

# gVim-Kurzanleitung

Dr.-Ing. Fritz Mehner

## Inhaltsverzeichnis

1	Der Editor <i>Vim/gVim</i>	1	10	Tastenkürzel	14
2	Betriebsarten (Modi)	2	11	Kommentierung und Formatierung	15
3	Positionierung	4	12	Navigation im Quellcode	18
4	Textveränderungen	5	13	Verwendung von Shell-Befehlen	21
5	Suchen	9	14	Weitere plug-ins	22
6	Wortergänzung	9	15	Benutzereinstellungen	23
7	Unterschiedliche Befehle	10	16	Zusätzliche Benutzungshinweise	29
8	Erstellen von <i>C/C++</i> -Programmen	12	17	Weitere Informationsquellen	29
9	Übersetzen und Ausführen	13			

## 1 Der Editor *Vim* / *gVim*

Der Editor *Vim* und seine graphische Variante *gVim* sind die leistungsfähigsten Weiterentwicklungen des klassischen *Unix*-Editors *vi*. Ein wesentlicher Vorteil dieser Editoren ist die Möglichkeit, aus einigen grundlegenden Editierbefehlen (Einfügen, Löschen, Kopieren, Austauschen, ...), zusammen mit Positionsangaben und Wiederholungsfaktoren, systematisch mächtige Editierbefehle zusammensetzen. Meist lassen sich so mit wenigen Tastatureingaben Änderungen vornehmen, für die andere Editoren mehrere Einzelbefehle benötigen. Dieses Befehlssystem ermöglicht ein außerordentlich schnelles und bequemes Arbeiten.

*Vim* und *gVim* sind ausgesprochene Programmiereditoren, die viele Möglichkeiten bieten, den Programmierer zu unterstützen. Dazu gehören die Syntax-Einfärbung für etwa 530 Programmiersprachen (oder Dialekte), reguläre Ausdrücke beim Suchen, die Möglichkeit, Shell-Programme als Textfilter zu verwenden, das Einlesen von Fehlermeldungen von Compilern und vieles mehr.

Ein Programmiereditor ist in der Regel programmiersprachenunabhängig, kann aber meist, etwa durch Zusatzmodule, einzelne Programmiersprachen hervorragend unterstützen. Damit ist er ein universell einsetzbares Werkzeug für alle anspruchsvollen Textbearbeitungen und wird zu einer Entwicklungsumgebung, die mehrere Programmiersprachen gleichzeitig unterstützt.



## 2 Betriebsarten (Modi)

Betriebsart	Zweck	Abb.
<b>normal mode</b>	Alle Eingaben wirken als Editierbefehle, sofern die entsprechenden Tastenbelegungen und Tastenkombinationen als Befehle definiert sind.	
<b>insert mode</b>	Der eingegebene Text erscheint im Puffer.	
<b>replace mode</b>	Der eingegebene Text überschreibt den Text an der Cursorposition.	
<b>visual mode</b>	Entspricht dem normal mode. Die Bewegungsbefehle (siehe unten) verändern den hervorgehobenen Textbereich. Die anderen Befehle wirken sich nur auf den hervorgehobenen Textbereich aus.	1
<b>command-line mode</b>	Dieser Modus erlaubt die Eingabe umfangreicherer Befehle in der Befehlszeile am unteren Rand des Hauptfensters. Hier können Ex-Befehle, Suchbefehle und Filterbefehle eingegeben werden.	2

Tabelle 1: Betriebsarten

<i>von</i> \ <i>nach</i>	normal	insert	replace	visual	command
normal	—	<b>[Einfg]</b>	R	v V <b>[Strg]</b> v	: / ?
insert	<b>[Esc]</b>	—	<b>[Einfg]</b>	—	—
replace	<b>[Esc]</b>	<b>[Einfg]</b>	—	—	—
visual	<b>[Esc]</b>	c C	—	—	:
command	—	:start	—	—	—

Tabelle 2: Wechsel zwischen den Betriebsarten

Die meisten Editoren sind nach dem Aufruf im Einfügemodus und ermöglichen damit die sofortige Texteingabe. Der Editor *gVim* besitzt dagegen mehrere Betriebsarten (Modi). Nach dem Aufruf befindet man sich zunächst im Normalmodus (normal mode); mit Hilfe der Taste **[Einfg]** gelangt man in den Einfügemodus. Um die weiter unten beschriebenen, mächtigen Editierbefehle zu verwenden zu können, muss gelegentlich mit Hilfe von **[Esc]** in den Normalmodus zurückgeschaltet werden.

Diese (und andere) Betriebsartenumschaltungen erscheinen demjenigen, der mit den Möglichkeiten des Editors noch wenig vertraut ist, zunächst als hinderlich. Sobald man jedoch einen Überblick über die Bedienung dieses Editors gewonnen hat, erkennt man, dass dieser kleine Nachteil durch die zur Verfügung stehenden, einzigartigen Befehlskombinationen mehr als aufgewogen wird.

Tabelle 2 zeigt die Wechsellmöglichkeiten zwischen den in Tabelle 1 erläuterten Betriebsarten. Der Cursor nimmt je nach Betriebsart eine andere Form an (siehe Tabelle 3). Alle Modi (außer dem Normalmodus) werden in der Statuszeile angezeigt.

Abbildung 1 zeigt den Editor im **Markierungsmodus** (visual mode). Der Cursor hat Blockform und der markierte Text ist dunkel unterlegt. In der Fußzeile werden der Modus und die Cursorposition angezeigt. Abbildung 2 zeigt den Editor im **Befehlsmodus** (command line mode). Der Cursor hat Blockform und steht hinter der Eingabe in der Befehlszeile am unteren Rand des Hauptfensters. Die anderen Angaben der Fußzeile sind bis zum Abschluss der Befehlseingabe nicht sichtbar.




<b>Normalmodus</b>		Der Cursor hat Blockform. In der Fußzeile wird die Cursorposition aber kein Modus angezeigt.
<b>Einfügemodus</b>		Der Cursor hat Strichform und in der Fußzeile werden der Modus und die Cursorposition angezeigt.
<b>Überschreibmodus</b>		Der Cursor hat Unterstrichform und in der Fußzeile werden der Modus und die Cursorposition angezeigt.

Tabelle 3: Betriebsarten und Cursorformen

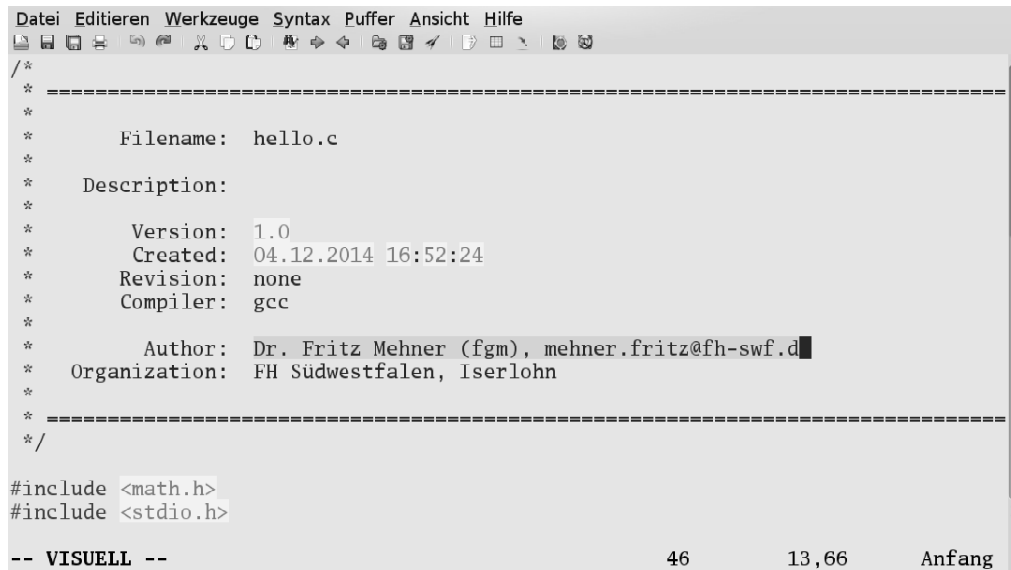


Abbildung 1: Editor im visual mode (Bereich markiert)

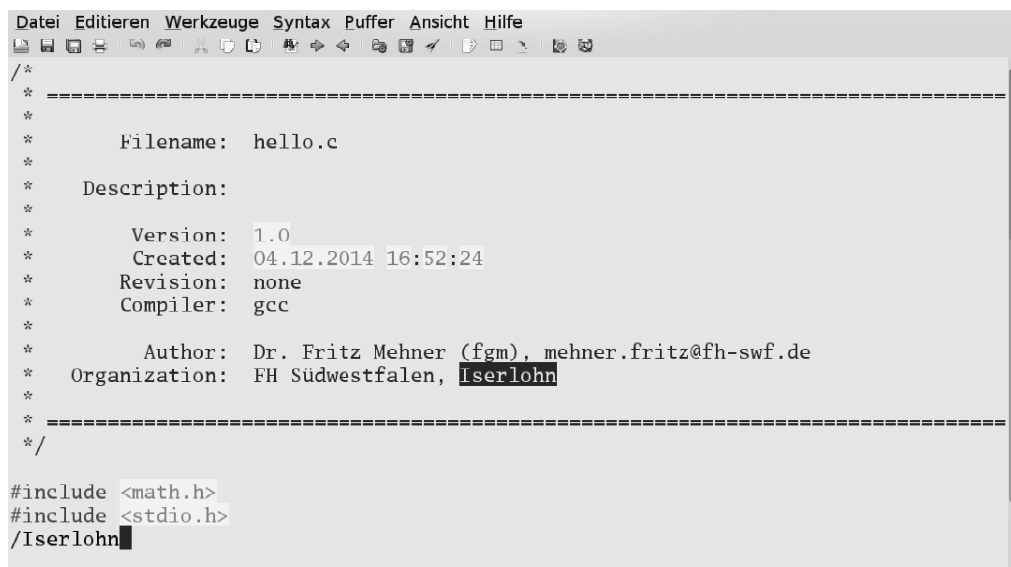


Abbildung 2: Editor im command-line mode (Befehlseingabe in der Fußzeile)

### 3 Positionierung

Zur Positionierung des Cursors stehen mehrere umfangreiche Gruppen von Positionierungsbefehlen, sogenannte „Bewegungen“ (engl. motion) zur Verfügung. Diese Bewegungen dienen außer zur *Cursorpositionierung* noch als *Angabe des Wirkungsbereiches der eigentlichen Editierbefehle*, die zur Veränderung des Textes erforderlich sind (zum Beispiel Suchen, Löschen, Ersetzen, Umwandlung in Großbuchstaben). Durch diese Verbindung sind äußerst mächtige Editierbefehle möglich.

Fast alle Positionier- und Editierbefehle können vorangestellte *Wiederholungsfaktoren* besitzen. Der Ersatzwert für einen nicht vorhandenen Wiederholungsfaktor ist eins. So springt der Cursor im Normalmodus bei Eingabe des Befehls

**w**

um ein Wort weiter, bei Eingabe des Befehls

**3w**

jedoch um drei Wörter. In den folgenden Tabellen wird die Bezeichnung *N* für diesen Wiederholungsfaktor verwendet.

#### Waagerechte Bewegungen






<i>N</i> <b>h</b>	links ( Pfeiltaste  und backspace  sind ebenfalls möglich)
<i>N</i> <b>l</b>	rechts ( Pfeiltaste  und Leertaste  sind ebenfalls möglich)
<b>0</b>	zum ersten Zeichen der Zeile (  ist ebenfalls möglich)
<b>^</b>	zum ersten Nichtleerzeichen der Zeile
<i>N</i> <b> </b>	zur <i>N</i> -ten Spalte (Ersatzwert 1)
<i>N</i> <b>f</b> <i>Zeichen</i>	zum <i>N</i> -ten Vorkommen von <i>Zeichen</i> nach rechts (find)
<i>N</i> <b>F</b> <i>Zeichen</i>	zum <i>N</i> -ten Vorkommen von <i>Zeichen</i> nach links (Find)

Tabelle 4: Waagerechte Bewegungen im Normalmodus (Auswahl)

#### Senkrechte Bewegungen



<i>N</i> <b>k</b>	<i>N</i> Zeilen nach oben; die Spaltenposition wird beibehalten, wenn die Zielzeile mindestens die gleiche Länge besitzt ( Pfeiltaste  ist ebenfalls möglich)
<i>N</i> <b>j</b>	<i>N</i> Zeilen nach unten; die Spaltenposition wird beibehalten, wenn die Zielzeile mindestens die gleiche Länge besitzt ( Pfeiltaste  ist ebenfalls möglich)
<i>N</i> <b>-</b>	<i>N</i> Zeilen nach oben zum ersten Nichtleerzeichen der Zielzeile
<i>N</i> <b>+</b>	<i>N</i> Zeilen nach unten zum ersten Nichtleerzeichen der Zielzeile
<i>N</i> <b>G</b>	zur Zeile <i>N</i> (Ersatzwert: letzte Zeile)
<i>N</i> <b>gg</b>	zur Zeile <i>N</i> (Ersatzwert: erste Zeile)

Tabelle 5: Senkrechte Bewegungen im Normalmodus (Auswahl)

### Textobjekt-Bewegungen

$N$	w	$N$ Wörter vorwärts	(word)
$N$	W	$N$ WÖRTER vorwärts	
$N$	e	vorwärts zum Ende des $N$ -ten Wortes	(end)
$N$	E	vorwärts zum Ende des $N$ -ten WORTES	
$N$	b	rückwärts zum Anfang des $N$ -ten Wortes	(begin)
$N$	B	rückwärts zum Anfang des $N$ -ten WORTES	
$N$	[ (	rückwärts zur $N$ -ten öffnenden runden Klammer; vollständige Klammerpaare (...) werden übersprungen	
$N$	[ {	rückwärts zur $N$ -ten öffnenden geschweiften Klammer; vollständige Klammerpaare {...} werden übersprungen	
$N$	] )	vorwärts zur $N$ -ten schließenden runden Klammer; vollständige Klammerpaare (...) werden übersprungen	
$N$	] }	vorwärts zur $N$ -ten schließenden geschweiften Klammer; vollständige Klammerpaare {...} werden übersprungen	
$N$	[ *	rückwärts zum $N$ -ten Beginn eines $C$ -Kommentars	
$N$	] *	vorwärts zum $N$ -ten Ende eines $C$ -Kommentars	

Wort bezeichnet eine Zeichenkette bestehend aus Buchstaben, Ziffern und Unterstrichen ohne Leerzeichen; WORT bezeichnet eine Zeichenkette aus beliebigen Zeichen ohne Leerzeichen

Tabelle 6: Textobjekt-Bewegungen im Normalmodus (Auswahl)

## 4 Textveränderungen

Die verändernden Befehle sind in den folgenden Tabellen gruppenweise zusammengefasst. Es gibt eine Reihe von Grundbefehlen, die als Tastenkürzel oder in Tastenkombinationen verwendet werden. Die Tastenkürzel ergeben sich aus den englischen Verben, die den Befehl beschreiben und sind somit leicht zu merken.

**Grundbefehle** Die Grundbefehle lauten:

<b>a</b>	<b>append</b>	<i>anhängen</i>
<b>c</b>	<b>change</b>	löschen und einfügen ( <i>ändern</i> )
<b>d</b>	<b>delete</b>	<i>löschen</i>
<b>i</b>	<b>insert</b>	<i>einfügen</i>
<b>j</b>	<b>join</b>	<i>zusammenfassen</i>
<b>o</b>	<b>open</b>	<i>öffnen</i>
<b>p</b>	<b>put</b>	Text aus dem Kopierregister <i>einfügen</i>
<b>s</b>	<b>substitute</b>	löschen und einfügen ( <i>ersetzen</i> )
<b>x</b>		zeichenweise löschen
<b>y</b>	<b>yank</b>	Text in das Kopierregister <i>herausziehen</i>

**Wiederholungsfaktor** Fast allen Befehlen kann ein Wiederholungsfaktor vorangestellt werden. Der Faktor wird hier mit  $N$  bezeichnet und ist eine Zahlenangabe. So löscht der Befehl

## Weitere Bewegungen

$N$	<b>%</b>	suche die nächste Klammer in der Zeile oder verwende die Klammer unter dem Cursor; springe zu deren Gegenstück; Klammern sind ( [ { } ] ) , sowie die Präprozessoranweisungen <code>#if</code> , <code>#ifdef</code> , <code>#else</code> , <code>#elif</code> , <code>#endif</code> und Beginn und Ende eines <i>C</i> -Kommentars <code>/* */</code>
$N$	<b>H</b>	springe zur $N$ -ten Zeile des Fensters (Home)
	<b>M</b>	springe zur Mitte des Fensters (Middle line)
$N$	<b>L</b>	springe zur $N$ -ten Zeile von unten des Fensters (Last line)

Tabelle 7: Weitere Bewegungen im Normalmodus (Auswahl)

**dd**

die Zeile, in der der Cursor steht. Der Befehl

**3dd**

löscht insgesamt drei Zeilen, beginnend mit der Zeile, in der der Cursor steht.

**Bewegung** Einzelnen Befehlen kann als Angabe ihres Wirkungsbereiches eine Bewegung nachgestellt werden. Die Bewegungen werden nachfolgend mit *Bewegung* bezeichnet. Der Befehl

**dw**

löscht den Rest des Wortes ab der Stelle an der der Cursor steht. Die Bewegung ist hier die Positionsangabe **w** für Wort. Alle oben angegebenen Positionsangaben können so verwendet werden. Bei den einzelnen Tabellen sind weitere Beispiele angegeben.

**Markierung** Einzelnen Befehlen kann als Angabe ihres Wirkungsbereiches eine Markierung vorangestellt werden. Die Markierungen werden nachfolgend mit *Markierung* bezeichnet. Der Befehl

**d**

löscht den markierten Bereich und speichert ihn im Kopierregister. Von dort kann er bei Bedarf mit Einfügebefehlen wieder an anderer Stelle eingefügt werden (zum Beispiel mit **p**).

Die Markierung eines Bereiches kann mit der Maus oder mit der Tastatur erfolgen. Bei Verwendung der Maus haben ein bis vier Klicks (kurz nacheinander) folgende Wirkung:

<b>1</b> Klick	Cursor an die gewählte Position setzen
<b>2</b> Klicks	Wort unter dem Cursor markieren
<b>3</b> Klicks	Zeile unter dem Cursor markieren
<b>4</b> Klicks	Blockmarkierung einschalten

Wenn die linke Maustaste nach dem jeweils letzten Klick niedergehalten wird, kann durch Ziehen ein weiterer Bereich markiert werden (zeichen-, wort-, zeilenweise oder rechteckig).

Der Markierungsmodus kann im Normalmodus auch mit Hilfe von Tastenkürzeln eingeschaltet werden:

<b>v</b>	zeichenweise Markierung	(visual)
<b>V</b>	zeilenweise Markierung	
<b>[Strg] v</b>	Blockmarkierung	

Wenn Markierungsmodus eingeschaltet ist, kann der Bereich mit den Richtungstasten oder mit Positionsangaben (zum Beispiel **G** oder **%**) erweitert werden.

Wenn der Cursor an einem Wortanfang steht und der Markierungsmodus mit **v** eingeschaltet

wurde, dann markiert die danach eingegebene Positionsangabe

**3w**

die nächsten drei Wörter. Wenn der Cursor auf der öffnenden Klammern { eines *C*-Blockes steht und der Markierungsmodus eingeschaltet wurde, dann markiert die danach eingegebene Positionsangabe

**%**

den gesamten Block bis zur schließenden Klammern. Ein markierter Block kann dann mit einem Grundbefehl bearbeitet werden ( *c*, *d*, *s*, *x*, *y* ).

### Einfügen


<i>N</i> <b>a</b>	Text hinter dem Cursor einfügen ( <i>N</i> -fach)
<i>N</i> <b>A</b>	Text am Zeilenende einfügen ( <i>N</i> -fach)
<i>N</i> <b>i</b>	Text vor dem Cursor einfügen ( <i>N</i> -fach) (Einfügetaste  ist ebenfalls möglich)
<i>N</i> <b>I</b>	Text vor dem ersten Nichtleerzeichen der Zeile einfügen ( <i>N</i> -fach)
<i>N</i> <b>o</b>	neue Zeile unter der Cursorposition öffnen und Text einfügen ( <i>N</i> -fach)
<i>N</i> <b>O</b>	neue Zeile über der Cursorposition öffnen und Text einfügen ( <i>N</i> -fach)

Tabelle 8: Einfügen im Normal- und Markierungsmodus (Auswahl)

### Löschen



<i>N</i> <b>x</b>	<i>N</i> Zeichen unter und nach dem Cursor löschen ( die Löschtaste  kann ebenfalls verwendet werden)
<i>N</i> <b>X</b>	<i>N</i> Zeichen vor dem Cursor löschen
<i>N</i> <b>d</b> <i>Bewegung</i>	Text bis zur der Position löschen, die durch <i>Bewegung</i> festgelegt ist
<i>Markierung</i> <b>d</b>	den markierten Text löschen
<i>N</i> <b>dd</b>	lösche <i>N</i> Zeilen
<i>N</i> <b>D</b>	lösche bis zum Zeilenende (und ggf. <i>N</i> -1 weitere Zeilen)
<i>N</i> <b>J</b>	<i>N</i> Zeilen zu einer Zeile zusammenfassen
<i>Markierung</i> <b>J</b>	die markierten Zeilen zu einer Zeile zusammenfassen
 <b>w</b>	Wort <i>vor</i> dem Cursor löschen (Einfügemodus)

Tabelle 9: Löschen im Normal- und Markierungsmodus (Auswahl)

Hier einige Verwendungsbeispiele zum Löschen:

<b>20x</b>	20 Zeichen ab der Cursorposition nach rechts löschen
<b>5X</b>	5 Zeichen vor der Cursorposition löschen
<b>3dd</b>	3 Zeilen löschen, beginnend mit der Zeile, in der der Cursor steht
<b>dG</b>	von der Cursorposition bis zum Dateiende löschen
<b>d2f)</b>	von der Cursorposition bis zur zweiten schließenden, runden Klammer in der aktuellen Zeile löschen

## Kopieren

$N$ <b>yy</b>	$N$ Zeilen in den Kopierpuffer übernehmen
$N$ <b>Y</b>	$N$ Zeilen in den Kopierpuffer übernehmen
$N$ <b>y</b> <i>Bewegung</i>	Text bis zur der Position, die durch <i>Bewegung</i> festgelegt ist, in den Kopierpuffer übernehmen
<i>Markierung</i> <b>Y</b>	markierten Text in den Kopierpuffer übernehmen
$N$ <b>P</b>	Text aus dem Kopierpuffer $N$ -mal vor der Cursorposition einfügen
$N$ <b>p</b>	Text aus dem Kopierpuffer $N$ -mal nach der Cursorposition einfügen

Tabelle 10: Kopieren im Normal- und Markierungsmodus (Auswahl)

Einige Verwendungsbeispiele zum Kopieren:

<b>5yy</b>	5 Zeilen in den Kopierpuffer übernehmen
<b>yG</b>	alles von der Cursorposition bis zum Dateiende in den Kopierpuffer
<b>y2f)</b>	von der Cursorposition bis zur zweiten schließenden, runden Klammer in den Kopierpuffer
<b>P</b>	Inhalt des Kopierpuffers vor der aktuellen Zeile einfügen
<b>3P</b>	Inhalt des Kopierpuffers 3 Mal vor der aktuellen Zeile einfügen
<b>2p</b>	Inhalt des Kopierpuffers 2 Mal nach der aktuellen Zeile einfügen

## Änderungen

$N$ <b>cc</b>	$N$ Zeilen ersetzen
$N$ <b>c</b> <i>Bewegung</i>	Text bis zur der Position, die durch <i>Bewegung</i> festgelegt ist, ersetzen
<i>Markierung</i> <b>C</b>	den markierten Text ersetzen
$N$ <b>C</b>	ersetze bis zum Zeilenende (und ggf. $N-1$ weitere Zeilen)
$N$ <b>S</b>	$N$ Zeilen ersetzen
$N$ <b>s</b>	$N$ Zeichen ersetzen
$N$ <b>~</b>	Groß- / Kleinschreibung für $N$ Zeichen wechseln
<i>Markierung</i> <b>~</b>	Groß- / Kleinschreibung für den markierten Bereich wechseln
<i>Markierung</i> <b>u</b>	markierten Bereich in Kleinbuchstaben umwandeln
<i>Markierung</i> <b>U</b>	markierten Bereich in Großbuchstaben umwandeln

Tabelle 11: Änderungen im Normal- und Markierungsmodus (Auswahl)

Einige Verwendungsbeispiele zum Ändern:

<b>4cc</b>	4 Zeilen löschen und in den Einfügemodus umschalten
<b>c%</b>	wenn der Cursorposition auf einer Klammer steht, alles bis zur Gegenklammer löschen und in den Einfügemodus umschalten
<b>5s</b>	5 Zeichen löschen und in den Einfügemodus umschalten
<b>8~</b>	Groß-/Kleinschreibung für 8 Zeichen ab der Cursorposition umschalten



## Änderungen rückgängig machen


$N$ <b>u</b>	die letzten $N$ Änderungen rückgängig machen (undo)
$N$  <b>R</b>	die letzten $N$ rückgängig gemachten Änderungen rückgängig machen (redo)
<b>U</b>	die letzte geänderte Zeile wieder herstellen

Tabelle 12: Änderungen

## 5 Suchen




$N$ <i>/Suchmuster</i> 	das $N$ -te Vorkommen von <i>Suchmuster</i> vorwärts suchen
$N$ <i>?Suchmuster</i> 	das $N$ -te Vorkommen von <i>Suchmuster</i> rückwärts suchen
$N$ <b>n</b>	die letzte Suche wiederholen (next)
$N$ <b>N</b>	die letzte Suche in Gegenrichtung wiederholen
$N$ <b>*</b>	das Wort unter dem Cursor vorwärts suchen
$N$ <b>#</b>	das Wort unter dem Cursor rückwärts suchen
$N$ <b>gd</b>	zur lokalen Vereinbarung der Größe unter dem Cursor (zum Beispiel einer lokalen Variablen in <i>C</i> )
$N$ <b>gD</b>	zur globalen Vereinbarung der Größe unter dem Cursor (zum Beispiel einer globalen Variablen in <i>C</i> )

Tabelle 13: Suchbefehle (Auswahl)

Die ersten beiden Befehle in Tabelle 13 ( */* und *?* ) öffnen die Kommandozeile am unteren Rand des Hauptfensters (siehe auch Abbildung 2). Nach der Eingabe des Suchmusters wird die Eingabe durch  abgeschlossen. Zum Suchen, sowie zum Suchen und Ersetzen stehen außerdem zwei Menüeinträge im Hauptmenü **Edit** (oder **Editieren** in der deutschen Version) zur Verfügung.

## 6 Wortergänzung



 <b>p</b>	die Zeichenkette links vom Cursor (Wortanfang) durch ein Wort ergänzen, welches vor dem Cursor vorkommt. (previous)
 <b>n</b>	die Zeichenkette links vom Cursor (Wortanfang) durch ein Wort ergänzen, welches nach dem Cursor vorkommt. (next)

Tabelle 14: Wortergänzungen

Im Einfügemodus können angefangene Wörter durch die beiden Befehle in Tabelle 14 ergänzt werden. Dadurch kann, zum Beispiel bei längeren Variablen- und Funktionsnamen, erhebliche Tipparbeit eingespart und gleichzeitig lästiges Kopieren vermieden werden.

Bei mehreren möglichen Ergänzungen werden Wahlmöglichkeiten angeboten. Wenn im aktuellen Puffer keine passende Ergänzungen vorkommt, dann werden die anderen Puffer durchsucht. Findet sich auch dort nichts, dann werden (falls vorhanden) abgespeicherte Wortlisten (dictionaries) durchsucht.

## 7 Unterschiedliche Befehle

Es gibt eine große Anzahl sehr nützlicher Befehle, die das Editieren beschleunigen und erleichtern. Eine umfassende Darstellung ist an dieser Stelle nicht möglich. Nachfolgend wird eine kleine Auswahl von Befehlen eingeführt, die der geübte Benutzer fast immer benötigt.

### Letzte Änderung wiederholen ( . )

Im Normalmodus wiederholt die Eingabe eines Punktes die zuletzt ausgeführte Änderung. Wurde zum Beispiel zuletzt ein Wort durch ein anderes ersetzt, dann reicht es aus, den Cursor auf das nächste zu ersetzende Wort zu stellen und die Ersetzung durch den Punkt-Befehl zu wiederholen.

Mehrfachersetzungen führt man wie folgt aus: Die zu ersetzenden Wörter (zum Beispiel ein Variablenname) werden durch den Befehl `*` markiert (siehe Abschnitt 5). Eines der markierten Wörter wird ersetzt (zum Beispiel mit `cw`). Mit Hilfe des Positionierungsbefehls `n` wird nun das nächste Vorkommen gesucht und mit dem Punkt-Befehl ersetzt.

### Zahlenwert erhöhen oder vermindern ( <Strg-a> <Strg-x> )

Häufig ist es erforderlich, einen Zahlenwert zu erhöhen oder zu vermindern. Das kann im Normalmodus durch die Befehle `<Strg-a>` und `<Strg-x>` für die Zahl unter dem Cursor geschehen. Wird ein Wiederholungsfaktor verwendet, dann wird dieser Faktor addiert beziehungsweise subtrahiert. Oktalwerte (führende Null) und Hexadezimalwerte (führendes `0x` oder `0X`) werden richtig behandelt.

### Positionsmarken verwenden ( m{a-zA-Z} '{a-zA-Z} <Strg-o> )

Textpositionen können mit den Marken `a-z` und `A-Z` bezeichnet werden. Dazu dient der Befehl `m`, gefolgt von einem Markennamen (normal mode). Der Befehl `ma` ordnet der Zeile, in der der Cursor steht, die Marke `a` zu. Mit dem Befehl `'` gefolgt vom Markennamen wird die entsprechende Marke angesprungen (normal mode). Die Marke `a` wird also mit dem Befehl `'a` angesprungen.

Die kleingeschriebenen Marken sind nur im aktuellen Puffer bekannt, großgeschriebene Marken sind global und können über Puffergrenzen hinweg angesprungen werden.

Für Rücksprünge zu den letzten Cursorpositionen steht der Befehl `<Strg-o>` zur Verfügung. Gelegentlich muss dieser Befehl mehrfach angewendet werden, um zu der letzten Position zurückzukehren.

### Zeile vervollständigen ( <Strg-x><Strg-l> )

Mit Hilfe des Befehls `<Strg-x><Strg-l>` kann eine begonnene Zeile im Eingabemodus so ergänzt werden, dass sie einer anderen, bereits vorhandenen Zeile, entspricht, die denselben Anfang besitzt. Das zweite `for` im nachfolgenden Beispiel wird zu einer vollständigen Zeile ergänzt, die der ersten Zeile entspricht:

```
for( i=0; i<n; i+=1 )
    b[i] = a[i];
```

```
for
```

Durch wiederholtes Eingeben von `<Strg-l>` nach einem vorausgegangenen `<Strg-x>` werden der Reihe nach alle Zeilen als Ergänzung angeboten, die den vorhandenen Anfang besitzen.

Dadurch können häufig eine Suche und ein Kopiervorgang eingespart werden, ohne dass die augenblicklich bearbeitete Stelle im Puffer verlassen werden muss.

**Dateinamen vervollständigen ( <Strg-x><Strg-f> )**

Mit Hilfe des Befehls <Strg-x><Strg-f> kann der Anfang eines Dateinamens im Eingabemodus ergänzt werden. Dazu wird nach Dateien gesucht, die diesen Anfang haben. Wenn mehrere Dateien mit diesem Anfang vorhanden sind, wird eine Auswahlliste angezeigt.

**Datei öffnen ( gf )**

Der Befehle **gf** (**g**oto **f**ile) öffnet die Datei, deren Name unter dem Cursor steht. Damit kann zum Beispiel eine Datei geöffnet werden, deren Name in einer **include**-Anweisung einer **C**-Quelle enthalten ist.

**C-Block ersetzen oder löschen ( caB ciB daB diB )**

Wenn der Cursor innerhalb eines **C**-Blockes ( { ... } ) steht, dann löscht der Befehl **caB** (**change a** block) den gesamten Block einschließlich der geschweiften Klammern und schaltet in den Einfügemodus um.

Der Befehl **ciB** (**change inner block**) löscht nur den Blockinhalt. Der Cursor steht anschließend im Einfügemodus zwischen den geschweiften Klammern.

Die Befehle **daB** und **diB** löschen den gesamten Block beziehungsweise den Blockinhalt, behalten aber den Normalmodus bei.

**C-Bedingung ersetzen oder löschen ( cab cib dab dib )**

Wenn der Cursor zwischen runden Klammern steht, dann löscht der Befehl **cab** (**change a block**) den gesamten Inhalt einschließlich der runden Klammern und schaltet in den Einfügemodus um.

Der Befehl **cib** (**change inner block**) löscht nur den Klammerinhalt. Der Cursor steht anschließend im Einfügemodus zwischen den verbliebene Klammern.

Die Befehle **dab** und **dib** löschen den gesamten Block beziehungsweise den Inhalt, behalten aber den Normalmodus bei.

**Befehlsfolge aufzeichnen und anwenden ( q{a-z} )**

Eine Folge von Editierbefehlen kann mit Hilfe des Befehls **q**, gefolgt von einem Registernamen (**a-z**), aufgezeichnet werden. Die Aufzeichnung wird durch ein weiteres **q** beendet.

Wenn der Cursor anschließend neu positioniert ist, kann die Befehlsfolge mit dem Befehl **@**, gefolgt von dem Registernamen, angewendet werden.

Die Angabe eines Wiederholungsfaktors vor der Verwendung veranlaßt eine Mehrfachausführung.

## 8 Unterstützung beim Erstellen von C/C++-Programmen

Zur Unterstützung der Erstellung von C/C++-Programmen kann das plug-in `c.vim`<sup>1</sup> verwendet werden. Nach der Installation ist in der Hauptmenüleiste der Menüeintrag `C/C++` vorhanden (Abbildung 3).

Die ersten sechs Untermenüs ( `Comments` ... `C++` ) dienen dem Einfügen von Anweisungen, Kon-

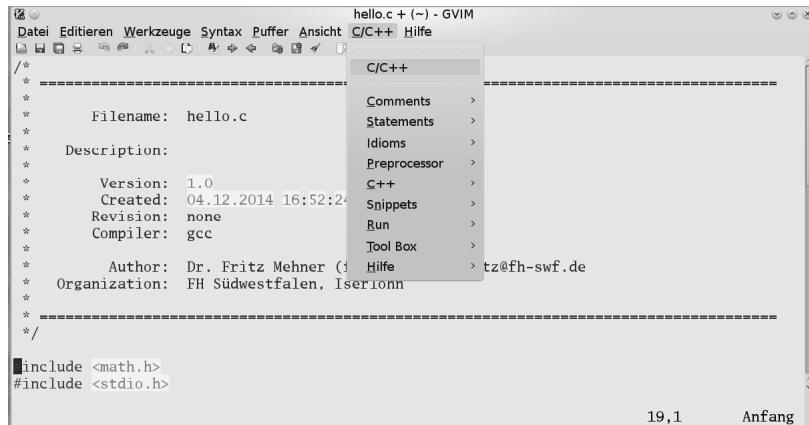


Abbildung 3: *gVim*-Hauptfenster mit dem Menü des C-plug-ins

strukturen und Kommentaren, die immer wieder vorkommen. Die meisten Einträge sind selbsterklärend und können einfach ausprobiert werden. Die Codeteile und Kommentare aus diesen Untermenüs werden beim Einfügen entsprechend ihrer Position im umgebenden Code automatisch eingerückt.

Das `Run`-Untermenü dient zur Unterstützung der Compilierung und Ausführung der gerade im Puffer befindlichen Programme, sowie zur Erzeugung von Ausdrucken (Abschnitt 9).

Das Untermenü `Comments` bietet umfangreiche Möglichkeiten zur Kommentierung der Quellen. Diese Kommentierungsmöglichkeiten sollen vor allem einen durchgängigen Kommentierungsstil ermöglichen. Dieser ist eine wichtige Voraussetzung für die gute Lesbarkeit und Wartbarkeit von Programmen. Er ermöglicht außerdem den Einsatz von Hilfsmitteln zur automatischen Erstellung von Dokumentationen (zum Beispiel mit dem Programm `doxygen`) und erleichtert die Suche in umfangreichen Quellcode-Beständen mit Suchprogrammen (zum Beispiel `grep`). Darüber hinaus erspart die Verwendung der vorgefertigten Kommentare natürlich eine Menge Schreibarbeit. Der Aufbau und die Zusammenstellung der Kommentare ist an professionellen Stilrichtlinien und Dokumentationsvorschriften ausgerichtet.

Das Untermenü `Snippets` bietet die Möglichkeit, eigene Code-Schnipsel unter einem Dateinamen in einem dafür vorgesehenen Verzeichnis abzulegen und bei Bedarf von dort wieder in den aktuellen Puffer einzulesen. Der Eintrag `write code snippet` kopiert einen markierten Zeilenbereich in eine Code-Schnipsel-Datei. Im Normalmodus (keine Markierung vorhanden) wird der gesamte Pufferinhalt kopiert.

Der Eintrag `pick up prototype` kopiert einen markierten Zeilenbereich, der einen C-Funktionskopf oder den Kopf einer Methodenimplementierung enthält, in einen Prototypen-Puffer und hängt ein Semikolon an. Kommentare werden entfernt. Wenn bereits Prototypen im Puffer waren, dann wird der neue Prototyp an diese angehängt. Der Eintrag `insert prototype(s)` fügt die im Prototypen-Puffer gesammelten Prototypen als neue Zeilen unterhalb des Cursors ein und löscht den Prototypen-Puffer.

<sup>1</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=213](http://vim.sourceforge.net/scripts/script.php?script_id=213), Fritz Mehner

## 9 Übersetzen und Ausführen von C/C++-Programmen

Die ersten drei Einträge des Untermenüs **Run** (Abbildung 4b) ermöglichen die Ausführung der Einzelschritte zur Erzeugung eines Programmes aus dem Inhalt des aktiven Editierpuffers:

**save and compile** Der Inhalt des aktiven Puffers wird gespeichert und kompiliert. Es entsteht eine Objekt-Datei.

**link** Aus der Objekt-Datei die zur C/C++-Datei im aktiven Puffer gehört, wird ein ausführbares Programm erzeugt. Wenn die Objekt-Datei nicht vorhanden oder älter als die Quelle ist, dann wird sie zunächst erzeugt.

**run** Die ausführbare Datei, die zur C/C++-Datei im aktiven Puffer gehört, wird ausgeführt. Wenn diese Datei nicht vorhanden ist, oder älter als die Objekt-Datei oder die Quelle ist, dann wird sie zunächst erzeugt.

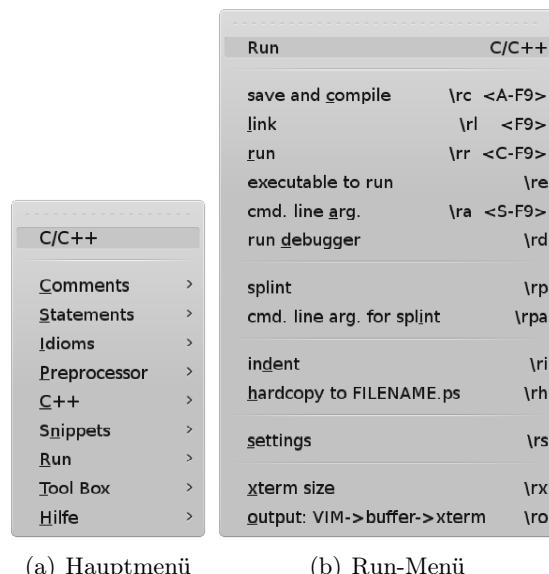


Abbildung 4: Das Hauptmenü und das Run-Menü des C-plug-ins

Es genügt also in den meisten Fällen, nach einer Änderung des aktuellen Pufferinhaltes sofort **run** auszuführen. Statt der Menüeinträge können auch die Tastenkürzel verwendet werden. Benötigt das auszuführende Programm Kommandozeilenargumente, dann können diese über den Menüeintrag **command line arguments** angegeben oder geändert werden. Bei einem Aufruf des Programmes über den Menüeintrag **run** oder über das Tastenkürzel **[Strg] [F9]** werden diese Argumente berücksichtigt.

Wurden bei der Übersetzung Fehler erkannt, dann werden die Fehlerbeschreibungen in den Editor zurückgelesen und in einem Fehlerfenster im unteren Teil des Hauptfensters dargestellt (Abbildung 5). Die einzelnen Meldungen können mit der Maus oder mit der Eingabetaste ausgewählt werden. Der Cursor springt dann an die Stelle, an der der Fehler erkannt wurde.

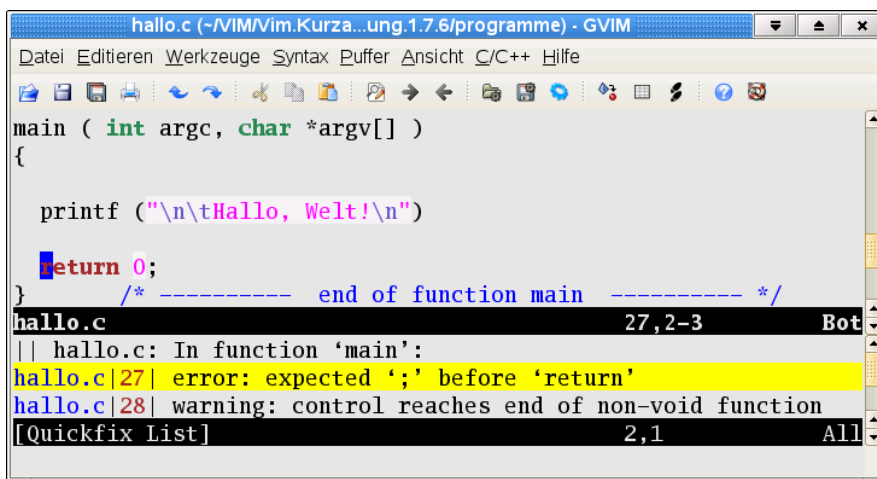












Abbildung 5: Fehlerfenster nach einer Übersetzung








## 10 Tastenkürzel

In einer benutzereigenen Einstellungsdatei und in Editor-Erweiterungen, wie zum Beispiel im *C*-plugin *c.vim*, können zur Bedienerleichterung einige Tastenkürzel festgelegt werden oder bereits festgelegt sein auch Abschnitt 15 und Liste 1, Abschnitt 15.1. Nachfolgend werden die Voreinstellungen verwendet. Die erste Gruppe ist unabhängig vom Pufferinhalt gültig, das heißt also auch unabhängig von der gerade verwendeten Programmiersprache:

	aktuellen Puffer abspeichern	
	erweiterten Datei-Browser aufrufen	
	preview-Fenster öffnen (siehe Abschnitt 12.2)	
	Fenster mit der Liste der Compilierungsfehler öffnen	
	Fenster mit der Liste der Compilierungsfehler schließen	[0.0ex]
	zum vorherigen Compilierungsfehler springen	
	zum nächsten Compilierungsfehler springen	
 	aktuellen Puffer abspeichern und zum vorhergehenden Puffer der Pufferliste springen	
 q	alle Puffer abspeichern und den Editor verlassen	








### 10.1 Kürzel für *C/C++*-Programme

Befindet sich im aktuellen Puffer eine *C*- oder *C++*-Datei, dann sind folgende Tastenkürzel definiert:

 	Die zum aktuellen Puffer gehörende ausführbare Datei starten, gegebenenfalls vorher übersetzen und binden <sup>2</sup> .
 	Aktuellen Puffer abspeichern und compilieren.
	Das zur Quelle im aktuellen Puffer gehörende Objekt binden.
 	Kommandozeilenargument für das Programm im aktuellen Puffer eingeben.










### 10.2 Kürzel für *bash*-Skripte

Befindet sich im aktuellen Puffer ein Shell-Skript, dann ist das folgende Tastenkürzel definiert, wenn das plugin *bash-support.vim* installiert ist (siehe Seite 22):

 	Das Skript im aktuellen Puffer abspeichern und ausführen <sup>2</sup> .
 	Kommandozeilenargument für das Skript im aktuellen Puffer eingeben.
 	Aktuellen Puffer abspeichern und die Syntax prüfen.
	<i>bash</i> -Debugger <i>bashdb</i> starten.

### 10.3 Kürzel für *Perl*-Programme

Befindet sich im aktuellen Puffer eine *Perl*-Datei, dann sind folgende Tastenkürzel definiert, wenn das plugin *perl-support.vim* installiert ist (siehe Seite 22):

 	Das Skript im aktuellen Puffer abspeichern und ausführen <sup>2</sup> .
 	Aktuellen Puffer abspeichern und die Syntax prüfen.
 	Kommandozeilenargument für das Skript im aktuellen Puffer eingeben.
	Debugger starten.
 	Für den Begriff unter dem Cursor die <i>Perl</i> -Dokumentation aufrufen (Programm <i>perldoc</i> ) und in einem neuen Fenster darstellen.

## 11 Kommentierung und Formatierung von Quellcode

Die Formatierung von Programmquellen verfolgt zwei Hauptziele: die Verbesserung der Lesbarkeit und die Verbesserung der Wartbarkeit. Der Sinn des ersten Zieles ist offensichtlich. Die Wartbarkeit wird in sinnvoll formatierten Quellen hauptsächlich dadurch erleichtert, dass beim wiederholten Editieren Blockbefehle angewendet werden können, mit denen beliebige rechteckige Ausschnitte eines Editierpuffers durch einzelne Editierbefehle verändert werden (siehe Abschnitt 4). Die Verwendung von Blockbefehlen erlaubt in vielen Fällen ein sehr schnelles Editieren.

### 11.1 Einrückung (=)

Der folgende Programmausschnitt ist offensichtlich unformatiert und äußerst unleserlich. Die Schachtelungstiefe der Vereinbarungen und Anweisungen ist nicht zu erkennen. Dieser Programmausschnitt wirkt stümperhaft!

```
int
main ( int argc, char *argv[] )
{
int i, n = 10, summe = 0;
for ( i=1; i<=n; i+=1 )
summe += i;
printf ("\n\tSumme 1 .. %d : %d\n", n, summe );
return 0;
}      /* ----- end of function main ----- */
```

Der Editor *Vim/gVim* ermöglicht mit Hilfe des Befehls = die Einrückung von Quelltexten. Im einfachsten Fall werden dazu die betreffenden Zeilen mit der Maus oder mit der Tastatur (Seite 6) markiert und anschließend der Befehl = eingegeben. Der Text sieht nun wie folgt aus:

```
int
main ( int argc, char *argv[] )
{
    int i, n = 10, summe = 0;
    for ( i=1; i<=n; i+=1 )
        summe += i;
    printf ("\n\tSumme 1 .. %d : %d\n", n, summe );
    return 0;
}      /* ----- end of function main ----- */
```

Man erkennt, dass innerhalb des Hauptprogrammes alle Anweisungen um zwei Zeichen eingerückt sind. Der Inhalt der **for**-Schleife ist um weitere zwei Zeichen eingerückt. Entsprechend würde bei tieferen Schachtelungen verfahren.

Wenn längere Programmabschnitte oder die gesamte Datei formatiert werden sollen, dann ist die Markierung mit der Maus schlecht oder gar nicht durchführbar. Im Normalmodus kann hier zur Formatierung der Befehl = mit nachfolgender Bewegung (siehe auch Seite 4 ff.) eingegeben werden. Hierzu einige Beispiele:

- = die aktuelle Zeile formatieren
- =iB den Block formatieren, in dem der Cursor steht (**i**inner **B**lock)
- =aB den Block einschließlich der Blockklammern formatieren, in dem der Cursor steht (**a** **B**lock)
- =% wenn der Cursor auf einer geschweiften Klammer steht: alles bis zur Gegenklammer formatieren (in *C/C++* einen vollständigen Block)
- =gg =G von der Cursorposition bis zum Dateianfang / Dateiende formatieren

## 11.2 Kommentare ( **gq** )

Kommentare erhalten die Einrückung, die der Schachtelungstiefe der kommentierten Programmeinheit entspricht.

Für Kommentartexte, die nur aus Fließtext bestehen und keine weitere Struktur aufweisen müssen, besteht die Möglichkeit, den Text zu umbrechen. Dabei entsteht linksbündig ausgerichteter Text, der höchstens bis zur Spalte 79 reicht. Der Kommentar im folgenden Beispiel ist zwar eingerückt, aber unformatiert:

```

int
main ( int argc, char *argv[] )
{
  /* Für Kommentartexte, die nur aus Fließtext
   * bestehen und keine weitere Struktur aufweisen
   * müssen, besteht die Möglichkeit, den Text zu umbrechen. Dabei entsteht
   * linksbündig ausgerichteter Text, der höchstens bis zur Spalte 79 reicht.
   */
  return 0;
}      /* ----- end of function main ----- */

```

Der Text wird nun markiert (Maus oder Tastatur) und anschließend der Befehl **gq** eingegeben. Damit erhält der Kommentar das folgende Aussehen:

```

int
main ( int argc, char *argv[] )
{
  /* Für Kommentartexte, die nur aus Fließtext bestehen und keine weitere
   * Struktur aufweisen müssen, besteht die Möglichkeit, den Text zu umbrechen.
   * Dabei entsteht linksbündig ausgerichteter Text, der höchstens bis zur
   * Spalte 79 reicht.
   */
  return 0;
}      /* ----- end of function main ----- */

```

Der Befehl **gq** ist in gleicher Weise auf **C++**-Kommentare anwendbar.

## 11.3 Ausrichtung

Das **Vim/gVim**-plug-in **Align**<sup>2</sup> ermöglicht die Ausrichtung verschiedener Bestandteile eines **C**-Programmes. Dadurch kann eine Menge lästiger Formatierungsarbeit eingespart werden. Drei Möglichkeiten sollen hier vorgestellt werden.

### Präprozessoranweisungen ( **\adef** )

Die folgenden Präprozessoranweisungen sollen ausgerichtet werden:

```

#define      SEKUNDE      1                /* Zeitbasis */
#define      MINUTE      (60*SEKUNDE)
#define      STUNDE      (60*MINUTE)
#define      TAG          (24*STUNDE)
#define PROCEDURE void          /* neues Schlüsselwort */

```

<sup>2</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=294](http://vim.sourceforge.net/scripts/script.php?script_id=294), Charles Campbell



Wenn das plug-in installiert ist, dann genügt die Markierung der Zeilen und die anschließende Eingabe der Tastenfolge **\adef** um das folgende Ergebnis zu erhalten:

```
#define SEKUNDE 1          /* Zeitbasis          */
#define MINUTE  (60*SEKUNDE)
#define STUNDE  (60*MINUTE)
#define TAG     (24*STUNDE)
#define PROCEDURE void      /* neues Schlüsselwort */
```

Die einzelnen Bestandteile der **define**-Anweisungen beginnen nun in der selben Spalte. Die Kommentare beginnen ebenfalls in der selben Spalte und werden auf eine gemeinsame Länge gesetzt.

### Variablenvereinbarungen ( \adec )

Die folgenden Vereinbarungen können nach ihrer Markierung mit dem Befehl **\adec** ausgerichtet werden.

```
struct abc_str abc;
int a; /* a */
char x[5]; /* x[5] */
struct abc_str *pabc; /* pabc */
static double *c=NULL; /* c */
static struct abc_str abcde; /* abcde */
```

Als Ergebnis erhält man das folgende Aussehen:

```
struct abc_str      abc;
int                 a;          /* a */
char                x[5];       /* x[5] */
struct abc_str      *pabc;      /* pabc */
static double       *c = NULL;  /* c */
static struct abc_str abcde;    /* abcde */
```

Die einzelnen Bestandteile der Vereinbarungen sind nun übersichtlich in mehreren Spalten angeordnet.

### Zuweisungen ( \t= )

Aufeinanderfolgende Zuweisungen sollten wegen der besseren Lesbarkeit unbedingt an den Gleichheitszeichen ausgerichtet sein. Die zugehörigen Zeilenendkommentare sollten in der selben Spalte beginnen. Wenn die Kommentare ähnliche Längen haben, ist die Ausrichtung der Kommentaren ebenfalls wünschenswert. Die folgenden Zeilen stellen ein auffallend schlechtes Beispiel für die Formatierung aufeinanderfolgender Zuweisungen dar:



```
x0 = 0.0; /*Intervallanfang*/
x1 = 6.4; /*Intervallende*/
n = 100; /*Teilintervalle */
xinkr=(x1-x0)/n; /* Schrittweite */
```

Nach ihrer Markierung können diese jedoch mit dem Befehl **\t=** wie folgt ausgerichtet werden:

```
x0 = 0.0; /* Intervallanfang */
x1 = 6.4; /* Intervallende */
n = 100; /* Teilintervalle */
xinkr = (x1-x0)/n; /* Schrittweite */
```



## 12.2 Navigation mit Hilfe von tags und dem preview-Fenster

Mit Hilfe des Icons <sup>5</sup> kann für alle Dateien im und unterhalb des Arbeitsverzeichnisses eine tag-Liste erstellt werden. Wenn diese Liste vorhanden ist, dann kann mit Hilfe des Icons  zu der Definition des Programmelementes unter dem Cursor gesprungen werden. Befindet sich das Sprungziel nicht im aktuellen Puffer, dann wird die entsprechende Datei geöffnet oder in einen bereits offenen Puffer gesprungen.

Beim direkten Sprung zu einer anderen Dateiposition (zum Beispiel der Definition einer Funkti-

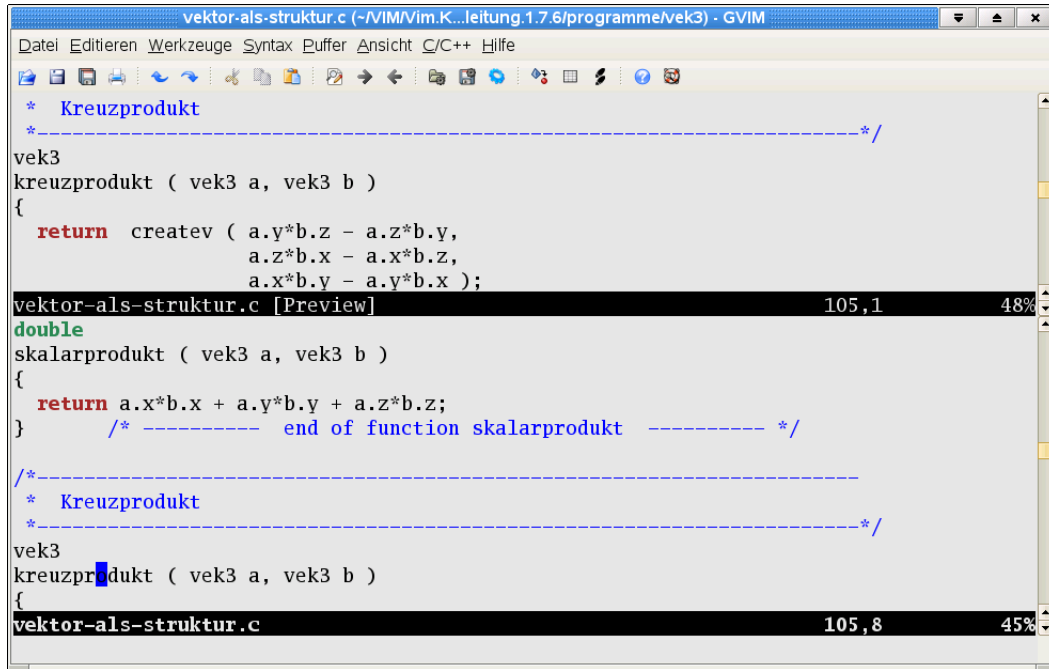


Abbildung 7: Verwendung des preview-Fensters

on) ist der Startpunkt (zum Beispiel der Aufruf der Funktion) natürlich nicht mehr sichtbar. Diese Unschönheit kann durch die Verwendung eines sogenannte preview-Fensters vermieden werden. In Abbildung 7 steht der Cursor im unteren Fenster auf dem Aufruf der Funktion `kreuzprodukt`. Durch das Tastenkürzel `[F4]` wird in der tag-Liste die Position der Funktionsdefinition gesucht und die entsprechende Datei im oberen Fenster, dem preview-Fenster, dargestellt. Das preview-Fenster bleibt nun offen und kann in der selben Weise zur Anzeige weiterer Programmelemente weiterverwendet werden. Der Cursor bleibt dabei stets im aktiven Puffer.

## 12.3 Navigation in vielen Puffern

Benutzer, die häufig mit einer größeren Anzahl Dateien arbeiten, müssen ständig den aktuellen Puffer wechseln. Der Wechsel kann über das Puffer-Menü (*gVim*) oder über die Kommandozeile (*Vim*: Befehle `:ls` und `:edit`) erfolgen. Der folgende Eintrag in der Datei `.vimrc` verbindet die beiden Kommandozeilenbefehle und belegt die Funktionstaste `[F12]` damit:

```
noremap      <F12>      :ls<CR>:edit #
inoremap     <F12>     <C-C>:ls<CR>:edit #
```

<sup>5</sup>Die Icons können, abhängig von den bei der Übersetzung des *gVim* verwendeten Graphikbibliotheken, ein unterschiedliches Aussehen haben.

Die Funktionstaste zeigt nun die Pufferliste im Fußbereich des Hauptfensters. Der Befehl `:edit #` erwartet eine der Puffernummern aus dieser Liste. Nach der Eingabe wird der ausgewählte Puffer zum aktuellen Puffer.

Weitergehende Ansprüche erfüllt das plug-in `bufexplorer.zip`<sup>6</sup>. Es stellt die Pufferliste in einem eigenen Fenster dar. Die Einträge können unter anderem nach unterschiedlichen Gesichtspunkten sortiert werden. Die Suche nach Namensbestandteilen ist mit den üblichen Suchbefehlen möglich (Abschnitt 5). Abbildung 8 zeigt den `buffer explorer` in einem Unterfenster.

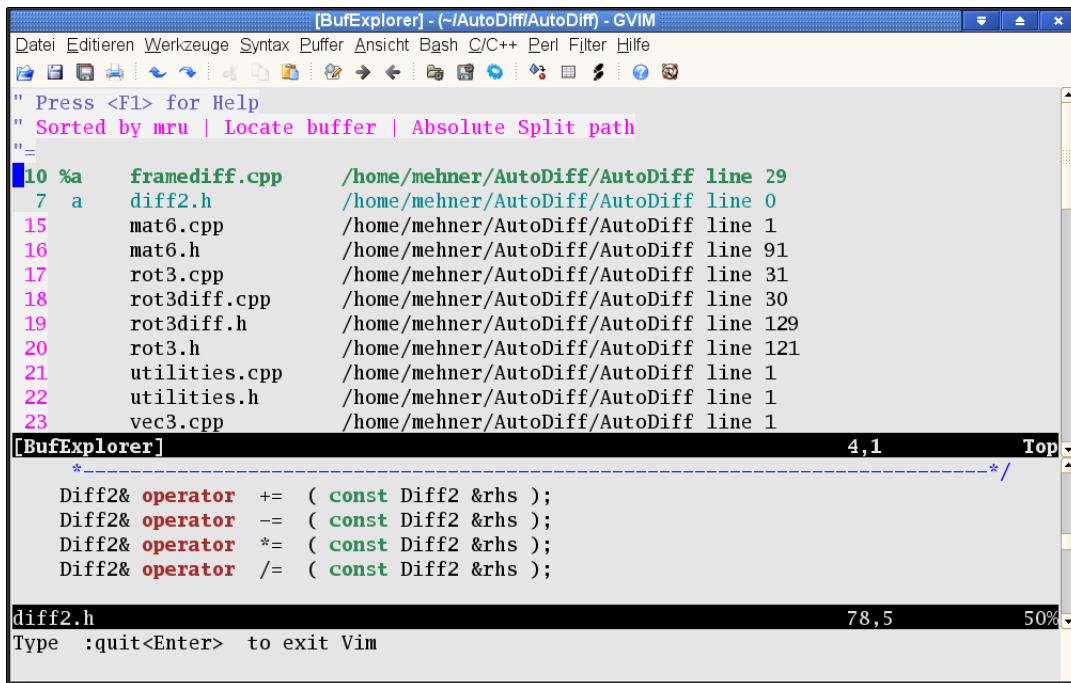




Abbildung 8: Verwendung des `buffer explorer` zur Auswahl des nächsten Editierpuffers

Alternativ zu der oben gezeigten Tastenbelegung, kann auch die Verwendung des `buffer explorer` auf eine Funktionstaste gelegt werden:

```
noremap <silent> <F12>          :BufExplorer<CR>
inoremap <silent> <F12>    <C-C>:BufExplorer<CR>
```

Damit ist eine sehr schnelle und bequeme Navigation bei vielen Puffern möglich.

## 12.4 Liste der zuletzt verwendeten Dateien

Das plug-in `mru.zip`<sup>7</sup> unterhält eine Liste der zuletzt verwendeten Dateien. Die Liste erscheint als Untermenü im Datei-Menü oder in einem eigenen Puffer. In Liste 2 wird zum Öffnen der Liste die Tastenkombination   festgelegt. Damit können insbesondere die Dateien aus den letzten Editorsitzungen sehr schnell wieder geöffnet werden.

<sup>6</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=42](http://vim.sourceforge.net/scripts/script.php?script_id=42)

<sup>7</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=521](http://vim.sourceforge.net/scripts/script.php?script_id=521)

## 13 Verwendung von Shell-Befehlen

*Linux-/Unix*-Systemprogramme, Skripte und Filterprogramme, die von der Kommandozeile aufrufbar sind, können auch aus dem Editor heraus zur Bearbeitung oder Ergänzung des Inhaltes verwendet werden. Pipes und Ein-/Ausgabeumlenkungen sind möglich.

Die folgenden Zeilen wurden zur Verwendung als Prototypen zusammengestellt und sollen der besseren Lesbarkeit wegen tabellenartig formatiert werden:

```
void tausche_double ( double *a, double *b );
void sortiere_feld_aufsteigend ( double a[], int n );
void sortiere_feld_absteigend ( double a[], int n );
void fuehle_feld ( double a[], int n, int unten, int oben );
void gib_feld_aus ( double a[], int n );
```

Diese Zeilen werden zunächst markiert, dann wird die Befehlszeile aufgerufen. Dort erscheint bereits der markierte Zeilenbereich (: '<', '>'). Dieser kann wie folgt ergänzt werden:

```
:'<,>' ! column -t | sort
```

Das Ausrufezeichen leitet die nachfolgenden Shell-Befehle ein, an die der markierte Bereich zur Filtrung weitergeleitet wird. Der Befehl `column -t` richtet die Zeilen tabellenartig aus (siehe `man 1 column`). Das Ergebnis wird durch eine pipe `|` an den Filter `sort` weitergegeben (siehe `man 1 sort`). Diese Befehlsfolge ersetzt den markierten Bereich durch das formatierte Ergebnis:

```
void fuehle_feld          ( double a[], int n, int unten, int oben );
void gib_feld_aus        ( double a[], int n );
void sortiere_feld_absteigend ( double a[], int n );
void sortiere_feld_aufsteigend ( double a[], int n );
void tausche_double      ( double *a, double *b );
```

In ähnlicher Weise lassen sich Programm- und Befehlsausgaben in den Editor einlesen. Die nachstehende Zeile liest mit Hilfe des Editorkommandos `read` die Ausgabe des nachfolgenden Shell-Befehls in den aktuellen Puffer ein.

```
:read ! ls bilder/*.png
```

Der Befehl `ls bilder/*.png` listet alle `png`-Dateien im Unterverzeichnis `bilder` auf:

```
bilder/ctags.png
bilder/format-1.png
bilder/format1.png
bilder/preview.png
bilder/taglist.png
...
```

Das plug-in `textfilter.vim`<sup>8</sup> stellt verschiedene Filteroperationen zur Textbearbeitung zur Verfügung (Sortierungen, Zeilennummerierung, Ausrichtung von rechteckigen Bereichen, Umwandlung Tabulatoren  $\Leftrightarrow$  Leerzeichen und andres mehr).

Gelegentlich ist es erforderlich, eine größere Anzahl von Dateien in den Editor zu laden. Um nicht jede Datei einzeln öffnen zu müssen, gibt man die Dateien mit Hilfe geeigneter Befehle auf der Aufrufzeile an. Im folgenden Aufruf ermittelt `find` alle `html`-Dateien in und unterhalb des Arbeitsverzeichnisses. Der Befehl `grep -l` gibt von den gefundenen Dateien nur noch die Namen derjenigen aus, die die Zeichenkette `mfh-iserlohn` enthalten. Diese Dateinamen werden als Kommandozeilenparameter beim Aufruf von `gVim` verwendet:

```
gvim $( grep -l "mfh-iserlohn" $( find . -name "*.html" ) )
```

<sup>8</sup><http://lug.fh-swf.de/vim>, Fritz Mehner

Mit Hilfe des Kommandoszusatzes **bufdo** kann nun ein Editor Kommando der Reihe nach auf alle Puffer angewendet werden, zum Beispiel eine globale Ersetzung mit Bestätigung:

```
:bufdo %s/mfh-iserlohn/fh-swf/gc
```

## 14 Weitere plug-ins

Durch Hinzufügen von plug-ins kann der Bedienkomfort des Editors weiter gesteigert werden. Weiterhin können Funktionen ergänzt werden, die von Integrierten Entwicklungsumgebungen her bekannt sind. Es folgen einige Vorschläge, die besonders für Programmierer interessant sind. Diese plug-ins haben jeweils eine eigene Dokumentation und zum Teil umfangreiche Einstellmöglichkeiten zur Anpassung an die Benutzerbedürfnisse.

### 14.1 Allgemein verwendbare plug-ins

**Align** Textausrichtung (siehe Abschnitt 11.3).

**a.vim**<sup>9</sup> Schneller Wechsel zwischen den Puffern zusammengehöriger **c-/cpp-** und **h-/hpp-**Dateien.

**bufexplorer.vim**<sup>10</sup> Schneller Wechsel zwischen beliebigen Puffern (siehe Abschnitt 12.3).

**Git Support**<sup>11</sup> Zugriff auf Git-Archive.

**mru.vim**<sup>12</sup> Unterhält eine Liste der zuletzt verwendeten Dateien. Die Liste erscheint als Untermenü im Datei-Menü oder in einem eigenen Puffer (siehe Abschnitt 12.4).

**matchit.vim**<sup>13</sup> Erweiterung des **Vim**-Operators **%** (Seite 6), der das Gegenstück zu einer Klammer eines Klammerpaares sucht. Diese Erweiterung springt zusammengehörige Schlüsselwortklammern und Markierungen an, zum Beispiel **if – then – else (C/C++)** oder **<TABLE> – </TABLE>** (HTML). Zu den unterstützten Sprachen gehören unter anderen **L<sup>A</sup>T<sub>E</sub>X**, Pascal, XML und verschiedene Shells.

**project**<sup>14</sup> Leistungsfähige Projektverwaltung.

**taglist.vim** Quellcode-Browser (siehe Abschnitt 12.1).

### 14.2 Sprachspezifische plug-ins

**bash-support.vim**<sup>15</sup> Dieses plug-in stellt eine Bash-IDE zur Verfügung. Das Einfügen von Kommentaren, Anweisungen, Code-Schnipseln, Tests, Bash-Optionen und regulären Ausdrücken wird unterstützt. Die meisten Funktionen stehen per Tastenkürzel auch im nicht-graphischen Modus zur Verfügung.

**perl-support.vim**<sup>16</sup> Dieses plug-in stellt eine Perl-IDE zur Verfügung. Das Einfügen von Kommentaren, Anweisungen, Code-Schnipseln, Test und Spezialvariablen wird unterstützt. Das bequeme und schnelle Erzeugen, Analysieren und Testen von regulären Ausdrücken ist ebenso möglich wie das Erstellen und Testen von POD-Dokumentationen. Die Verwendung von **perltidy**,

<sup>9</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=31](http://vim.sourceforge.net/scripts/script.php?script_id=31)

<sup>10</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=42](http://vim.sourceforge.net/scripts/script.php?script_id=42)

<sup>11</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=4497](http://vim.sourceforge.net/scripts/script.php?script_id=4497)

<sup>12</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=521](http://vim.sourceforge.net/scripts/script.php?script_id=521)

<sup>13</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=39](http://vim.sourceforge.net/scripts/script.php?script_id=39)

<sup>14</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=69](http://vim.sourceforge.net/scripts/script.php?script_id=69)

<sup>15</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=365](http://vim.sourceforge.net/scripts/script.php?script_id=365)

<sup>16</sup>[http://vim.sourceforge.net/scripts/script.php?script\\_id=556](http://vim.sourceforge.net/scripts/script.php?script_id=556)

`perlritic` und drei Profilern ist integriert. Die meisten Funktionen stehen per Tastenkürzel auch im nicht-graphischen Modus zur Verfügung.

Weiter sprachspezifische plug-ins finden sich in der Skript-Sammlung auf der Vim-Seite [Vim].

## 15 Benutzereinstellungen

Die systemweit gültigen Einstellungsdateien des *Vim*-Editors sind bei einer Standard-Installation unter *Linux* im Verzeichnisbaum mit der Wurzel `/usr/share/vim` enthalten und müssen normalerweise nicht bearbeitet werden.

Für einen Benutzer empfiehlt es sich, eigene Einstellungsdateien `.vimrc` und `.gvimrc` in seinem home-Verzeichnis anzulegen. Die Listen 1 und 2 zeigen Beispiele für solche Dateien. In diesen Dateien können beliebige Einstellungen eingetragen werden, um den Editor den persönlichen Bedürfnissen anzupassen. Beide Dateien werden von *Vim* und *gVim* bei ihrem Aufruf gelesen, wenn sie vorhanden sind. Die Einstellungen in `.vimrc` werden für beide Varianten wirksam, diejenigen in `.gvimrc` nur für den *gVim*.

Der Benutzer hat weiterhin die Möglichkeit, Erweiterungsskripte (sogenannte plug-ins), Wörterbücher, Vorlagedateien, Code-Schnipsel, zusätzliche Syntaxeinfärbungen und so weiter zu verwenden. Für einige dieser Erweiterungen müssen Standardverzeichnisse verwendet werden. Abbildung 9 zeigt einen Verzeichnisbaum unterhalb des versteckten Verzeichnisses `/home/Benutzer/.vim`, wobei *Benutzer* hier den tatsächlichen Verzeichnisnamen unterhalb `/home` darstellt. Der Name und der Verwendungszweck der folgenden Verzeichnisse ist fest vorgegeben:

**`/.vim/doc`** In diesem Verzeichnis stehen die Hilfedateien, die zu einzelnen plug-ins gehören.

**`/.vim/ftplugin`** In diesem Verzeichnis stehen Dateien, in denen Einstellungen und Maßnahmen beschrieben sind, die nur für bestimmte Dateitypen gültig sind (zum Beispiel nur für *C/C++*-Dateien). Einzelne plug-ins bringen derartige Dateien mit. Der Benutzer kann auch selbst welche erstellen.

**`/.vim/plugin`** In diesem Verzeichnis werden die eigentlichen plug-ins abgelegt. Die plug-ins haben stets die Erweiterung `.vim`. Alle plug-ins in diesem Verzeichnis werden vom Editor beim Start geladen.

**`/.vim/autoload`** Diesem Verzeichnis wird unter anderem vom *Align*-plug-in verwendet.

Im Verzeichnisbaum mit der Wurzel `.vim` können weitere Verzeichnisse und Dateien vorhanden sein. Sofern diese von fertigen plug-ins (zum Beispiel von <http://vim.sourceforge.net>) verwendet werden, sind genaue Angaben der begleitenden Dokumentation zu entnehmen.

### 15.1 Beispiele für eigene Einstellungsdateien

Eine weitere Möglichkeit besteht darin, eigene Dateien `.gvimrc` und `.vimrc` anzulegen, die eigene Benutzereinstellungen enthalten. Die Listen 1 und 2 zeigen Beispiele für nützliche Einstellungen.

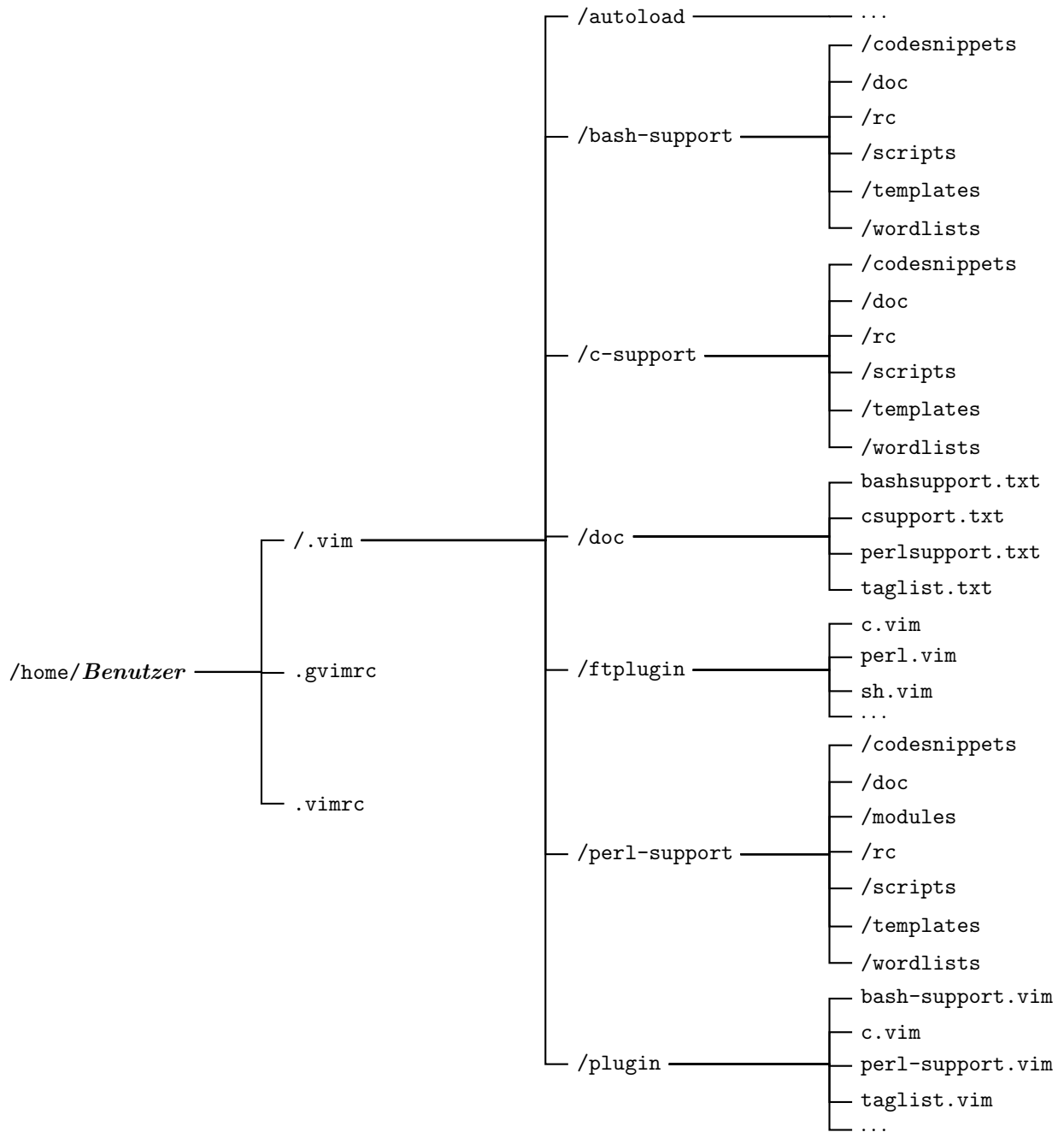


Abbildung 9: Persönliche *Vim*-Verzeichnisse und Einstellungsdateien (Auswahl)



Liste 1: Beispiel für die Einstellungsdatei `.vimrc`

```

1  "=====
2  " GENERAL SETTINGS
3  "=====
4
5  "-----
6  " Use Vim settings, rather than Vi settings.
7  " This must be first, because it changes other options as a side effect.
8  "-----
9  set nocompatible
10 "
11 "-----
12 " Enable file type detection. Use the default filetype settings.
13 " Also load indent files, to automatically do language-dependent indenting.
14 "-----
15 filetype plugin on
16 filetype indent on
17 "
18 "-----
19 " Switch syntax highlighting on.
20 "-----
21 syntax on
22 "
23 "-----
24 " Various settings
25 "-----
26 set autoindent           " copy indent from current line
27 set autoread             " read open files again when changed outside Vim
28 set autowrite            " write a modified buffer on each :next , ...
29 set backspace=indent,eol,start " backspacing over everything in insert mode
30 set backup               " keep a backup file
31 set browsedir=current   " which directory to use for the file browser
32 set columns=104         " number of columns of the screen
33 set complete+=k         " scan the files given with the 'dictionary' option
34 set history=50          " keep 50 lines of command line history
35 set hlsearch            " highlight the last used search pattern
36 set incsearch           " do incremental searching
37 set lines=48            " number of lines of the Vim window
38 set listchars=tab:>.,eol:\$ " strings to use in 'list' mode
39 set mouse=a             " enable the use of the mouse
40 set nowrap              " do not wrap lines
41 set popt=left:8pc,right:3pc " print options
42 set ruler               " show the cursor position all the time
43 set shiftwidth=2        " number of spaces to use for each step of indent
44 set showcmd            " display incomplete commands
45 set smartindent         " smart autoindenting when starting a new line
46 set tabstop=2           " number of spaces that a <Tab> counts for
47 set visualbell         " visual bell instead of beeping
48 set wildignore=*.bak,*.o,*.e,*~ " wildmenu: ignore these extensions
49 set wildmenu            " command-line completion in an enhanced mode
50 set scrolloff=1        " number of lines to keep above and below the cursor
51 "
52 set foldmethod=syntax   " the kind of folding
53 set nofoldenable        " keep all folds open
54 "
55 "=====

```

```

56 " BUFFERS, WINDOWS
57 "=====
58 "
59 "-----
60 " The current directory is the directory of the file in the current window.
61 "-----
62 if has("autocmd")
63     autocmd BufEnter * :lchdir %:~:~
64 endif
65 "
66 "-----
67 " close window (conflicts with the KDE setting for calling the process manager)
68 "-----
69 noremap <C-Esc>      :close<CR>
70 inoremap <C-Esc> <C-C>:close<CR>
71 "
72 "-----
73 " Fast switching between buffers
74 " The current buffer will be saved before switching to the next one.
75 " Choose :bprevious or :bnext
76 "-----
77 noremap <silent> <s-tab>      :if &modifiable && !&readonly &&
78     \                          &modified <CR> :write<CR> :endif<CR>:bprevious<CR>
79 inoremap <silent> <s-tab> <C-C>:if &modifiable && !&readonly &&
80     \                          &modified <CR> :write<CR> :endif<CR>:bprevious<CR>
81 "
82 "-----
83 " Leave the editor with Ctrl-q (KDE): Write all changed buffers and exit Vim
84 "-----
85 nnoremap <C-q>      :wqall<CR>
86 "
87 "-----
88 " When editing a file, always jump to the last known cursor position.
89 " Don't do it when the position is invalid or when inside an event handler
90 " (happens when dropping a file on gvim).
91 "-----
92 if has("autocmd")
93     autocmd BufReadPost *
94         \ if line("'\"") > 0 && line("'\"") <= line("$") |
95         \   exe "normal! g'\"" |
96         \ endif
97 endif " has("autocmd")
98 "
99 "-----
100 " additional hot keys
101 "-----
102 "     F2 - write file without confirmation
103 "     F3 - call file explorer Ex
104 "     F4 - show tag under cursor in the preview window (tagfile must exist!)
105 "     F5 - show the current list of errors
106 "     F6 - close the quickfix window (error list)
107 "     F7 - display previous error
108 "     F8 - display next error
109 "-----
110 noremap <silent> <F2>      :write<CR>
111 noremap <silent> <F3>      :Explore<CR>

```

```

112 noremap <silent> <F4>      :execute ":ptag ".expand("<cword>")<CR>
113 noremap <silent> <F5>      :copen<CR>
114 noremap <silent> <F6>      :cclose<CR>
115 "
116 inoremap <silent> <F2>      <C-C>:write<CR>
117 inoremap <silent> <F3>      <C-C>:Explore<CR>
118 inoremap <silent> <F4>      <C-C>:execute ":ptag ".expand("<cword>")<CR>
119 inoremap <silent> <F5>      <C-C>:copen<CR>
120 inoremap <silent> <F6>      <C-C>:cclose<CR>
121 "
122 noremap <silent> <F7>      :cprevious<CR>
123 inoremap <silent> <F7>      <C-C>:cprevious<CR>
124 noremap <silent> <F8>      :cnext<CR>
125 inoremap <silent> <F8>      <C-C>:cnext<CR>
126 "
127 "-----
128 " use Q for formatting a paragraph
129 "-----
130 map Q gq
131 "
132 "-----
133 " comma always followed by a space
134 "-----
135 inoremap , ,<Space>
136 "
137 "-----
138 " autocomplete parenthesis, brackets and braces (visual and insert mode)
139 "-----
140 inoremap ( ()<Left>
141 inoremap [ []<Left>
142 inoremap { {}<Left>
143 "
144 vnoremap ( s()<Esc>P<Right>%
145 vnoremap [ s[]<Esc>P<Right>%
146 vnoremap { s{}<Esc>P<Right>%
147 "
148 "-----
149 " autocomplete quotes (visual and insert mode)
150 "-----
151 inoremap ' ''<Left>
152 inoremap " ""<Left>
153 inoremap ' ``<Left>
154 "
155 vnoremap ' s''<Esc>P<Right>%
156 vnoremap " s""<Esc>P<Right>%
157 vnoremap ' s``<Esc>P<Right>%
158 "
159 "=====
160 " VARIOUS PLUGIN CONFIGURATIONS
161 "=====
162 "
163 "-----
164 " plugin bufferexplorer.vim
165 " F12 - toggle buffer explorer window
166 "-----
167 noremap <silent> <F12>      :BufExplorer<CR>

```

```

168 inoremap <silent> <F12> <C-C>:BufExplorer<CR>
169 "
170 let Tlist_GainFocus_On_ToggleOpen = 1
171 let Tlist_Close_On_Select = 1
172 "
173 "-----
174 " plugin taglist.vim : toggle the taglist window
175 "-----
176 noremap <silent> <F11> :TlistToggle<CR>
177 inoremap <silent> <F11> <C-C>:TlistToggle<CR>

```

## Liste 2: Beispiel für die Einstellungsdatei .gvimrc

```

1 set cmdheight=2 " Make command line two lines high
2 set mousehide " Hide the mouse when typing text
3
4 highlight Normal guibg=grey90
5 highlight Cursor guibg=Blue guifg=NONE
6 highlight lCursor guibg=Cyan guifg=NONE
7 highlight NonText guibg=grey80
8 highlight Constant gui=NONE guibg=grey95
9 highlight Special gui=NONE guibg=grey95
10
11 let c_comment_strings=1 " highlight strings inside C comments
12 "
13 "-----
14 " Moving cursor to other windows:
15 " shift down : change window focus to lower one (cyclic)
16 " shift up : change window focus to upper one (cyclic)
17 " shift left : change window focus to one on left
18 " shift right : change window focus to one on right
19 "-----
20 nnoremap <s-down> <c-w>w
21 nnoremap <s-up> <c-w>W
22 nnoremap <s-left> <c-w>h
23 nnoremap <s-right> <c-w>l
24 "
25 "-----
26 " plugin mru.vim (Yegappan Lakshmanan)
27 " Shift-F3 - open list of recently used files
28 "-----
29 noremap <silent> <s-F3> :MRU<CR>
30 inoremap <silent> <s-F3> <Esc>:MRU<CR>
31 "
32 "-----
33 " toggle insert mode <--> 'normal mode with the <RightMouse>-key
34 "-----
35 nnoremap <RightMouse> <Insert>
36 inoremap <RightMouse> <ESC>
37 "
38 "-----
39 " use font with clearly distinguishable brackets : ()[]{}
40 "-----
41 set guifont=Luxi\ Mono\ 14



```

## 16 Zusätzliche Benutzungshinweise

**Editoraufrufe** Rufen Sie den Editor möglichst nur einmal auf, um mögliche Konflikte mit mehrfach geöffneten Dateien zu vermeiden. Es ist im allgemeinen einfacher, mehrere Dateien im selben Editor zu bearbeiten (Vim-Hilfe `:help buffers`).

**Dateinamen** Verwenden Sie unter *Linux*-/*Unix*-Systemen möglichst Dateinamen ohne Leerzeichen.


**Wechseldatenträger** Editieren Sie keine Dateien direkt auf einer Diskette oder einem USB-Stick. Das ist einerseits langsam, andererseits legt der Editor eine Auslagerungsdatei an, die offen bleiben kann, wenn der Editor nicht richtig beendet wurde. Möglicherweise kann dadurch die Einbindung des Datenträger (mount) anschließend nicht gelöst werden! Kopieren Sie die benötigten Dateien vor dem Gebrauch auf die Platte und danach auf den Datenträger zurück.

**Tastenkombination**   Diese Tastenkombination ist in der KDE-Oberfläche (*Linux*) für den Wechsel zum (meist nicht vorhandenen) neunten Desktop vorgesehen. Um die Tastenkombination zum Compilieren verwenden zu können, muss diese Belegung aufgehoben werden. Das geschieht mit Hilfe des KDE-Kontrollzentrums (Tastenkürzel-Serien).

**Tabulatorweite** Beim Wechsel zwischen verschiedenen Editoren oder IDEs geht ein Teil der Formatierung verloren, wenn die Tabulatorweite der Editoren unterschiedlich eingestellt sind. Übereinstimmende Einstellungen beheben diese Schwierigkeit. Der sicherste Weg ist die Umwandlung aller Tabulatoren in Leerzeichen vor dem letzten Speichern der Datei (Vim-Hilfe `:help retab` oder plug-in `textfilter`, Seite 21).

## 17 Weitere Informationsquellen

Die vorliegende Kurzanleitung führt in die grundlegende Bedienung des Editors ein. Es gibt jedoch weitere, reichhaltige Fähigkeiten und Bedienmöglichkeiten, deren Darstellung den hier gewählten Rahmen sprengen würde. Dazu zählen unter anderem eine Reihe weiterer Befehle, reguläre Ausdrücke (Suchen und Ersetzen) und die Erweiterbarkeit des Editors durch zusätzliche Skripte (plug-ins). Darüber hinaus gibt es eine Reihe von Konfigurationsoptionen zur Anpassung an die Benutzerbedürfnisse.

Die **Originaldokumentation** ist Bestandteil des Programmpaketes und kann über die Hilfefunktion (  , Menü oder Kommandozeile) eingesehen werden. Die Hilfetexte sind verlinkt; eine Suchfunktion ist vorhanden. Die Hilfe beschreibt alle Befehle und jede Einstellungs- und Programmiermöglichkeit und wenden sich daher an Anwender mit Grundkenntnissen.

Für den Anfänger ist eine Einführung vorhanden (VIM USER MANUAL). Weiterhin stehen die unten angeführten Bücher und Anleitungen zur Verfügung.

## Literatur

- [Gre] GREGOIRE, Laurent: *VIM Reference Card*. <http://tnerual.eriogerg.free.fr/vim.html>. – Die wichtigsten vim-Befehle auf 2 Seiten als Gedächtnisstütze (in mehreren Sprachen erhältlich).
- [LRH08] LAMB, Linda ; ROBBINS, Arnold ; HANNAH, Elbert: *Learning the vi and Vim Editors*. 7. Edition. O'Reilly, 2008. – ISBN 978-0-596-52983-3. – The standard guide for vi and the leading vi clone Vim.
- [Qua01] QUALINE, Steve: *Vi IMproved - Vim*. New Riders Publishing, 2001. – Umfangreiches Handbuch, das schrittweise in die Benutzung, Konfigurierung und Programmierung einführt. Empfehlenswert. Unter <http://vim.sourceforge.net/docs.php> als PDF-Datei erhältlich.
- [Rob00] ROBBINS, Arnold: *vi Editor kurz & gut*. 2. Auflage. O'Reilly, 2000. – ISBN 978-0-596-52983-3. – Kurzanleitung für den vi und die wichtigsten Abkömmlinge (insbesondere vim).
- [Vim] VIM: *The Vim (Vi IMproved) Home Page*. <http://vim.sourceforge.net>. – Homepage des vim-Projektes. Viele Verweise auf andere Seiten und auf zusätzliche Dokumentation. vim-Skripte zur Erweiterung des Editors, Tips, Neuigkeiten, Patches, ...
- [Wob07] WOBST, Reinhard: *vim GE-PACKT*. 3. Auflage. mitp-Verlag, Bonn, 2007. – ISBN 978-3-8266-1781-2. – Handliche Einführung in die Benutzung und in die vim-Skriptsprache.