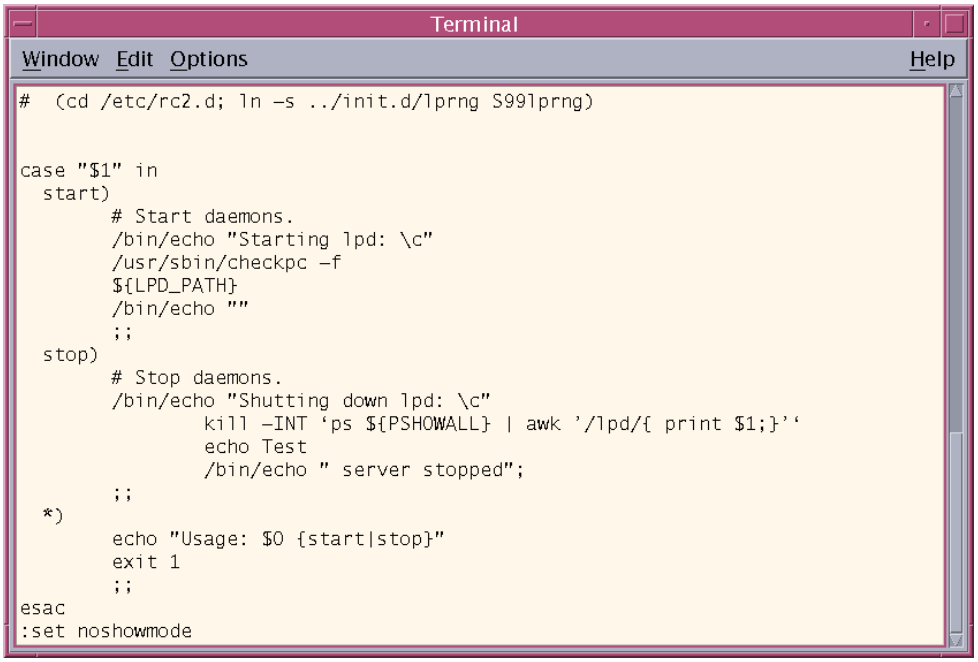

Abbildung 50: Modus-Anzeige ausschalten

5.8.6 Steuerzeichen anzeigen

```
set list
```

bewirkt, daß die im Text befindlichen Steuerzeichen angezeigt werden.

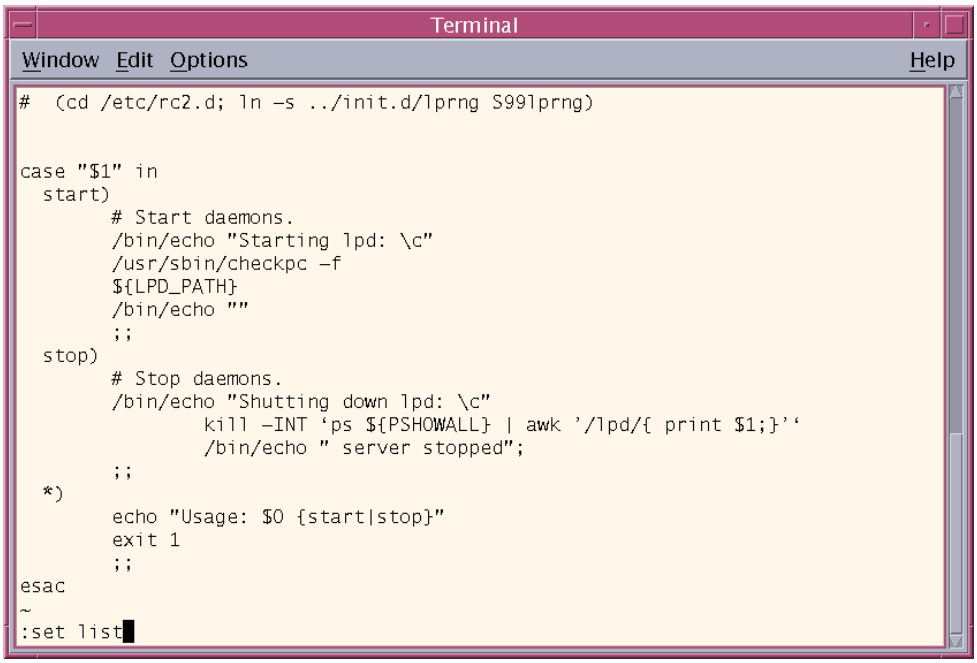

Abbildung 51: Steuerzeichen-Anzeige einschalten

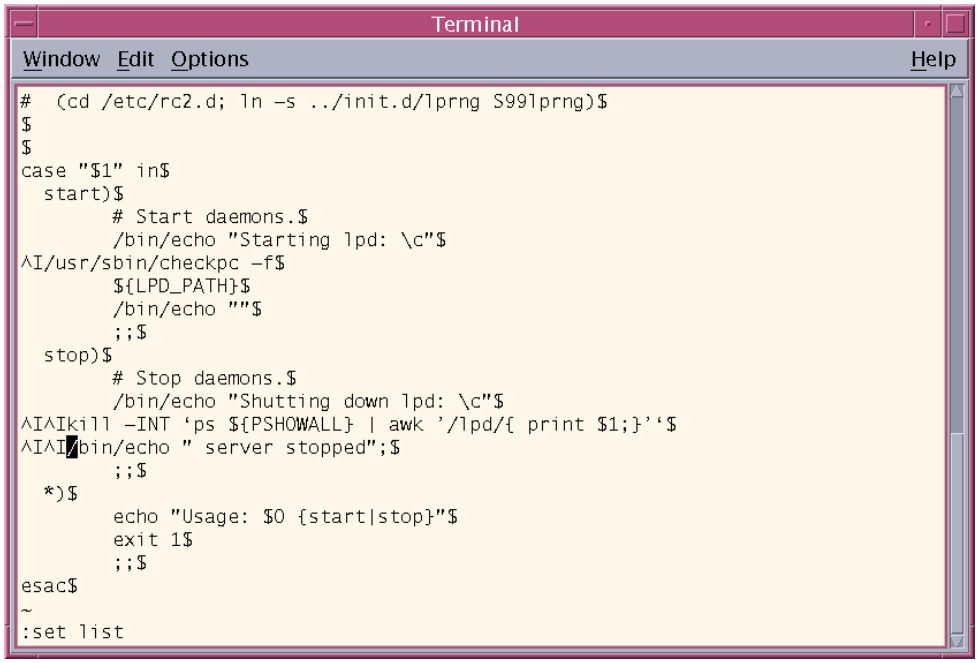

Abbildung 52: Text mit angezeigten Steuerzeichen

`set nolist`

schaltet die Anzeige von Steuerzeichen aus.

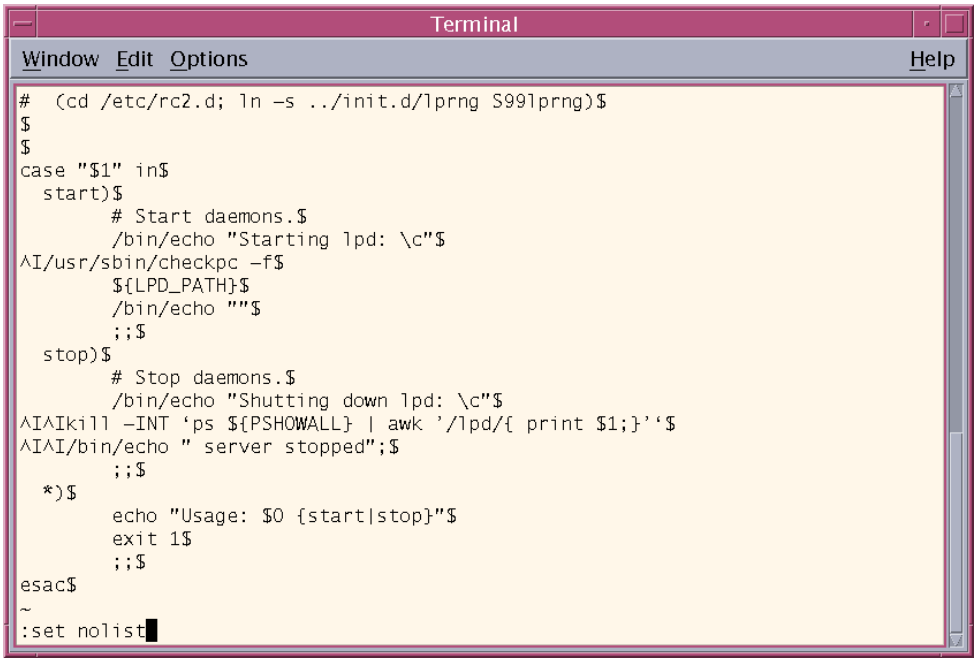

Abbildung 53: Steuerzeichen-Anzeige ausschalten

6 Einstellungen dauerhaft festlegen

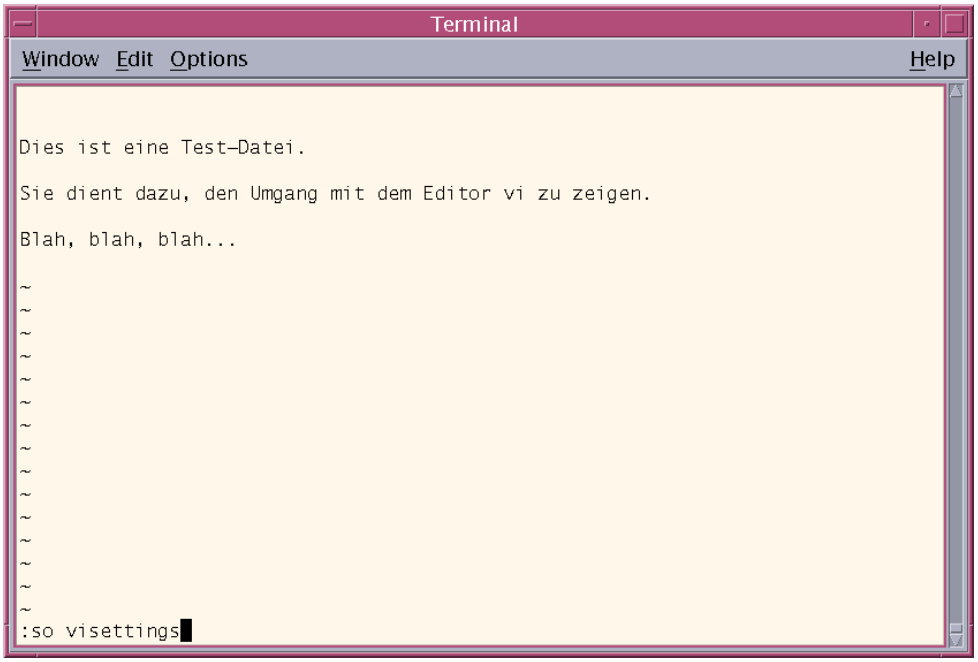

Abbildung 54: Kommandodatei einlesen

Einstellungen und Makrodefinitionen können mit dem Befehl `so` (source)

`so` *Datei*

aus einer Datei eingelesen werden.
Damit kann man sich ersparen, die Einstellungen immer wieder aufs Neue manuell festlegen zu müssen.
Bei jedem Programmstart wird geprüft, ob eine Variable EXINIT existiert. Falls ja, wird ihr Inhalt als Kommando interpretiert und ausgeführt. Falls nicht, wird getestet, ob im Home-Verzeichnis eine Datei „.exrc“ existiert. Falls ja, wird diese Kommandodatei eingelesen.
Hier ein Beispiel für eine EXINIT-Variablen-Definition in der C-Shell:

```
setenv EXINIT "set autoindent"
```

Hier eine Beispiel-Datei „.exrc“:

```
set autoindent
set number
```

Wird die „.exrc“-Datei beim Programmstart nicht eingelesen, sollte die EXINIT-Variable folgendermaßen definiert werden:

```
setenv EXINIT "so ${HOME}/.exrc"
```

Bei einigen vi-Clonen (z.B. vim, vi improved) hat die Start-Datei einen anderen Namen (z.B. $VIM\.._vimrc oder $HOME/.vimrc).

7 Reguläre Ausdrücke

7.1 Aufbau regulärer Ausdrücke

Beim Suchen und Ersetzen können einfache reguläre Ausdrücke verwendet werden. Mit regulären Ausdrücken werden Suchmuster definiert.
Einige Zeichen haben in Suchmustern eine spezielle Bedeutung (`/`, `&`, `!`, `.`, `^`, `*`, `\`, `?`, `[`, `]`). Diesen Zeichen muß ein Backslash `\` vorangestellt werden, wenn die Sonderbedeutung nicht zum Tragen kommen soll.

Konstrukt	Bedeutung
`^`	Zeilenbeginn
`$`	Zeilenende
`\<`	Wortanfang
`\>`	Wortende
`.`	ein beliebiges Zeichen
`[abcde]`	ein Zeichen aus der Menge a, b, c, d, e
`[a-f]`	ein Zeichen aus der Menge von a bis einschließlich f
`*`	der vorangegangene Suchausdruck 0- oder mehrfach

7.2 Beispiele

`^Text` sucht nach der Zeichenfolge `Text`, die am Zeilenanfang steht.
`Text$` sucht nach der Zeichenfolge `Text`, die am Zeileende steht.
`[abc]*text` sucht nach der Zeichenfolge `text`, der eine beliebige Anzahl (0 oder mehrfach) von `a`, `b` bzw. `c` vorausgeht.
`[^a-f]*text` sucht nach der Zeichenfolge `text`, der eine beliebige Anzahl Zeichen vorausgeht, die nicht im Bereich von `a` bis `f` liegen.

8 Unterstützung für Programmierer

Der Editor vi bietet einige Features, die das Programmieren in C bzw. anderen Sprachen unterstützen.

8.1 Automatische Einrückung

Automatische Einrückung hilft, gut strukturierte und somit gut lesbare Quelltexte zu erzeugen.
Beispielsweise ist der folgende Quelltext

```
double kreis_flaeche(double r)
{
  double back = 0.0;
  /* Radius testen, ob nicht negativ */
  if(r >= 0.0) {
    /* Radius ok */
    back = M_PI * r * r;
  } else {
    /* Fehler */
    printf("ERROR: %lf < 0.0\n", r);
  }
}
```

besser lesbar als der funktional gleiche Quelltext

```
double kreis_flaeche(double r)
{
  double back = 0.0;
  /* Radius testen, ob nicht negativ */ if(r >= 0.0) {
   /* Radius ok */
back = M_PI * r * r;
      } else {
/* Fehler */
  printf("ERROR: %lf < 0.0\n", r); } }
```

Ist der Modus für automatische Einrückung eingeschaltet (mit dem Kommando `set autoindent`), werden am Beginn einer neuen Zeile automatisch so viele Leerzeichen eingefügt, wie am Beginn der vorangegangenen Zeile.
Mit dem Kommando `set noai` wird die automatische Einrückung wieder ausgeschaltet.
Um im eingeschaltenen Einrückungsmodus am Anfang einer Zeile eine Einrückungsebene zurück zu gelangen, muß `CTRL-d` eingegeben werden.

8.2 Klammersuche

Möchte man zu einer öffnenden (bzw. schließenden) Klammer die zugehörige schließende (bzw. öffnende) Klammer finden, plaziert man den Cursor auf der bekannten Klammer und gibt im Auftextmode

```
%
```

ein. Durch mehrfache Eingabe wird zwischen zusammengehörigen Klammern hin- und hergewechselt.

8.3 Beginn einer Funktion suchen

Gibt man im Auftextmodus

```
[[
```

bzw.

```
]]
```

ein, springt der Cursor zur letzten (bzw. nächsten) öffnenden geschweiften Klammer, die den Funktionskörper einer C-Funktion eröffnet.

8.4 Nutzung von ctags

8.4.1 Tags-Datei anlegen

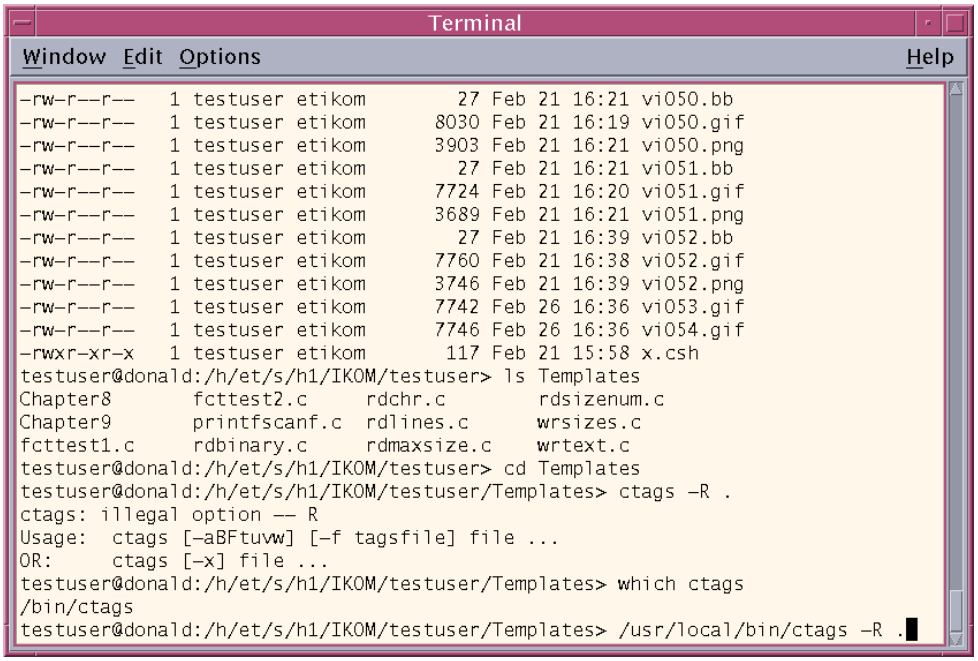

Abbildung 55: tags-Datei anlegen

In größeren Programmierprojekten ist es häufig schwierig zu überblicken, welche Funktion bzw. Variable in welchem Quelltextmodul definiert wird.
Hier hilft das Programm „ctags“. Das Programm durchsucht die Quelltexte und legt eine Datei (häufig mit dem Namen „tags“) an, die angibt, an welchen Stellen in welcher Datei die einzelnen Funktionen und Variablen definiert werden.
Diese Datei kann von einigen Editoren genutzt werden, um zu den Definitionen bestimmter Bezeichner zu springen. Die Syntax für den Aufruf von ctags variiert zwischen verschiedenen Rechnern. In unserem Workstation-Pool ist neben dem standardmäßig im Lieferumfang von Solaris enthaltenen ctags auch die Freeware „Exuberant Ctags“ [1] installiert.
In Abb. 55 ist zu sehen, wie mit dem Aufruf

```
/usr/local/bin/ctags -R .
```

eine „tags“-Datei für das aktuelle Verzeichnis (`.`) mit allen enthaltenen Dateien einschließlich derer in Unterverzeichnissen (`-R`, recursive) angelegt wird.
Der Pfad zu `/usr/local/bin/ctags` ist angegeben, um die nachträglich installierte Version von ctags anstelle der mit dem Betriebssystem mitgelieferten Version /bin/ctags zu nutzen.

[1] http://ctags.sourceforge.net

8.4.2 Zu Bezeichner springen

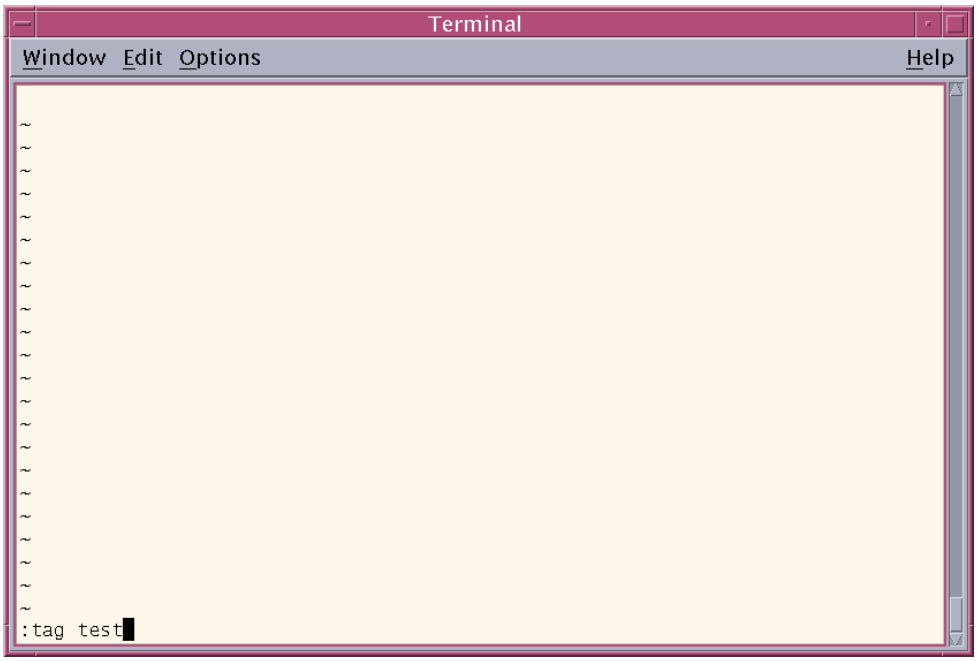

Abbildung 56: Sprung zu Bezeichner veranlassen

Abb. 56 zeigt das Kommando

`tag` *Bezeichner*

mit dem in diesem Fall zur Definition von „test“ gesprungen werden soll.

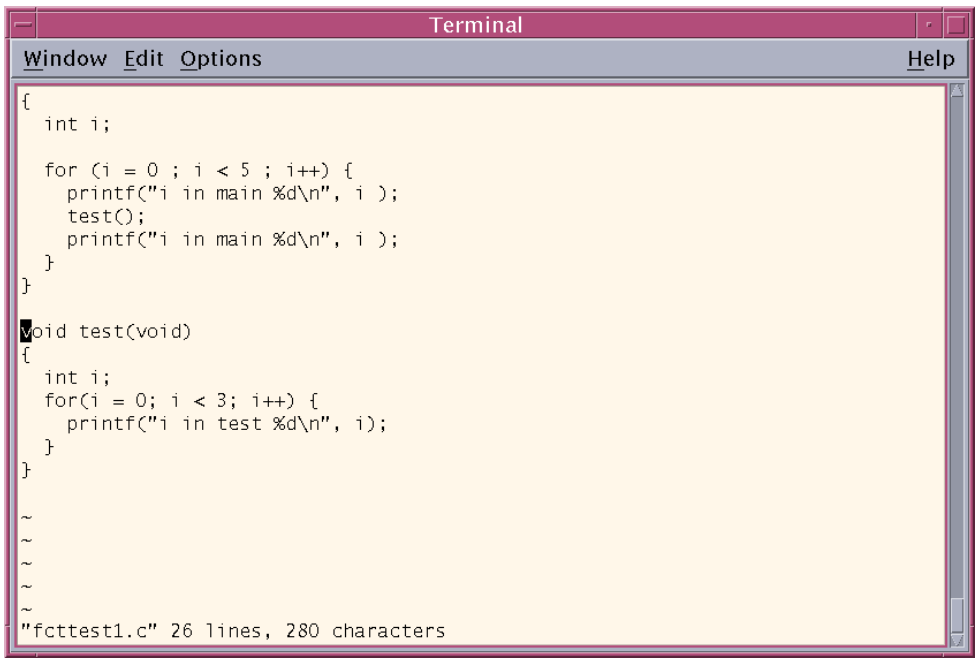

Abbildung 57: Bezeichner gefunden

Abb. 57 zeigt, daß die passende Datei gefunden und geöffnet wurde und zur Definition des Bezeichners „test“ gesprungen wurde.

8.4.3 Kompatibilitätsprobleme

Da es unterschiedliche Versionen von ctags-Programmen und auch unterschiedliche vi-Clones gibt, arbeitet möglicherweise nicht jede Version von ctags mit jedem vi-Clone zusammen, insbesondere da es unterschiedliche Format-Definitionen für die tags-Datei gibt.
Sollten Probleme auftreten, empfehle ich die Kombination von „Exuberant Ctags“ [2] und „Vi IMproved“ [3].
Beide Programme sind sowohl für Windows als auch für verschiedene UNIX- bzw. UNIX-ähnliche Betriebssysteme verfügbar.

[2] http://ctags.sourceforge.net
[3] http://www.vim.org

8.5 Compiler-Lauf in vim

Der vi-Clone „vim“ (möglicherweise auch andere vi-Clones) kann direkt das Programm „make“ starten.

8.5.1 Compiler-Lauf starten

Hierzu wird im Kommandomodus

`make` *target*

eingegeben, wobei „target“ durch den Namen der Datei ersetzt wird, die durch den make-Aufruf erzeugt werden soll. Im Beispiel in Bild 58 wird die Datei „x.c“ bearbeitet, ein make-Lauf wird gestartet. Die Ausgabe des make-Laufes erscheint im

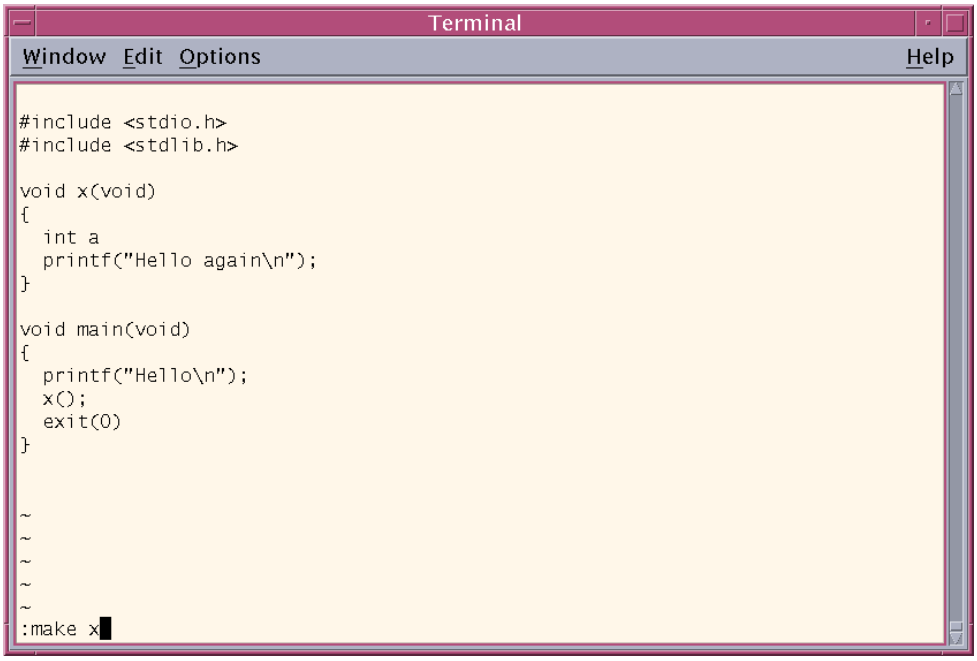

Abbildung 58: make-Lauf starten

Fenster mit der Aufforderung, ENTER zu drücken, siehe Abb. 59 auf der nächsten Seite.

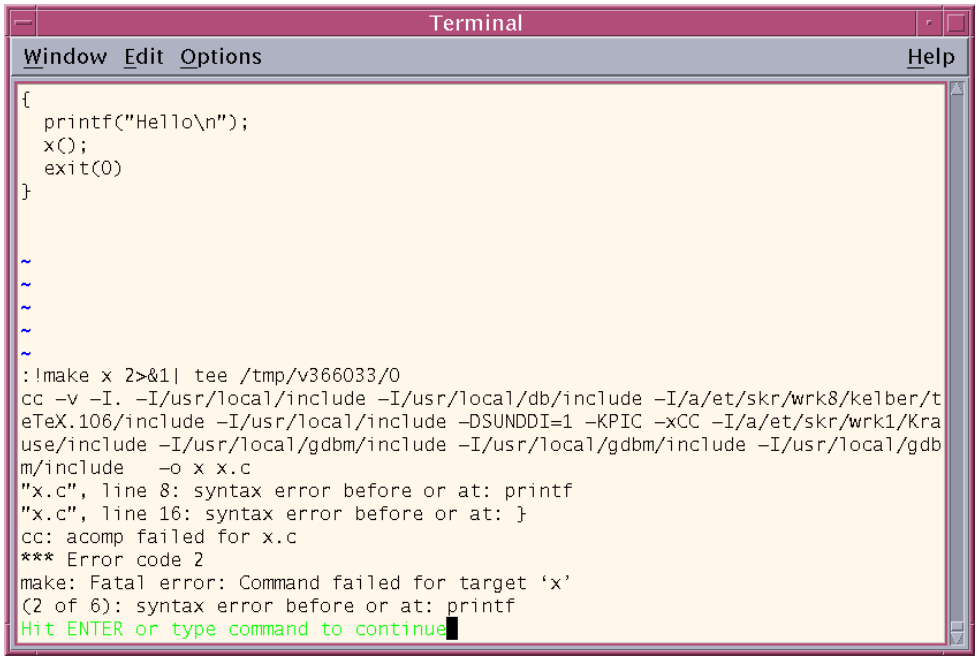

Abbildung 59: Nach dem make-Lauf

8.5.2 Reaktion bei Fehlern

Nachdem ENTER gedrückt wurde, wird der Quelltext geladen, in dem der erste Fehler auftrat. Es wird direkt zur Fehlerstelle gesprungen (Abb. 60).

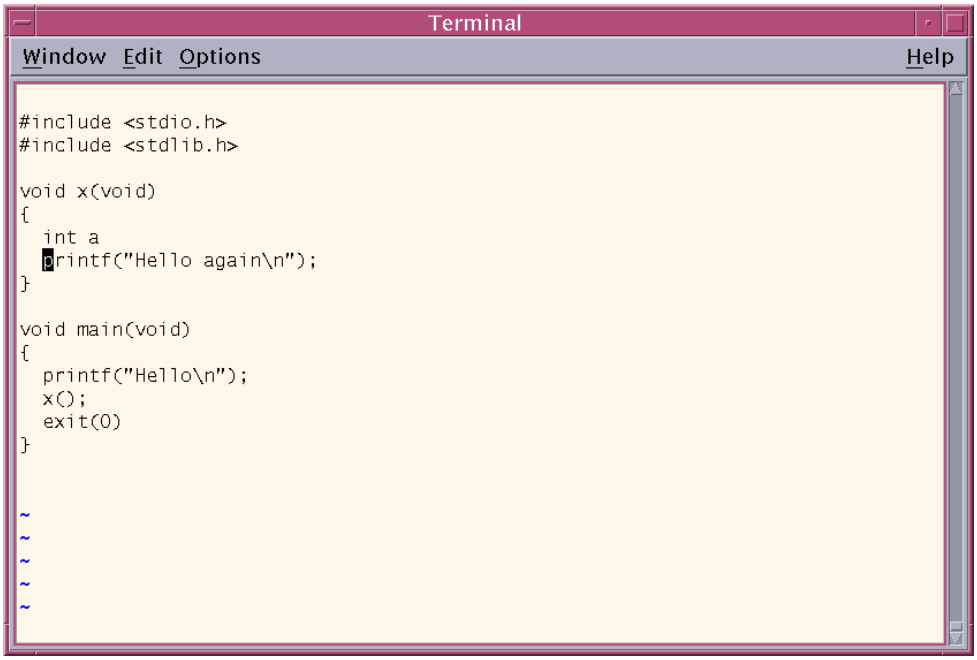

Abbildung 60: Positionierung auf erstem Fehler

Wurde der Fehler bereinigt, kann im Kommandomodus mit

```
cn
```

zur nächsten Fehlerstelle gesprungen werden (Abb. 61 und 62 auf der nächsten Seite). In der Statuszeile wird der Text der Fehlermeldung angezeigt. Mit dem Kommando

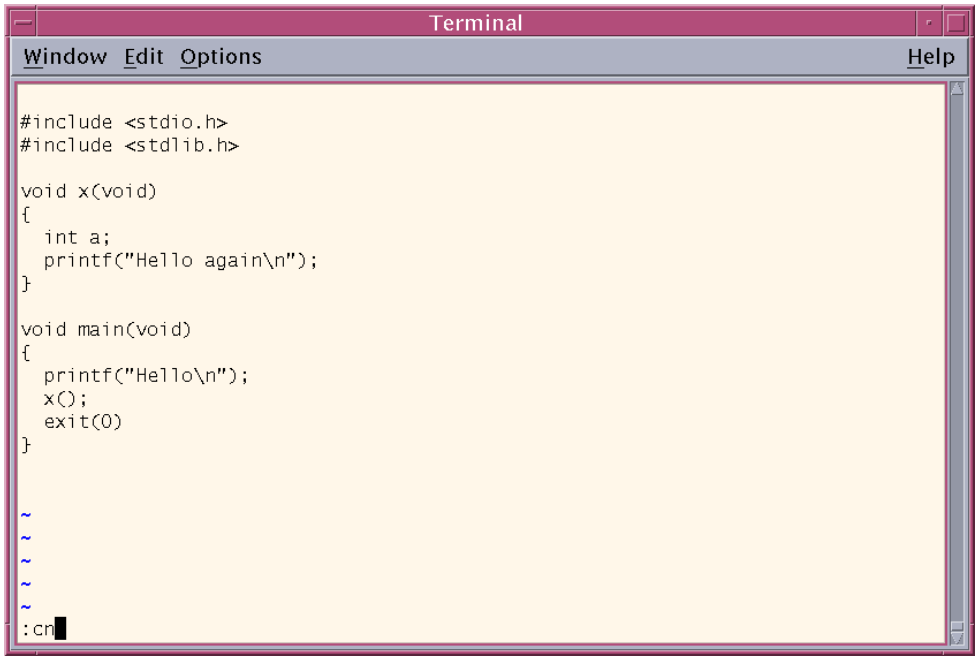

Abbildung 61: Zur nächsten Fehlerstelle springen

```
cp
```

wird zur vorherigen Fehlermeldung gesprungen.

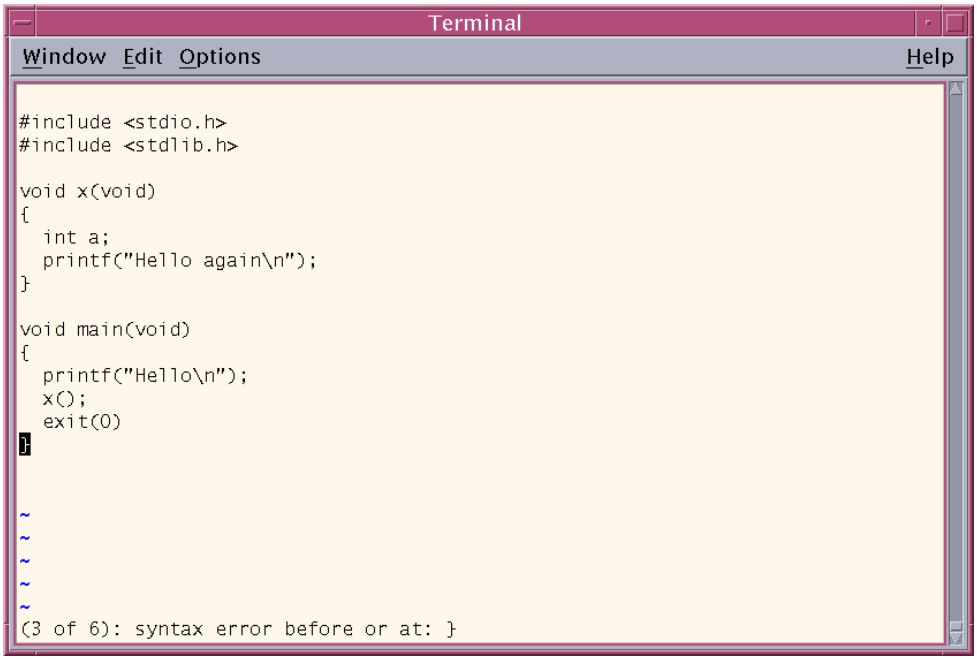

Abbildung 62: Positionierung auf nächster Fehlerstelle

Mit dem Kommando

```
cl
```

wird eine Liste aller „anspringbaren“ Fehler-Nummern angezeigt, siehe Abb. 63 und 64 auf der nächsten Seite. Im Beispiel können die Fehler 2 und 3 angesprungen

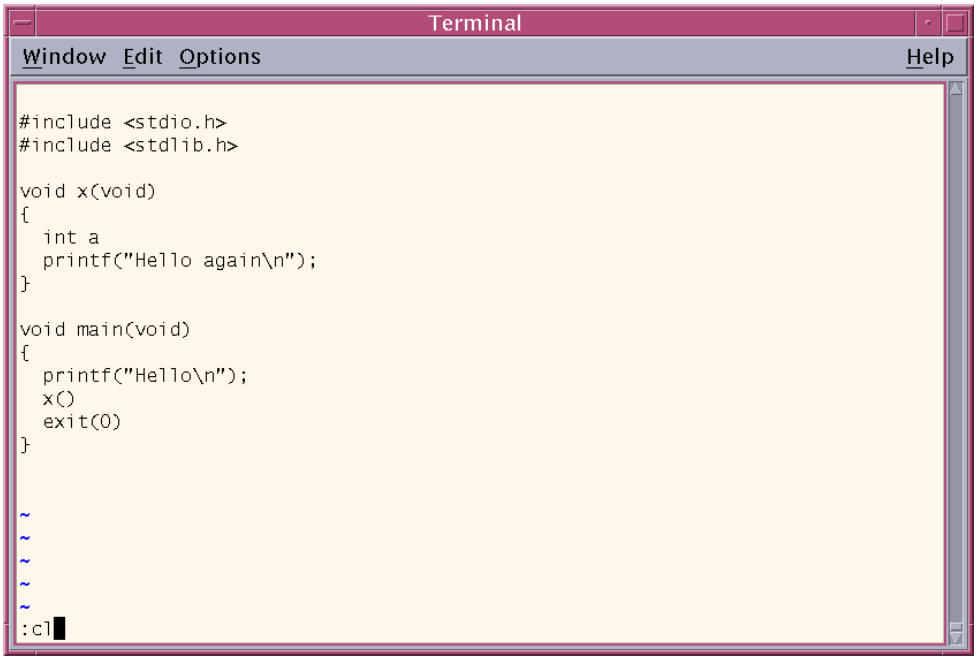

Abbildung 63: Fehler-Liste anzeigen lassen

werden, den anderen Fehlern ist keine bestimmte Stelle im Quelltext direkt zuzuordnen. Mit dem Kommando

```
ccn
```

wird direkt zum Fehler Nummer n gesprungen, dabei muß der Buchstabe „n“ durch die entsprechende Nummer ersetzt werden.

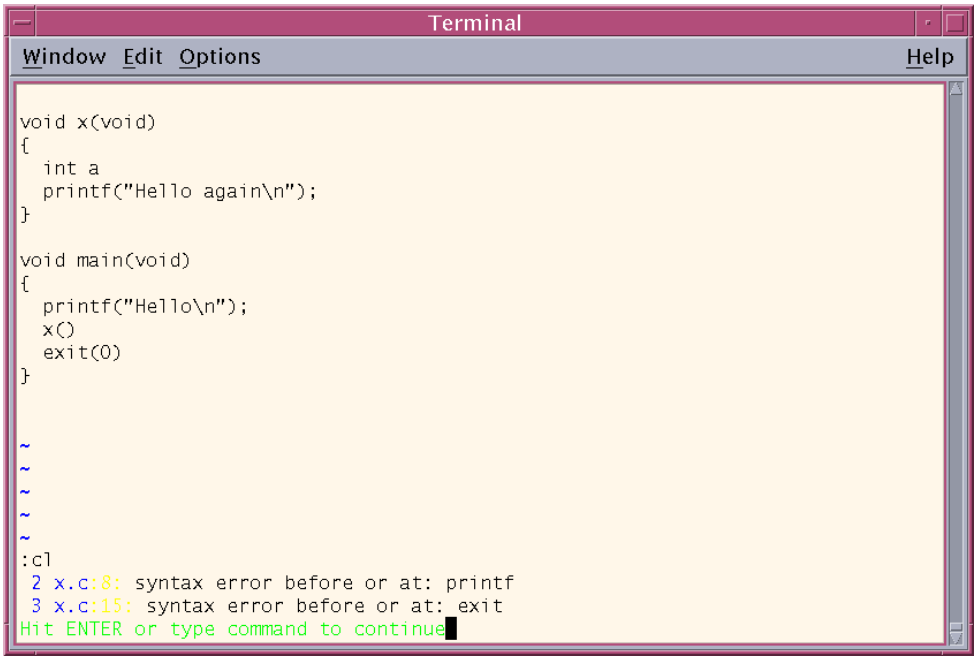

Abbildung 64: Fehler-Liste wird angezeigt

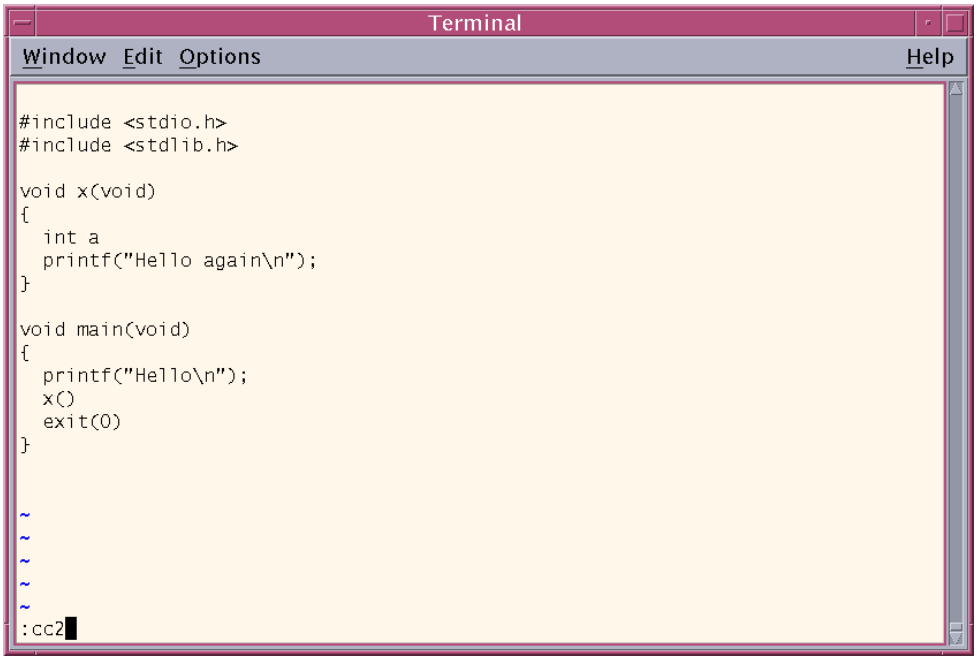

Abbildung 65: Zu Fehler 2 springen

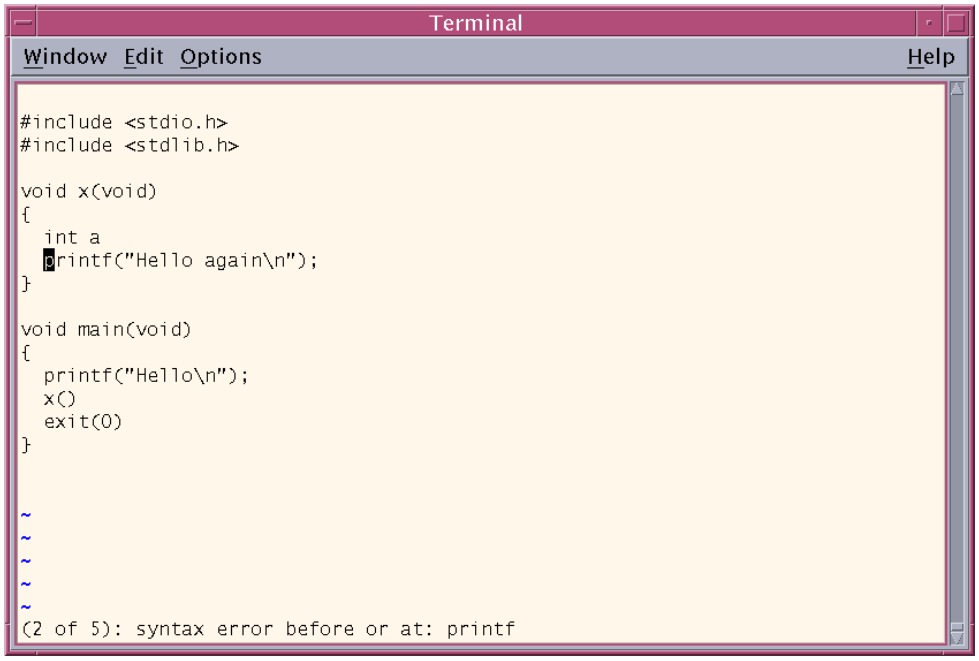

Abbildung 66: An Fehlerposition 2

8.5.3 Anderes make-Programm auswählen

Ein anderes make-Programm (z.B. „gmake“) kann im Kommandomodus mit

`set makeprg=`*prog*

eingestellt werden, hierbei muß „prog“ durch den Namen des Programmes (z.B. „gmake“ oder „dmake“) ersetzt werden (Abb. 67).

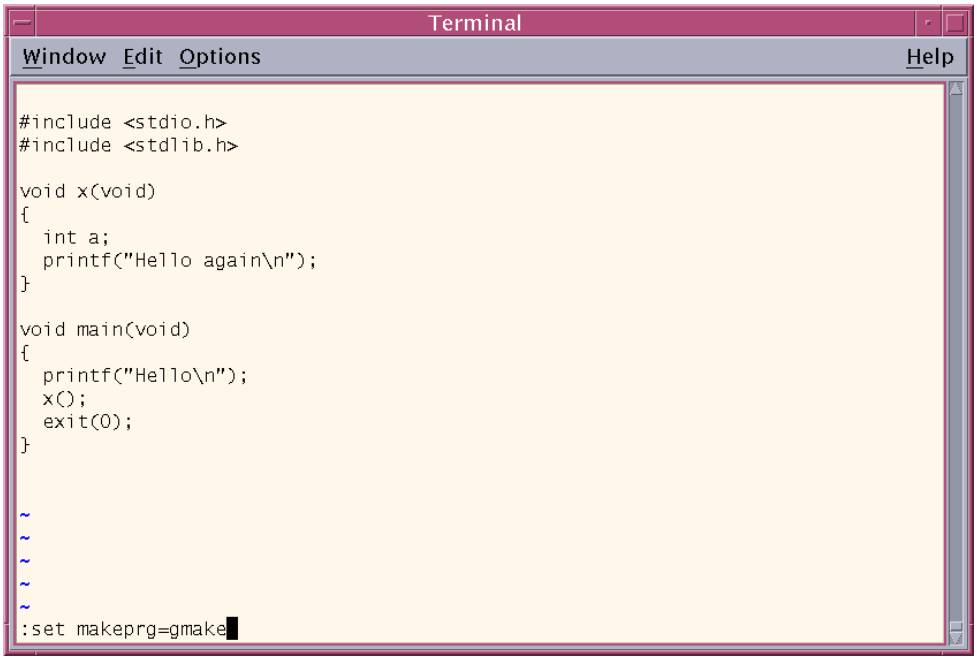

Abbildung 67: Anderes make-Programm festlegen

8.6 Syntax-Hervorhebung mit vim

Mit dem Kommando

```
syntax on
```

wird im vim die Syntaxhervorhebung eingeschalten, mit

```
syntax off
```

schaltet man sie wieder aus.

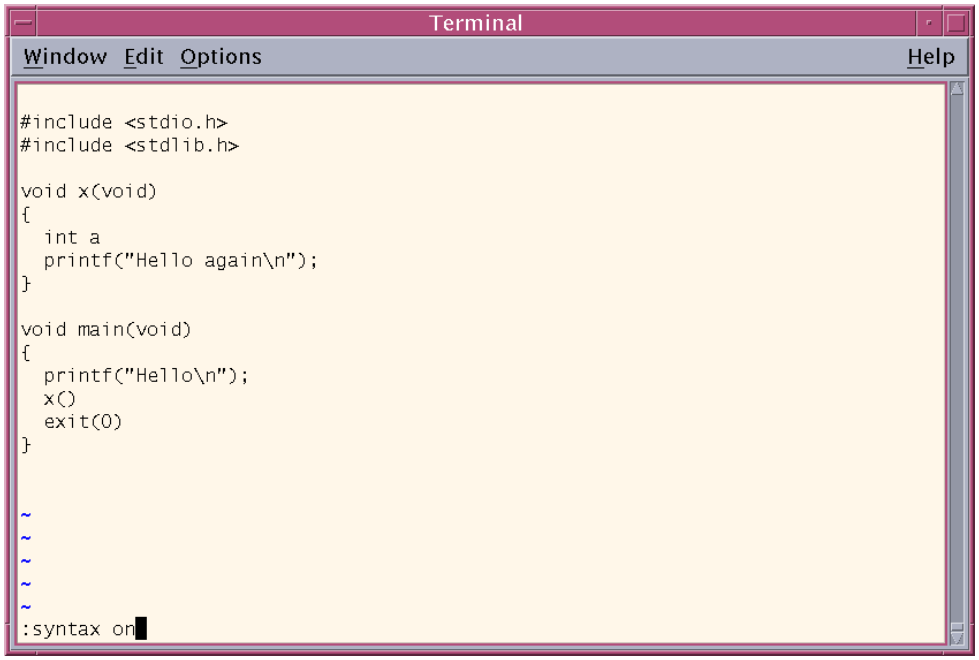

Abbildung 68: Syntaxhervorhebung einschalten

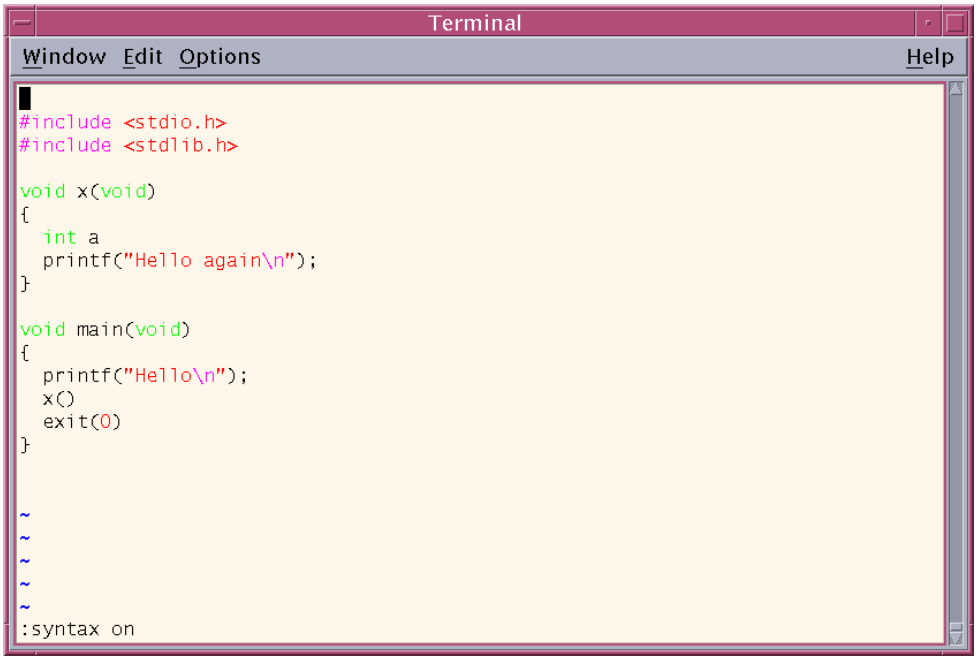

Abbildung 69: Quelltext mit hervorgehobener Syntax

9 Erweiterungen in vim

9.1 Überblick

In diesem Kapitel werden Erweiterungen des frei verfügbaren vi-Clones vim besprochen, die in den vorangegangenen Kapiteln nocht nicht erwähnt wurden.

9.2 Konfigurationsdateien

9.2.1 Start von vim

Vim sucht beim Start die Konfigurationsdatei „.vimrc“ anstelle der „.exrc“ des originalen vi.
Diese Datei wird zunächst im HOME-Verzeichnis des Benutzers gesucht, ist sie dort nicht vorhanden, wird in dem Verzeichnis weitergesucht, in dem vim installiert ist.
Unter Win32-Systemen heißt die Datei möglicherweise „_vimrc“.

9.2.2 Nutzer-spezifisches Konfigurationsverzeichnis

Nutzer haben die Möglichkeit, private Konfigurationsdaten im Unterverzeichnis „.vim“ in ihrem Homeverzeichnis abzulegen. Konfigurationsdateien, die dort gefunden werden, werden anstelle der entsprechenden Dateien im vim-Installationsverzeichnis genutzt.
Die Subdirectory-Struktur dieses Verzeichnisses entspricht der im vim-Installationsverzeichnis. Im folgenden werden einige Informationen zur Subdirectory-Struktur gegeben.

9.2.3 Plug-Ins

Als Plug-In werden Dateien bezeichnet, die automatisch bei jedem Start von vim geladen werden. Die Dateien befinden sich im Verzeichnis $HOME/.vim/plugin bzw. $VIM/. . . /plugin und haben die Dateiendung „.vim“.
Die Dateien enthalten vim-Scripts.

9.2.4 Einstellungen für bestimmte Dateitypen

Vim bietet die Möglichkeit, Dateitypen automatisch zu erkennen und beim Laden einer neuen Datei automatisch ein Script auszuführen, das dem Dateityp entspricht. Hierzu muß in „.vimrc“ folgende Anweisung aktiviert werden:

```
filetype plugin on
```

Die Erkennung der Dateitypen wird über die Datei „filetype.vim“ gesteuert, die jeweils in $HOME/.vim bzw. $VIM/... gesucht wird.
Wird als Dateityp beispielsweise „c“ erkannt, wird die Datei „c.vim“ ausgeführt, die in $HOME/.vim/ftplugin bzw. $VIM/... gesucht wird.
Die Einstellungen, die in dieser Datei gesetzt werden, sollen nur für die aktuelle Datei (bzw. den aktuellen vim-Puffer) gültig sein, nicht für alle Puffer. Es muß daher die Anweisung „setlocal“ anstelle von „set“ verwendet werden, beispielsweise um automatische Einrückung u.ä. zu setzen.
Mit

```
set filetype
```

kann der herausgefundene Dateityp abgefragt werden, mit

```
set filetype=c
```

wird explizit der Dateityp „c“ gesetzt.

9.3 Vim-Scripts

9.3.1 Variable

Variablen definieren In Vim-Script können Variable definiert werden, z.B. wird mit

```
let i = 1
```

eine globale Variable „i“ mit dem Wert 1 angelegt bzw. der Wert einer evtl. bereits vorhandenen globalen Variablen mit 1 überschrieben.
Global bedeutet, daß diese Variable für alle Fenster und Puffer innerhalb des vim-Prozesses gültig ist. Wenn mehrere Scripte dieselbe globale Variable verwenden, kann dies unangenehme Nebeneffekte haben, daher wird die Verwendung lokaler Variablen empfohlen.

Tabelle 1: Gültigkeitsbereiche für lokale Variable

Prefix	Bedeutung
s:*Name*	Variable ist lokal für die gerade verarbeitete Script-Datei
b:*Name*	Variable ist lokal für aktuellen Puffer
w:*Name*	Variable ist lokal für aktuelles Fenster
g:*Name*	Globale Variable, Benutzung auch aus Funktionen heraus möglich
v:*Name*	Variable ist durch vim vordefiniert

Mit

```
let s:i = 1
```

kann eine derartige lokale Variable gesetzt werden.
Variablennamen können aus einer Kombination von ASCII-Zeichen, Unterstrichen und Zahlen bestehen, dürfen aber nicht mit einer Zahl beginnen.

Werte für Variablen Variablen können numerische Werte haben, die dezimal, hexadezimal (mit führendem „0x“) oder oktal (mit führender „0“) angegeben werden können.

```
let s:i = 12
let s:j = 0x7f
let s:k = 036
```

Variablen können auch Text beinhalten.

```
let s:name = "Joe"
```

Sonderzeichen muß ein Backslash vorangestellt werden.

```
let s:title = "Das \"Feinste vom Feinsten\""
```

Der Text kann auch in einfache Anführungszeichen gesetzt werden, es werden dann alle Zeichen exakt so übernommen, wie sie im String stehen. Von Nachteil ist, daß dann kein einfaches Anführungszeichen im Text vorkommen kann.

```
let s:title = 'Das "Feinste vom Feinsten"'
```

Variablen anzeigen Mit

```
echo s:i
```

wird die script-lokale Variable „i“ angezeigt.

Test, ob Variable existiert Mit der „exists“-Anweisung kann festgestellt werden, ob eine Variable existiert. Dabei muß der Variablen-Name als String übergeben werden, z.B.

```
if(exists("s:i"))
  echo s:i
endif
```

Variable löschen Eine Variable kann mit der „unlet“-Anweisung gelöscht werden, z.B.

```
unlet s:i
```

Ist die Variable gar nicht vorhanden, führt dies zu einer Fehlermeldung. Dieses Problem kann umgangen werden, indem die „unlet!“-Anweisung verwendet wird, diese löscht eine Variable nur dann, wenn sie existiert.

```
unlet! s:i
```

9.3.2 Ausdrücke

Variablen, Umgebungsvariablen, Optionen und Register können miteinander verknüpft werden, um Ausdrücke zu bilden.

Tabelle 2: Zugriff auf Umgebungsvariablen, Optionen und Register

Zugriffsart	Bedeutung
$*Name*	Umgebungsvariable
&*Name*	Option
@Name	Register

Beispiel

```
echo "Die Tab-Stopp-Weite ist " &ts
echo "Ihr Home-Verzeichnis ist " $HOME
echo "Register a ist " @a
```

Numerische Werte können mit den Operatoren „+“, „-“, „*“, „/“ und „%“ (Modulo) mit den üblichen Vorrangregeln verknüpft werden.
Strings können mit dem Operator „.“ aneinandergereiht werden.

```
let s:a = "Un" . "sinn"
```

Aus der Sprache C wurde der Entscheidungsoperator „?:“ entnommen.

```
let s:a = exists("s:a") ? (5 * s:a) : 0
```

An Vergleichsoperatoren stehen „==“ (Gleichheit) „!=’“ (Ungleichheit), „>“, „<“, „>=’ und „<=“ zur Verfügung.
Wird ein String mit einem numerischen Wert verglichen, wird der String zu einer Zahl konvertiert. Wenn der String keine Zahl darstellt, ist das Ergebnis der Konvertierung 0.
Strings können darauf getestet werden, ob sie einen regulären Ausdruck enthalten (Operator „=~“) oder nicht enthalten (Operator „!~“).

```
if s:text =~ ’\.$’
  echo "Es liegt ein ganzer Satz vor, der mit endet."
endif
```

Bei Stringvergleichen und Test auf Erfüllung regulärer Ausdrücke wird der Wert der Option „ignorecase“ berücksichtigt, um zu entscheiden, ob Groß- u. Kleinschreibung ignoriert wird oder nicht.
Wird dem Operator ein „?“ angefügt, wird Groß- u. Kleinschreibung ignoriert (Großbuchstabe gleich Kleinbuchstabe), bei „#“ wird sie berücksichtigt.

```
if s:text ==? "rennen"
  ...
endif
```

9.3.3 Bedingte Anweisungen

Mit

```
if Bedingung
    Anweisungen
endif
```

werden die Anweisungen nur ausgeführt, wenn die gegebenen Bedingungen erfüllt sind.

Mit

```
if Bedingung
    Anweisungen1
else
    Anweisungen2
endif
```

werden die Anweisungen1 nur ausgeführt, wenn die gegebenen Bedingungen erfüllt sind, ansonsten Anweisungen2.

9.3.4 Schleifen

Mit

```
while Bedingung
   Anweisungen
endwhile
```

werden die Anweisungen solange ausgeführt, wie die Bedingung erfüllt ist. Der Test auf die Erfüllung der Bedingung erfolgt jeweils vor dem Ausführen der Anweisungen. Es handelt sich also um eine abweisende Schleife, d.h. wenn bereits beim ersten Test die Bedingung nicht erfüllt ist, werden die Anweisungen überhaupt nicht ausgeführt.

9.3.5 Funktionen

Wie auch andere modular aufgebaute Programmiersprachen kennt auch vim die Möglichkeit, Funktionen zu definieren und diese dann an anderer Stelle aufzurufen.
Der Aufruf kann dann entweder über

```
call search("Date: ", "W")
```

erfolgen (hier wird die Funktion „search“ mit den Strings „Date:“ und „W“ als Parametern aufgerufen) oder in der Berechnung von Ausdrücken

```
let repl = substitute(line, "\a", "*", "g")
```

9.3.6 Eigene Funktionen definieren

Eigene Funktionen werden mit

```
function name(Argumente)
  ...
endfunction
```

definiert, z.B.

```
function Max(arg1,arg2)
  if(a:arg1 > a:arg2)
    let larger = a:arg1
  else
    let larger = a:arg2
  endif
  return larger
endfunction
```

Der Funktionsname muß mit einem Großbuchstaben beginnen. Auf die Argumente wird mit dem Prefix „a:“ Bezug genommen. Alle Variablen, die innerhalb der Variablen definiert werden, sind lokale Variablen der Funktion, wenn nicht ein vorangestellter Prefix dies ändert.
Wird „function!“ anstelle von „function“ verwendet, wird eine evtl. bereits vorhandene gleichnamige Funktion stillschweigend überschrieben.

9.3.7 Funktionen mit einem Zeilenbereich

Wird eine Funktion

```
function Wort_zaehlung() range
  let n = a:firstline
  let count = 0
  while n <= a:lastline
    ...
    let n = n + 1
  endwhile
  echo "Es wurden " . count . " Worte gefunden."
endfunction
```

mit dem Kommando

```
10,20call Wort_zaehlung()
```

aufgerufen, so wird der angegebene Zeilenbereich in Form der Argumente „a:firstline“ und „a:lastline“ übergeben.
Wird eine Funktion ohne das Schlüsselwort „range“ definiert und mit der Form

```
10,20call Wort_zaehlung()
```

aufgerufen, so wird nacheinander jeweils für alle Zahlen von 10 bis 20 der Cursor in die entsprechende Zeile gesetzt und die Funktion aufgerufen.

9.3.8 Vordefinierte Funktionen

9.3.9 vim-Anweisungen ausführen

Mit

`execute` *Ausdruck*

wird der Ausdruck (String-Ausdruck) so ausgeführt, als wäre er im Kommando-Mode eingegeben worden, z.B. schreibt

```
execute "w"
```

den aktuellen Puffer in die zugehörige Datei.

9.4 Folding

9.4.1 Übersicht

Als Folding (Faltung) bezeichnet man, wenn im Editor-Fenster nicht der gesamte Text zu sehen ist sondern Textteile eingefaltet werden.
Im Editor-Fenster ist dann nur eine Markierung mit einer Kurzbeschreibung zu sehen.
Auf Wunsch kann der gefaltete Text hervorgeholt und bearbeitet werden.
Die Faltungen können dabei hierarchisch gestaffelt werden, so daß gefalteter Text weitere Faltungen enthalten kann.
Damit kann für mehr Übersicht im Editor-Fenster gesorgt werden.

9.4.2 Kriterien für die Faltung

Wenn die Faltungsstufe höher oder niedriger geschalten wird (in etwa vergleichbar mit Hinein- und Hinauszoomen bei Graphikprogrammen) kann nach verschiedenen Kriterien festgelegt werden, welcher Text und wieviel Text gefaltet bzw. hervorgeholt wird.
Folgende Verfahren stehen zur Verfügung:

- Nutzerdefinierte Faltungsbereiche (marker)
  In den Text werden Marken eingebracht, die Anfang und Ende von Faltungsbereichen anzeigen.

- Syntax-Einrückungen (indent)
  Es werden Bereiche gefaltet, die durch die Syntax-Einrückung gleich tief eingerückt sind. Dies bietet sich für Quelltexte von Programmiersprachen an.
  Voraussetzung hierfür ist, daß „shiftwidth“ gesetzt ist, damit die Einrückungstiefe richtig erkannt wird.

- Syntax-Elemente
  In den Syntax-Files für Quelltexte kann für Einträge die Option „fold“ angegeben werden, dann kann vim anhand dieser Angaben Faltungen durchführen.
  Das Anpassen der Syntax-Dateien ist allerdings keine einfache Sache.

9.4.3 Faltung über Marker

Im Beispiel soll der Text aus Datei „x.c“ enstprechend Abb. 70 so bearbeitet werden, daß er sinnvoll gefaltet werden kann. Dabei werden zwei Hierarchiestufen einge-

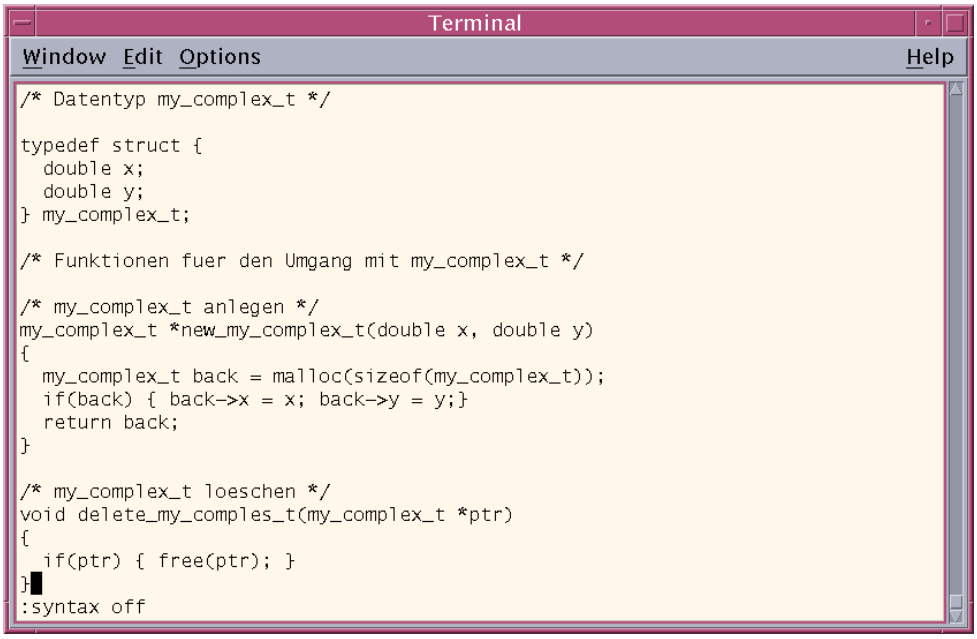

Abbildung 70: Original-Quelltext

führt:

- Datentypen
  - my_complex_t
- Funktionen
  - my_complex_t anlegen
  - my_complex_t löschen

Der Anfang für jeweils ein Gebiet der Hierarchieebene 1 wird mit einem Kommentar markiert, der „{{{1“ enthält, der beendende Kommentar enthält „}}}1“.
Der Anfang für ein Gebiet der Hierarchieebene 2 wird mit einem Kommentar markiert, der „{{{2“ enthält, der beendende Kommentar enthält „}}}2“. Weitere Hierarchieebenen sind prinzipiell möglich.
Abb. 71 auf der nächsten Seite zeigt das Ende des variierten Quelltextes. Wird nun der Cursor nach außerhalb der markierten Bereiche gebracht und im Auftextmodus `zm` eingegeben, so wird zum maximalen Faltungslevel übergegangen, d.h. es wird soviel Text weggefaltet wie möglich, siehe Abb. 72 auf Seite 88. Gibt man im Auftextmodus `zr` ein, wird das Faltungslevel um eins reduziert, d.h. die erste Hierarchieebene wird aufgelöst, siehe Abb. 73 auf Seite 88. Bewegt man im Auftextmodus

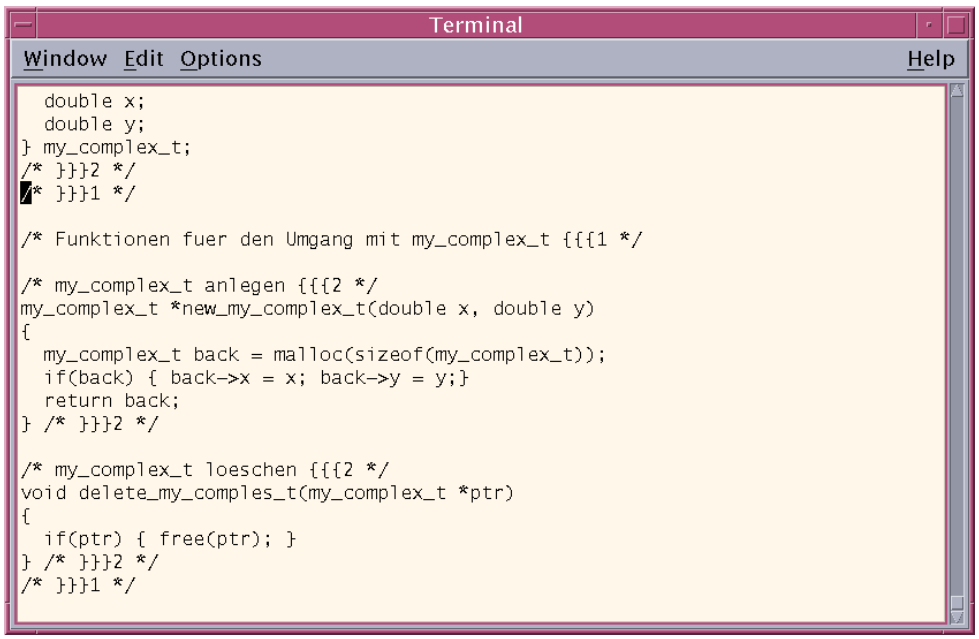

Abbildung 71: Quelltext mit Markern

den Cursor auf eine Faltungsmarkierung (siehe Abb. 74 auf Seite 89) und gibt dann `zo` ein (open), so wird nicht das ganze Dokument um eine Hierarchieebene ausgefaltet sondern nur die angewählte Faltung, siehe Abb. 75 auf Seite 89. Bewegt man den Cursor in einen Faltungsbereich hinein und gibt im Auftextmodus `zc` (close) ein, wird der entsprechende Bereich wieder eingefaltet.

9.4.4 Automatische Ein- und Ausfaltung

Die Anweisung

```
set foldopen=all
```

bewirkt ein automatisches Ausfalten, wenn eine Faltungsmarkierung betreten wird (d.h. wenn der Cursor auf die Faltungsmarkierung gebracht wird).
Mit

```
set foldclose=all
```

wird veranlaßt, daß Faltungsbereiche automatisch eingefaltet werden, wenn der Cursor aus dem Faltungsbereich herausbewegt wird.

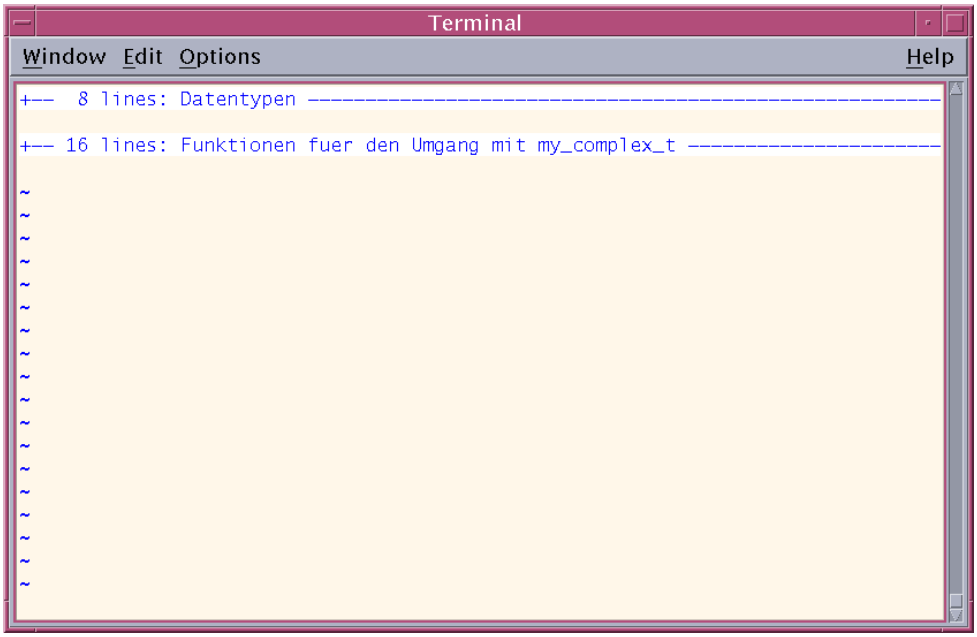

Abbildung 72: Text mit maximaler Faltung

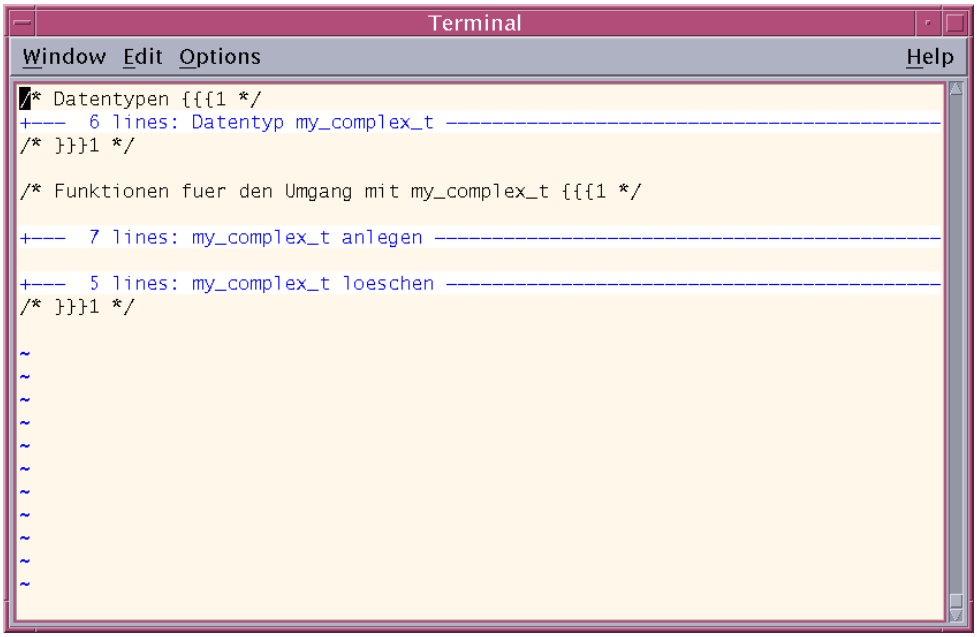

Abbildung 73: Nach Auflösen der ersten Hierarchieebene

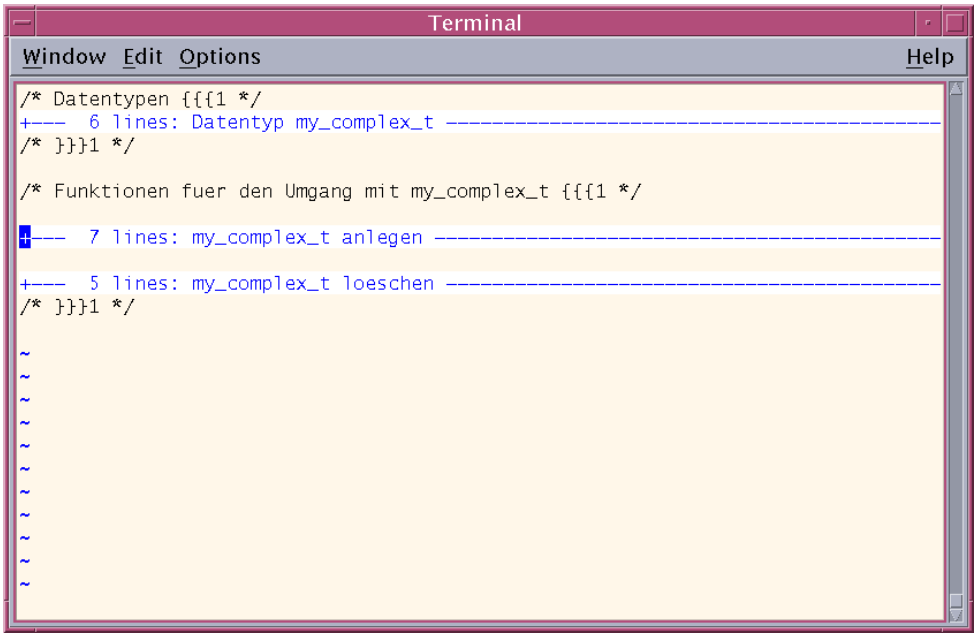

Abbildung 74: Positionierung auf Faltungsmarkierung

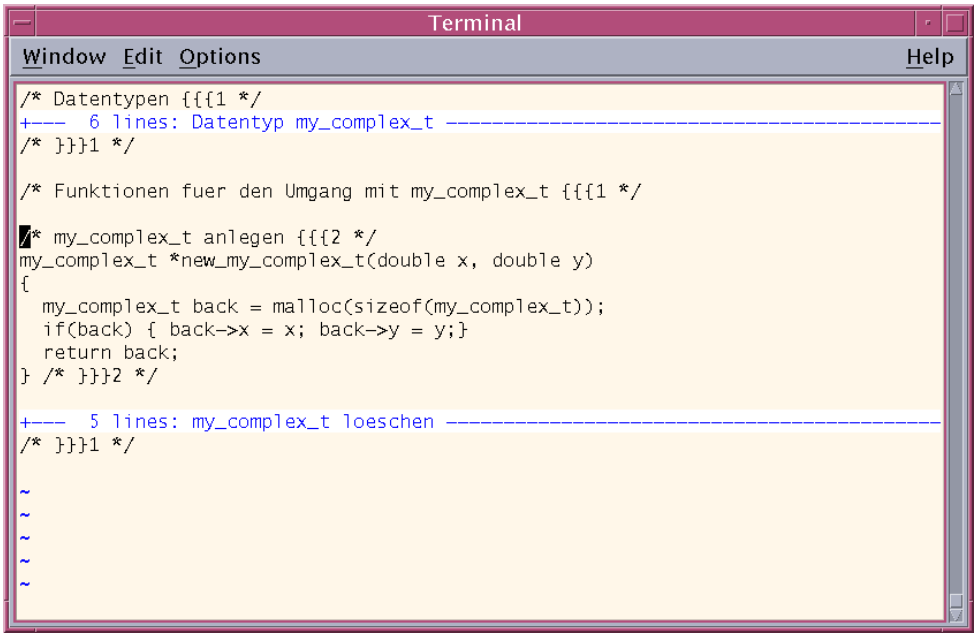

Abbildung 75: Text nach dem Ausfalten

9.5 Splitting - Mehrere Teilfenster

9.5.1 Übersicht

In vim kann ein Programmfenster in mehrere Teilfenster aufgeteilt werden. In diesen Teilfenstern können entweder unterschiedliche Stellen derselben Datei oder verschiedene Dateien bearbeitet werden.

9.5.2 Teilfenster öffnen

Neues Teilfenster mit derselben Datei Wird bereits eine Datei bearbeitet, kann mit dem Kommando

```
split
```

das Fenster geteilt werden. In beiden Teilfenstern wird dann dieselbe Datei bearbeitet, siehe Abb. 76 und 77 auf der nächsten Seite.

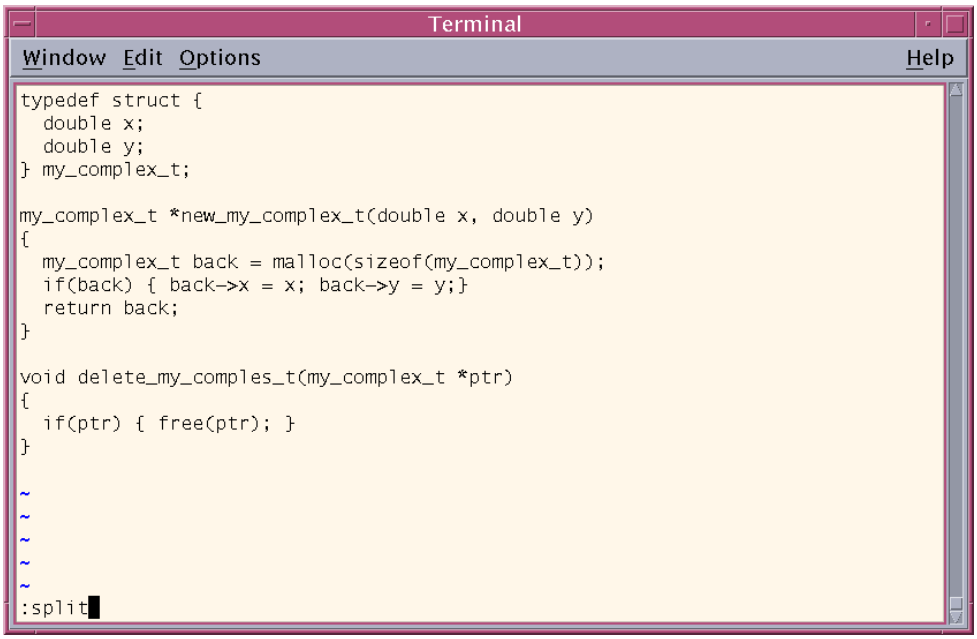

Abbildung 76: Fenster vor dem Splitting

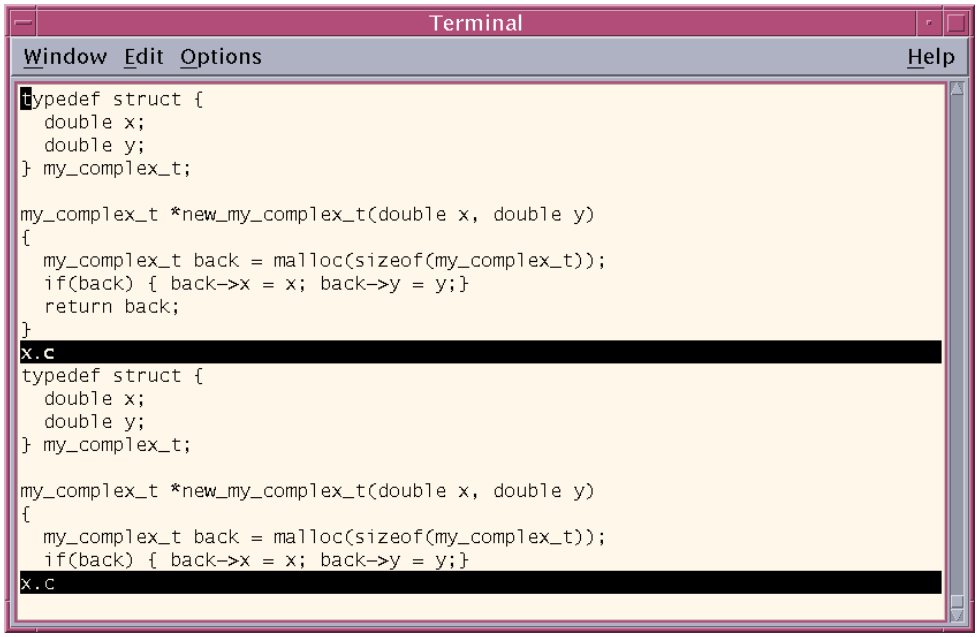

Abbildung 77: Fenster nach dem Splitting

Neues Teilfenster mit anderer Datei Soll in dem neuen Teilfenster eine andere Datei bearbeitet werden, wird er im split-Kommando mit angegeben, siehe Abb. 78 und 79 auf der nächsten Seite.

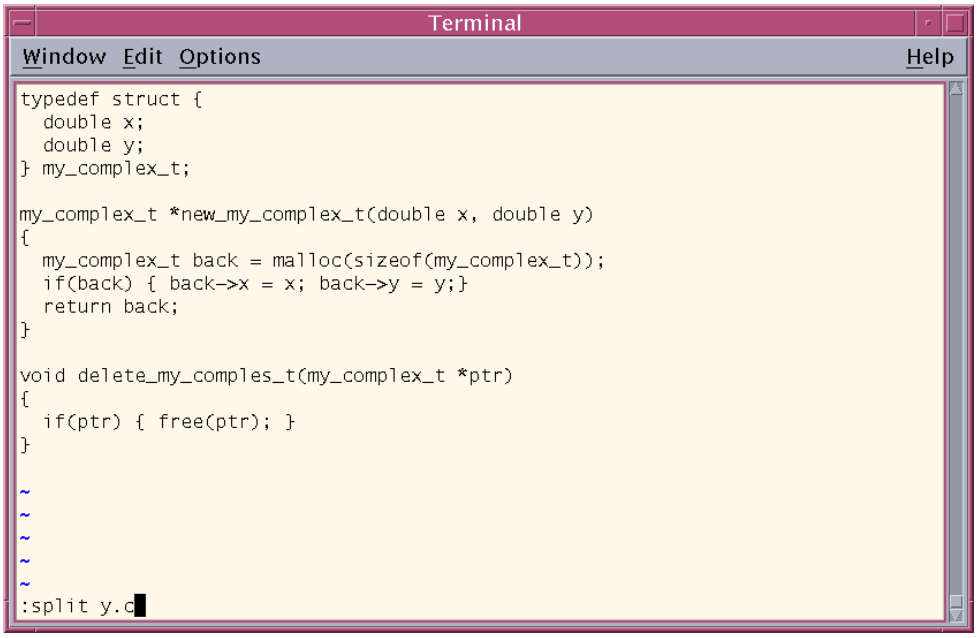

Abbildung 78: Fenster vor dem Splitting

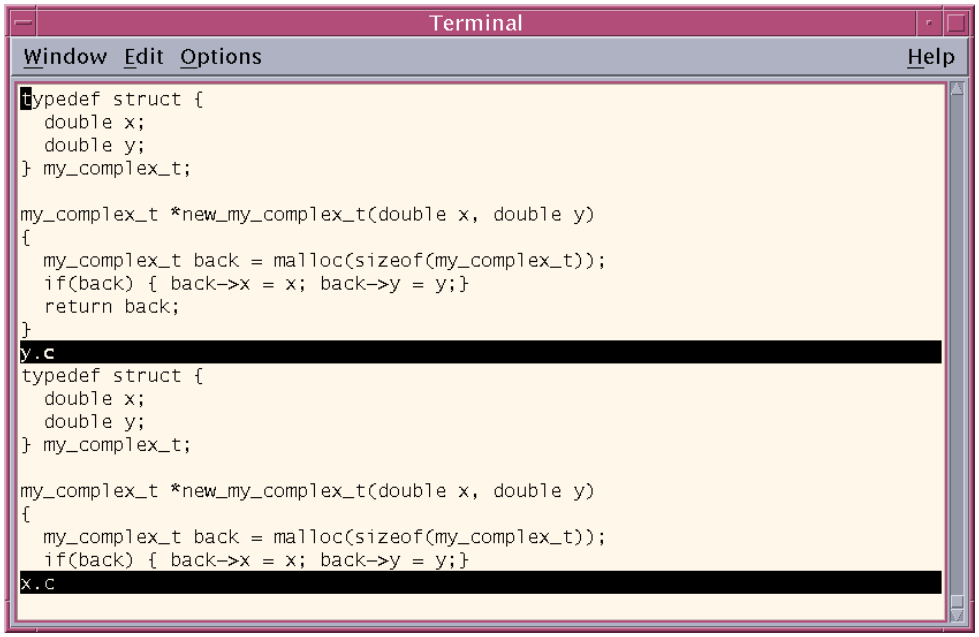

Abbildung 79: Fenster nach dem Splitting

Vertikales Splitting Mit dem split-Kommando erfolgt ein horizontales Splitting, d.h. es entstehen breite Fenster übereinander.
Wird stattdessen das Kommando

```
vsplit
```

benutzt, erfolgt vertikales Splitting, es entstehen schmale Fenster nebeneinander.

Größe für Teilfenster festlegen Im split-Kommando kann die Größe (Zeilenzahl) des neuen Fensters gleich mit angegeben werden, diese wird dem split-Kommando vorangestellt, z.B. als

```
5split y.c
```

Siehe Abb. 80 und 81 auf der nächsten Seite.

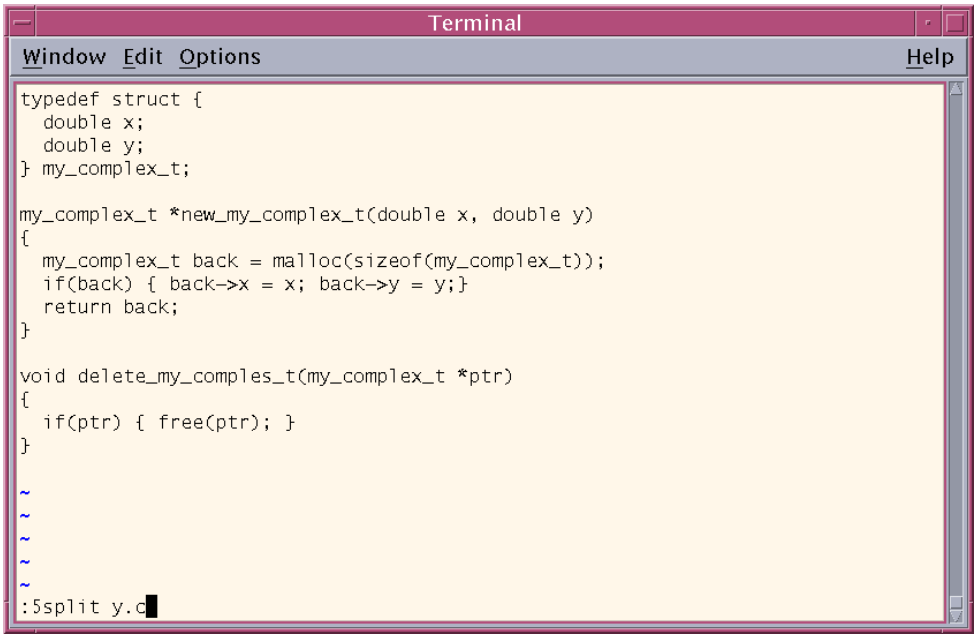

Abbildung 80: Befehl für Splitting mit Zeilenzahl

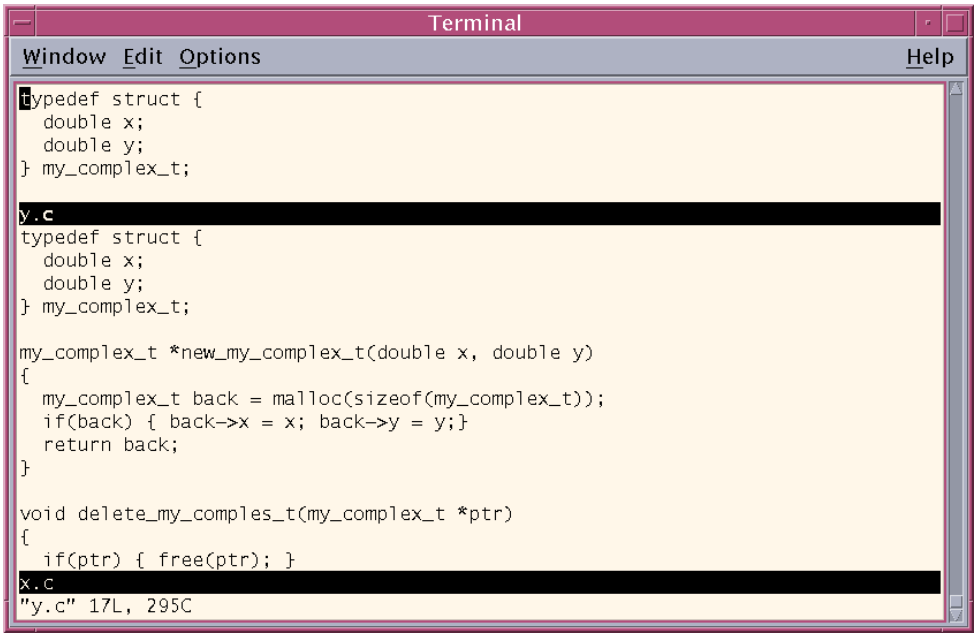

Abbildung 81: Fenster nach dem Splitting

9.5.3 Fenstergröße anpassen

Im Auftextmodus kann ein Fenster durch Eingabe der Tastenfolge

```
<CTRL-w>+
```

vergrößert werden, mit

```
<CTRL-w>-
```

wird es verkleinert.
Eine absolute Höhe kann mit *Höhe*`<CTRL-w>_` eingestellt werden, z.B.

```
5<CTRL-w>_
```

9.5.4 Zwischen den Fenstern hin- und herwechseln

Im Auftextmodus kann mit folgenden Tastenkombinationen zwischen den Fenstern hin- und hergewechselt werden:

Tabelle 3: Tastenkombinationen zum Wechseln zwischen Teilfenstern

Tastenkombination	Zielfenster
`<CTRL-w>h`	ein Fenster nach links
`<CTRL-w>j`	ein Fenster nach unten
`<CTRL-w>k`	ein Fenster nach oben
`<CTRL-w>l`	ein Fenster nach rechts
`<CTRL-w>t`	zum obersten Fenster
`<CTRL-w>b`	zum untersten Fenster
`<CTRL-w>w`	kann mehrfach benutztwerden, um nacheinander alle Fenster zu durchqueren

9.6 Fensteranordnung ändern

Mit folgenden Tastenkombinationen kann im Auftextmodus das aktuelle Fenster an eine andere Stelle gebracht werden:

Tabelle 4: Fensteranordnung ändern

Tastenkombination	Zielposition
<CTRL-w>K	ganz oben
<CTRL-w>H	ganz links
<CTRL-w>J	ganz unten
<CTRL-w>L	ganz rechts

9.6.1 Kommandos für alle Teilfenster

Folgende Kommandos wirken sich auf alle Teilfenster aus:

Tabelle 5: Kommandos für alle Teilfenster

Kommando	Wirkung
`wall`	Schreiben aller Puffer
`wqall`	Alle Puffer schreiben und schließen
`qall!`	Alle Puffer schließen ohne zu speichern

9.6.2 Teilfenster schließen

Nur aktuelle bearbeitete Datei offenlassen Um eine übersichtlichere Darstellung zu erhalten kann das Kommando

```
only
```

genutzt werden. Damit bleibt nur noch das Teilfenster geöffnet, in dem gerade gearbeitet wird, die anderen Teilfenster werden geschlossen.

Aktuelles Teilfenster schließen Ist die Arbeit in einem Teilfenster erledigt, kann dieses mit dem Kommando

```
close
```

geschlossen werden.

9.7 Vergleich zweier Dateien

Vergleichende Darstellung Mit dem Kommando `vertical diffsplit` *Dateiname* wird die aktuelle Datei mit der benannten verglichen, z.B. mittels

```
vertical diffsplit y.c
```

siehe Abb. 82 und 83 auf der nächsten Seite.
Die Unterschiede zwischen den beiden Dateien werden farblich markiert dargestellt. Gleiche Textbereiche werden größtenteils gefaltet dargestellt, um eine bessere Übersicht über die Unterschiede der Dateien zu haben. Um die geänderten Stellen wird noch ein wenig Kontext angezeigt.
Sollen die Teilfenster untereinander dargestellt werden, wird das Schlüsselwort „vertical“ weggelassen. Wird der Cursor in einem Teilfenster bewegt, wird die entspe-

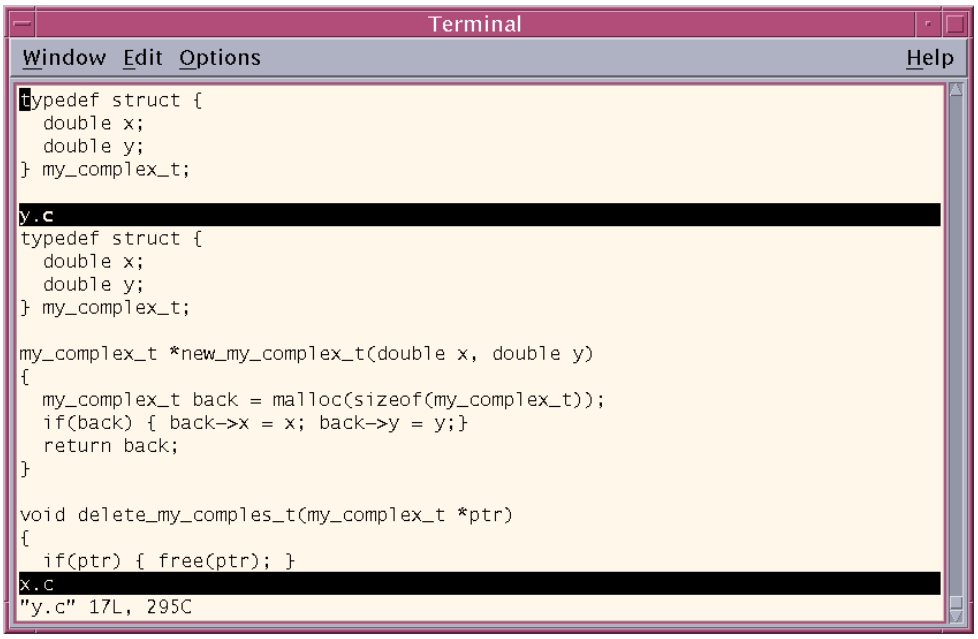

Abbildung 82: Befehl zum Vergleichen

chende Bewegung im anderen Teilfenster nachvollzogen.

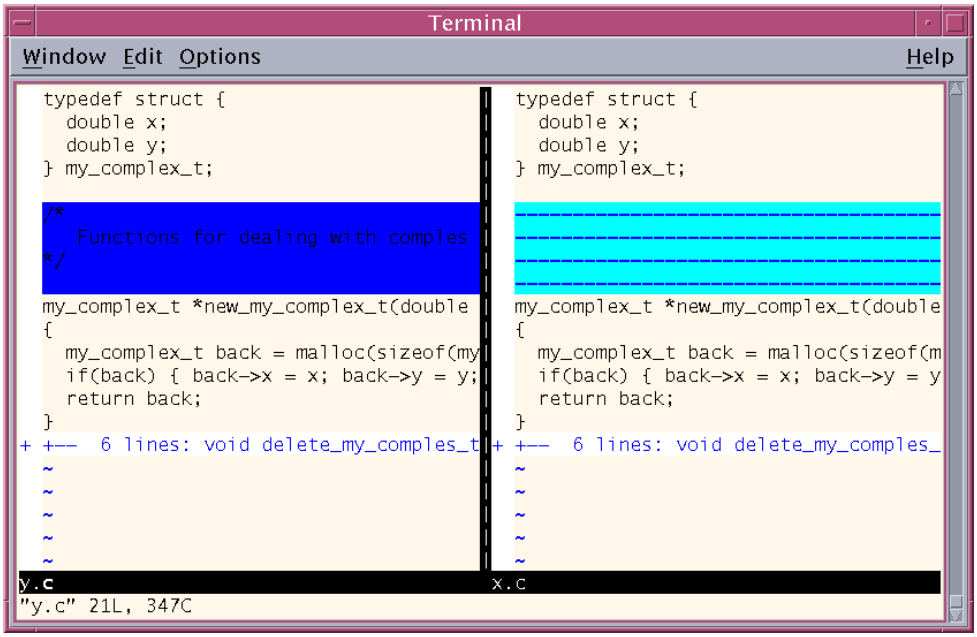

Abbildung 83: Vergleichende Anzeige

Scrollbindung aufheben und wiederherstellen Die Koppelung der Cursorbewegungen in beiden Teilfenstern ist mitunter lästig, da bei stark unterschiedlichen Dateien mitunter große Bereiche in jeweils einem Fenster übersprungen werden.
Mit dem Kommando

```
set noscrollbind
```

wird die Kopplung der Scrollpositionen aufgehoben, mit

```
set scrollbind
```

wird sie wiederhergestellt.

Änderung von einer Datei in die andere übertragen Um einen geänderten Textbereich von einer Datei in die andere zu übertragen, wird der Cursor zunächst einmal in den geänderten Textbereich hineingesetzt. Im Auftextmodus kann mit der Tastenkombination

```
dp
```

Text *vom aktuellen Teilfenster in das andere Teilfenster* übertragen werden, mit

```
do
```

erfolgt die Übertragung *vom anderen Teilfenster in das aktuelle Teilfenster*.

Index