

# Editoren

Dieser Teil beschreibt das Arbeiten mit Editoren. Dabei wird nur auf die elementaren Eigenschaften eingegangen – einen Editor lernt man nur durch Editieren.

Erläutert wird der Gebrauch von

- vi
- kedit

## Inhalt

1. vi .....	3
1.1. Wesentliche Designprinzipien und Grundlagen .....	3
1.2. Beispielsitzung .....	5
1.2.1. Befehle zum Beenden des vi .....	8
1.3. Funktionen zur Textbearbeitung .....	9
1.3.1. Kommandos im Kommandomodus .....	9
1.3.2. Die Kommandos im Last Line Modus .....	10

1.4. Suchen und Ersetzen .....	11
1.5. Die Verbindung zur Außenwelt .....	12
1.6. Makros und Abkürzungen .....	13
1.7. Aufrufoptionen des vi.....	15
1.8. Zusammenfassung der wichtigsten vi Kommandos.....	18
2. kedit.....	26

# 1. vi

Der vi (sprich vi-ei) ist als Standard-Editor auf allen Unix Systemen verfügbar. Er ist sehr leistungsfähig, aber gewöhnungsbedürftig, da er mit mehreren Modi arbeitet.

Obwohl beim Programmieren heute normalerweise mit integrierten Programmierumgebungen gearbeitet wird, sollte der elementare Umgang mit dem vi von jedem beherrscht werden. Dies ist erforderlich, da oftmals beim Einrichten von Anwendungen (auch einer Programmierumgebung) aber auch beim Installieren von entwickelten Softwarepaketen, Konfigurationsfiles u.ä. editiert werden müssen und der vi der einzig verfügbare Editor ist.

## 1.1. Wesentliche Designprinzipien und Grundlagen

Der vi kennt drei Modi:

### 1. Der Kommando Modus (command state, visual mode)

Dies ist der Modus, in den man zu Beginn einer Editiersitzung gelangt und in den man aus den anderen Modi zurückkehrt.

In diesem Modus werden alle **Tastatureingaben** als **Kommandos** interpretiert.

### 2. Der Insert Modus

Dies ist die Betriebsart, in der **Text ein- oder angefügt**. Man erreicht diesen Modus durch die Kommandos a, A, i, I, o, O, s, S R, c und verlässt ihn durch ESC.

### 3. Der Last Line Modus

In diesen Modus kommt man durch Eingabe von :, /, ? oder ! aus dem Kommando Modus heraus. Dabei wird durch : der Comamnd State des **ex** Editors (Schwester des vi) er-

reicht. Hier sind umfangreiche Textmanipulationen möglich (z.B. Suche-Ersetzen). Nach Beendigung des Modus ist man wieder im Kommando Modus.

Der vi kann wie folgt charakterisiert werden:

- Mit dem vi kann man ASCII Texte (7 oder 8 Bit ASCII) bearbeiten.
- Der vi arbeitet auf einer internen Darstellung des Textfiles im Verzeichnis /tmp (konfigurierbar), die beim Zurückschreiben (Speichern) wieder in lesbare Form gebracht wird.
- Der vi bearbeitet maximal 2 Files gleichzeitig, man kann beim Aufruf aber beliebig viele Dateien angeben, zwischen den dann umgeschaltet werden kann.
- Die letzten 9 gelöschten Texte können wieder verfügbar gemacht werden.
- Es gibt so viele frei setzbare Marken bzw. Textregister, wie Kleinbuchstaben verfügbar sind.
- Jedes Sonderzeichen kann im Text verwendet werden.
- Die Bedienung ist per Makros elementar konfigurierbar.
- Der vi expandiert im Insert Modus definierbare Textabkürzungen zu Volltexten.
- Der vi ist terminalunabhängig, er ist auf Basis der Termcap Mechanismus implementiert.
- Im vi können beliebige Shell-Skripten aufgerufen werden, es kann zum Betriebssystem verzweigt werden.

- Alle Unix Filter sind auf Textbereiche anwendbar.
- Durch eine Konfigurationsdatei ist er benutzerindividuell anpassbar.

**Der vi ist einer der schnellsten und mächtigsten Editoren, aber in der Handhabung gewöhnungsbedürftig!**

## **1.2. Beispielsitzung**

Der vi wird aufgerufen durch (hier mit neu zu erstellender Datei dat):

```
$ vi dat
```

```
|
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
"dat" [New File]
```

```
0,0-1
```

Nun meldet der vi in der letzten Zeile, dass die Datei neu erstellt wird. Dies ist die Zeile, in der Last Line Kommandos eingegeben werden.

Jede noch nicht beschriebene Zeile ist am Anfang mit ~ markiert.

Nun ist man im Kommandomodus. Nach Eingabe des Kommandos i (Einfügen) gelangt man in den Insert Modus und kann Texte eingeben.

```
Dies ist die erste Zeile.
```

```
Dies die 2.
```

```
Letzte Zeile
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
-- INSERT --
```

```
3,13
```

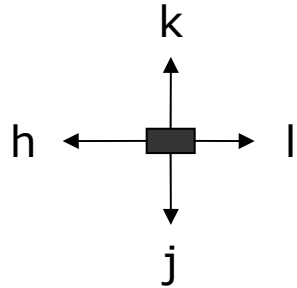
Mit ESC gelangt man wieder in den Kommandomodus.

Durch

**:wq**

wird in den Last Line Modus verzweigt und die Datei geschrieben (w) und der vi verlassen (q).

Im Kommandomodus kann man den Cursor mit den Pfeiltasten oder mit den Tasten **h, j, k, l** bewegen:



Oder z.B. wortweise rückwärts (**w**) oder vorwärts (**w**).

Weiter Kommandos zum Positionieren des Cursors sind:

- e** Ende des nächsten Wortes
- ^** Begin der Zeile
- \$** Ende der Zeile
- (** Satzanfang
- )** Satzende
- {** Abschnittsanfang
- }** Abschnittsende

### 1.2.1. Befehle zum Beenden des vi

**ZZ** Beenden mit sichern (nur gültig wenn Dateiname bekannt)

**: w Dateiname** Sichern

- : wq**                    Beenden mit sichern
- : x**                      Beenden mit sichern
- : q**                      Beenden (nur wenn vorher gesichert wurde)
- : q!**                     Beenden ohne sichern

### **1.3. Funktionen zur Textbearbeitung**

Texte können kopiert, gelöscht und verschoben werden. Dazu stehen zwei Varianten zur Verfügung: eine im Last Line Modus und eine im Kommandomodus.

#### **1.3.1. Kommandos im Kommandomodus**

Ein Textbereich (Cursorposition, Zeile, Zeilenbereich, Marken, ...) kann

<b>d</b> (elete)	gelöscht
<b>y</b> (ank)	gemerkt

werden.

Die gelöschten oder gemerkten Texte können durch

**p**(ut)

an die aktuelle Position eingefügt werden.

Beispiele:

**yw** merkt Text bis zum nächsten Wort  
**y3w** merkt 3 Worte  
**yy** merkt ganze Zeile (Verdopplung->Zeile)  
**y)** merkt bis zum Satzende  
**"ay)** merkt Text bis zum Satzende in Register a

### 1.3.2. Die Kommandos im Last Line Modus

Ein Textbereich kann

**co**(py) kopiert  
**m**(ove) verschoben  
**d**(elete) gelöscht  
**y**(ank) gemerkt

werden.

Die gelöschten oder gemerkten Texte können durch

**p**(ut)

an die aktuelle Position eingefügt werden.

Beispiele:

<b>3,5d</b>	Zeilen 3 bis 5 werden gelöscht
<b>3,5m.</b>	Zeilen 3 bis 5 werden an die aktuelle Position verschoben
<b>3,5co1</b>	Zeilen 3 bis 5 werden hinter Zeile 1 kopiert.

## 1.4. Suchen und Ersetzen

Zum Suchen und Ersetzen existieren im Kommandomodus Kommandos:

- / Vorwärtssuche
- ? Rückwärtssuche

Die zu suchende Zeichenkette kann durch einen regulären Ausdruck beschrieben werden. Das Zeichen „\`\`“ hebt dabei die Bedeutung des darauf folgenden Zeichens auf.

Dabei bedeuten:

<code>^</code>	Beginn der Zeile
<code>\$</code>	Ende der Zeile
<code>.</code>	beliebiges Zeichen
<code>*</code>	beliebige Zeichenfolge (Sequenz)
<code>\&lt;</code>	Wortbeginn
<code>\&gt;</code>	Wortende
<code>[abc]</code>	passt auf a, b und c (Alternativen)
<code>[^abc]</code>	passt auf alles außer a, b, oder c
<code>[a-g]</code>	passt auf alle Kleinbuchstaben von a bis g

Beispiele:

<b>/text</b>	sucht Zeichenfolge „text“
<b>/text/+2</b>	sucht Zeichenfolge „tex“, positioniert 2 Zeilen weiter
<b>/^text</b>	sucht nach „text“ an Zeilenanfang
<b>/text</b>	sucht nach „text“ am Zeilenende
<b>/^\$</b>	sucht nach Leerzeile
<b>/^a\$</b>	sucht Zeile, die nur aus Zeichen „a“ besteht
<b>/text\&gt;</b>	sucht „text“ am Wortende
<b>/.ein</b>	sucht nach beliebigem Zeichen, auf das String „ein“ folgt
<b>/[Hh]allo</b>	sucht nach „Hallo“ oder „hallo“

Ersetzoperationen werden durch **s**(ubstitute) im Last Line Modus initiiert. Die Form ist:

<Bereich>**s**/suchtext/ersetzungstext/Optionen

Beispiele:

<b>s/text1/text2</b>	ersetze in aktueller Zeile erstes Auftreten
<b>2,5s/text1/text2</b>	ersetze in Zeilen 2 bis 5 erste Auftreten
<b>%s/alt/neu</b>	im ganzen Text wird „alt“ durch „neu“ ersetzt
<b>s/text1/text2/g</b>	ersetze jedes Auftreten
<b>s/text1/text2/g</b>	ersetze jedes Auftreten mit Bestätigung
<b>&amp;</b>	wiederhole letztes Kommando

## 1.5. Die Verbindung zur Außenwelt

Im Last Line Modus kann durch

**:!<Shell Kommando>**

jedes Shellkommando aufgerufen werden.

Durch

**:sh**

verzweigt man zur Shell, wo man mit **exit** wieder zur vi Sitzung zurückkehrt.

Eine interessante Möglichkeit ist die Anwendung von Unix Filtern auf vi-Textbereiche.

Beispiele:

!}sort	sortiert bis zum nächsten Absatz
!)date	fügt Datum ein
!)man cp	fügt Manual Seiten von cp ein
!}wc -w	zählt Worte des Absatzes

## **1.6.Makros und Abkürzungen**

Im vi stehen zwei Möglichkeiten bereit, Makros zu definieren.

Befehlssequenzen, die mehrfach benutzt werden sollen, können in ein Textregister abgelegt werden. Dabei sind Kommandomodus- und Last Line Kommandos erlaubt. Eventuell notwendige „RETURN“ werden durch „CTR V“ entwertet.

Vorgehen:

1. Die Befehlssequenz wird als Textstring editiert.
2. Den Sting mit y oder d in eine Textregister puffern (z.B. a)
3. Das Makro durch @a aufrufen

Beispiel (sort soll mittels @s aufrufbar sein):

1. **!}sort -r** in aktuelle Zeile schreiben
2. **"sdd** dadurch wird aktuelle Zeile gelöscht und in Textregister s abgelegt
3. **@s** kann nun das Sortieren ausführen

Die zweite Möglichkeit besteht darin, durch das Kommando

**map** Wort Kommando

im Last Line Modus eine Befehlssequenz an ein Wort (oder Taste) zu binden.

Beispiele:

- |   |   |
|---|---|
| <code>:map s ZZ</code>                    | s wird Kommando ZZ (Schreiben und Verlassen       |
| <code>:map #1 :!man vi^V&lt;CR&gt;</code> | Funktionstaste F1 wird mit man page von vi belegt |
| <code>:map ^[A k</code>                   | Taste <Cursor up> wird mit k belegt               |

Die Zeichenketten dürfen nicht länger als 10 Zeichen sein und müssen innerhalb von einer Sekunde eingegeben werden.

Makros, die auch im Insert Modus verwendet werden sollen, sind mit **map!** zu definieren.

Sondertasten, wie <ESC> oder <CTR> sind nach einem vorangestellten <CTR> V einzugeben.

**unmap** hebt die Definition wieder auf.

Weiterhin sind Wortabkürzungen (Kommando **ab**) möglich.

Beispiele:

**ab** Mit freundlichen Grüßen

**ab** uk    Unix Kurs

**unab** hebt diese Definition auf.

## 1.7. Aufrufoptionen des vi

Der vi kann mit Optionen aufgerufen werden, gefolgt von einer Liste mit Dateien. Die wichtigsten Optionen sind:

- r**(ecovery) Bei Systemabsturz wird eine abgebrochene Sitzung soweit restauriert, dass maximal eine Zeile verloren geht.
- x** Der editierte Text wird verschlüsselt abgespeichert. Zum Lesen der Datei mit dem vi muss der Schlüssel angegeben werden.

Zur Initialisierung kann eine Menge von Makros, Abkürzungen und Optionen in der Datei **.exrc** im Home-Verzeichnis definiert werden.

Optionen werden im Last Line Modus mit

**se** <Option>

gesetzt und mit

**se no**<Option>

nicht gesetzt.

Einige Optionen des vi sind:

**ai** Auto indent

**nu** Number

**wrap** Wordumbruch

**ts** Tab Space

Ein Beispiel für eine **.exrc** Datei ist:

```
$ cat .exrc
" automatisch einruecken
set noautoindent

" suchen case-insensitiv
set ignorecase

" Koordinatenanzeige aktivieren
set ruler

" shell to start with !
" set shell=sh

" zeige passende klammern
set showmatch

" anzeige INSERT/REPLACE/...
set showmode

" einrueckung
set shiftwidth=4
set tabstop=4

" Tastatur-Belegung fuer diverse vi's
" Autor: Werner Fink <werner@suse.de>
" Version: 20.05.1997
```

```
" keys in display mode
map ^[OA k
map ^[[A k
map ^[OB j
map ^[[B j
map ^[OD h
map ^[[D h
map ^? h
map ^H h
...
$
```

## 1.8.Zusammenfassung der wichtigsten vi Kommandos

```
$ man vi
```

```
VIM(1)
```

```
VIM(1)
```

## NAME

vim - Vi IMproved, a programmers text editor

## SYNOPSIS

```
vim [options] [file ..]
vim [options] -
vim [options] -t tag
vim [options] -q [errorfile]
```

ex

view

gvim gview

rvim rview rgvim rgview

## DESCRIPTION

Vim is a text editor that is upwards compatible to Vi. It can be used to edit any ASCII text. It is especially useful for editing programs.

There are a lot of enhancements above Vi: multi level undo, multi windows and buffers, syntax highlighting, command line editing, filename completion, on-line help, visual selection, etc.. See `":help vi_diff.txt"` for a

summary of the differences between Vim and Vi.

While running Vim a lot of help can be obtained from the on-line help system, with the `":help"` command. See the ON-LINE HELP section below.

Most often Vim is started to edit a single file with the command

```
vim file
```

More generally Vim is started with:

```
vim [options] [filelist]
```

If the filelist is missing, the editor will start with an empty buffer. Otherwise exactly one out of the following four may be used to choose one or more files to be edited.

file ..      A list of filenames. The first one will be the current file and read into the buffer. The cursor will be positioned on the first line of the buffer. You can get to the other files with the `":next"` command. To edit a file that starts with a dash, precede the filelist with `--`.

- The file to edit is read from stdin. Commands are read from stderr, which should be a tty.

-t {tag} The file to edit and the initial cursor

1998 December 28

1

VIM(1)

VIM(1)

position depends on a "tag", a sort of goto label. {tag} is looked up in the tags file, the associated file becomes the current file and the associated command is executed. Mostly this is used for C programs, in which case {tag} could be a function name. The effect is that the file containing that function becomes the current file and the cursor is positioned on the start of the function. See ":help tag-commands".

-q [errorfile]

Start in quickFix mode. The file [errorfile] is read and the first error is displayed. If [errorfile] is omitted, the filename is

obtained from the 'errorfile' option (defaults to "AztecC.Err" for the Amiga, "errors.vim" on other systems). Further errors can be jumped to with the ":cn" command. See ":help quickfix".

Vim behaves differently, depending on the name of the command (the executable may still be the same file).

vim           The "normal" way, everything is default.

view           Start in read-only mode. You will be protected from writing the files. Can also be done with the "-R" argument.

gvim gview  
              The GUI version. Starts a new window. Can also be done with the "-g" argument.

rvim rview rgvim rgview  
              Like the above, but with restrictions. It will not be possible to start shell commands, or suspend Vim. Can also be done with the "-Z" argument.

## OPTIONS

The options may be given in any order, before or after filenames. Options without an argument can be combined after a single dash.

`+{num}` For the first file the cursor will be positioned on line "num". If "num" is missing, the cursor will be positioned on the last line.

1998 December 28

2

VIM(1)

VIM(1)

`+/{pat}` For the first file the cursor will be positioned on the first occurrence of {pat}. See `":help search-pattern"` for the available search patterns.

`+{command}`

- c {command} {command} will be executed after the first file has been read. {command} is interpreted as an Ex command. If the {command} contains spaces it must be enclosed in double quotes (this depends on the shell that is used).
- b Binary mode. A few options will be set that makes it possible to edit a binary or executable file.
- C Compatible. Set the 'compatible' option. This will make Vim behave mostly like Vi, even though a .vimrc file exists.
- d {device} Open {device} for use as a terminal. Only on the Amiga. Example: "-d con:20/30/600/150".
- e Start Vim in Ex mode, just like the executable was called "ex".
- f Foreground. For the GUI version, Vim will not fork and detach from the shell it was started in. On the Amiga, Vim is not restarted to open a new window. This option should be used when Vim is executed by a program that will

wait for the edit session to finish (e.g. mail). On the Amiga the ":sh" and ":@" commands will not work.

-F If Vim has been compiled with FKMAP support for editing right-to-left oriented files and Farsi keyboard mapping, this option starts Vim in Farsi mode, i.e. 'fkmap' and 'rightleft' are set. Otherwise an error message is given and Vim aborts.

-g If Vim has been compiled with GUI support, this option enables the GUI. If no GUI support was compiled in, an error message is given and Vim aborts.

-h Give a bit of help about the command line arguments and options. After this Vim exits.

1998 December 28

3

VIM(1)

VIM(1)

## 2. kedit

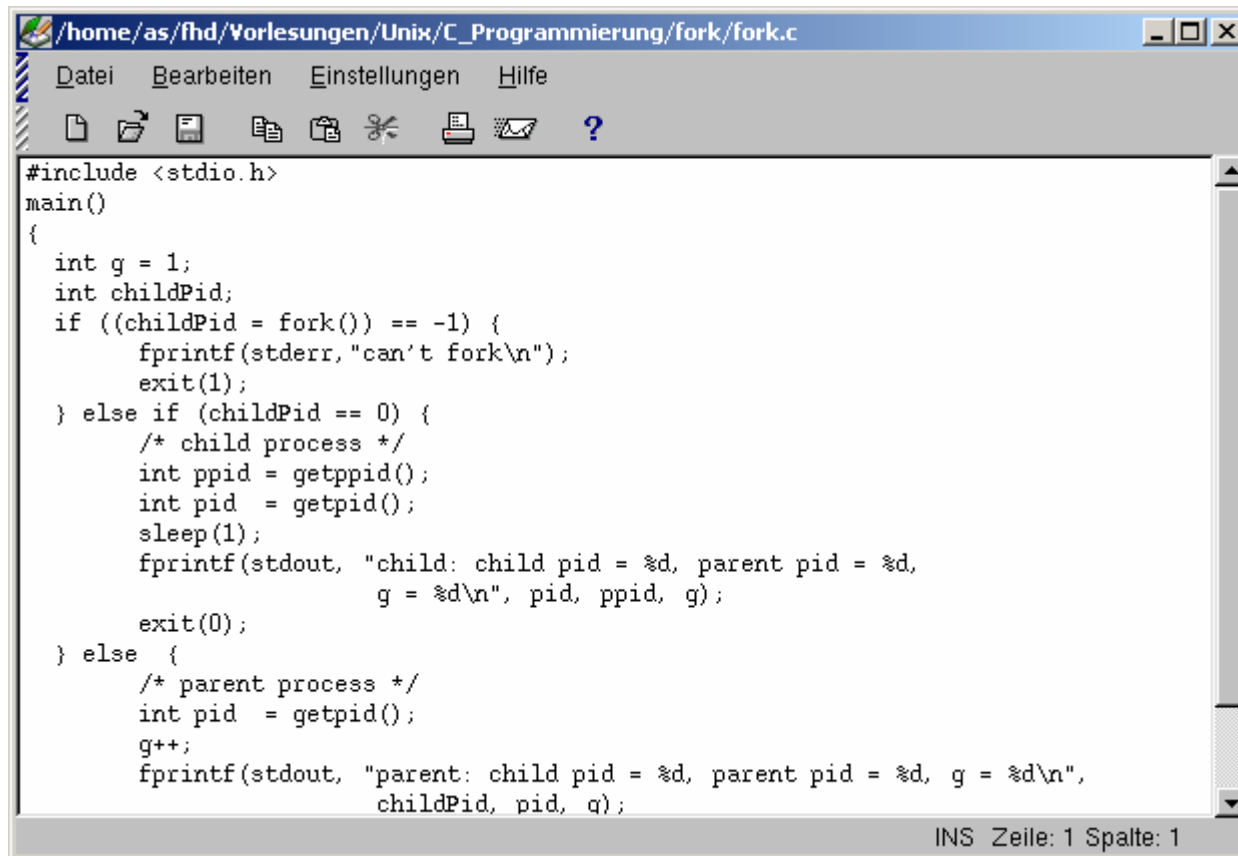
**KDE** ist eine moderne grafische Arbeitsumgebung für Unix-Computer. Sie kombiniert einfache Handhabung, umfassende Funktionalität und herausragendes grafisches Design mit den technischen Vorteilen von Unix.

**KDE** ist ein Projekt, das in jeder Hinsicht offen ist. Die Entwicklung findet im Internet statt und wird auf unseren [Mailing-Listen](#), Usenet-Newsgroups und IRC-Kanälen diskutiert.

**KDE** bietet eine ausgereifte Arbeitsumgebung für eine ständig wachsende Zahl von [Anwendungen](#). KDE stellt auch eine Entwicklungsumgebung von hoher Qualität bereit, die die rasche und effiziente Erstellung von neuen Anwendungen erlaubt.

kedit ist das „Notepad“ innerhalb KDE und wird aufgerufen durch:

```
$ kedit fork.c
```



```
#include <stdio.h>
main()
{
  int g = 1;
  int childPid;
  if ((childPid = fork()) == -1) {
    fprintf(stderr, "can't fork\n");
    exit(1);
  } else if (childPid == 0) {
    /* child process */
    int ppid = getppid();
    int pid = getpid();
    sleep(1);
    fprintf(stdout, "child: child pid = %d, parent pid = %d,
                  g = %d\n", pid, ppid, g);

    exit(0);
  } else {
    /* parent process */
    int pid = getpid();
    g++;
    fprintf(stdout, "parent: child pid = %d, parent pid = %d, g = %d\n",
              childPid, pid, g);
  }
}
```

INS Zeile: 1 Spalte: 1

Die Bedienung ist selbsterklärend.