

Inhaltsverzeichnis

1	Emacs-Konfiguration und -Programmierung	3
1.1	Grundlagen	4
	Konfigurationsdateien	4
	Verzeichnisse mit Lisp-Dateien	4
	Kommandozeilenoptionen	5
1.2	Makros	6
1.3	Programmiertechniken	7
	Einführung	7
	Online-Hilfe zur Lisp-Programmierung	8
	Aufruf von Lisp-Funktionen	9
	Variablen	10
	Definition eigener Funktionen	10
	Verzweigungen	11
	Schleifen	12
	Differenzierung zwischen GNU Emacs und XEmacs	12
1.4	Eigene Tastenkürzel	12
1.5	Beispiele zur Emacs-Programmierung	15
	Zwei Buchstaben vertauschen	15
	Abkürzungen automatisch laden und speichern	15
	Groß- und Kleinschreibung ändern	16
	Farbdarstellung	18
	Veränderung eines Bearbeitungsmodus	18
	Prozentuelle Cursorbewegung in großen Texten	19
	L ^A T _E X-Fontlock-Problem	20

Kapitel 1

Emacs-Konfiguration und -Programmierung

HINWEIS: Dieses Kapitel wurde aus der 4. Auflage des Buchs 'Linux: Installation, Konfiguration, Anwendung', Michael Kofler, Addison-Wesley 1999 übernommen. Das Kapitel wurde beginnend mit der 5. Auflage aus Platzgründen aus dem gedruckten Teil des Buchs eliminiert. Es wird *nicht* mehr weiter gewartet und beschreibt die Emacs-Programmierung auf der Basis der alten Versionen 20.2 (GNU Emacs) bzw. 20.4 (XEmacs).

Mehr Informationen zum Linux-Buch von Michael Kofler finden Sie hier:

<http://www.kofler.cc>

Die Defaulteinstellungen des Emacs – von der Farbgestaltung bis zu den Tastenkürzeln – können beinahe beliebig verändert werden. Dazu bieten sich zwei Vorgehensweisen an: die Verwendung der dafür vorgesehenen Menüs und Konfigurationsmasken (also Konfiguration per Mausclick), oder die direkte Veränderung der Konfigurationsdatei `.emacs` mit einem Editor.

Damit Sie die Möglichkeiten von `.emacs` voll ausnützen können, müßte in diesem Kapitel die Emacs-internen Programmiersprache Lisp samt allen verfügbaren Erweiterungsfunktionen und -variablen beschrieben werden. Dieses Thema gäbe ausreichend Stoff für ein ganzes Buch! Insofern ist dieses Kapitel also nur ein sehr bescheidener Versuch, eine erste, beispielorientierte Einführung zu geben.

Erschwerend kommt hinzu, daß die Lisp-Dialekte von GNU Emacs und XEmacs nicht vollständig kompatibel zueinander sind und daß die beiden Versionen oft unterschiedliche Erweiterungsfunktionen zur Verfügung stellen. Die in diesem Kapitel enthaltenen Basisinformationen gelten aber – soweit nicht explizit auf Unterschiede hingewiesen wird – für beide Emacs-Versionen (GNU Version 20.2, X Version 20.4).

1.1 Grundlagen

Konfigurationsdateien

Beim Start des GNU Emacs werden – soweit vorhanden – der Reihe nach drei Dateien gelesen:

```
/usr/share/emacs/site-lisp/site-start.el
~/.emacs
/usr/share/emacs/20.2/lisp/default.el
```

Beim XEmacs sind es sogar vier Dateien:

```
/usr/X11R6/lib/xemacs-20.4/lisp/site-start.el
~/.emacs
~/.xemacs-options
/usr/X11R6/lib/xemacs-20.4/lisp/default.el
```

Wenn die Programme je nach Distribution in einem anderen Verzeichnis installiert sind (üblich sind auch `/usr/lib` und `/usr/local`), gelten natürlich andere Pfadangaben. Verwenden Sie gegebenenfalls `locate`, um nach den Dateien zu suchen.

`site-start.el` ist für globale Einstellungen vorgesehen, die auf jeden Fall durchgeführt werden sollen. `.emacs` ist für persönliche Konfigurationseinstellungen. `default.el` gilt zwar wieder für alle Anwender, kann aber durch die Anweisung `(setq inhibit-default-init 1)` in `.emacs` individuell deaktiviert werden.

Die Datei `.xemacs-options` wird vom XEmacs automatisch erzeugt, sobald Sie zum erstenmal Veränderungen im OPTIONS-Menü durchführen und speichern. Der XEmacs fügt außerdem am Ende der Datei `.emacs` einige Zeilen an, so daß `.xemacs-options` automatisch vom XEmacs (nicht aber vom GNU Emacs) geladen wird.

HINWEIS

Mit SuSE-Linux wird eine eigene `.emacs`-Datei mitgeliefert (siehe Verzeichnis `/etc/skel`), die je nach Emacs-Version automatisch `.gnu-emacs` bzw. `.xemacs-custom` lädt. Zudem werden Änderungen durch `(Alt)+(X)` `customize-browse` automatisch in `.xemacs-custom` statt in `.emacs` gespeichert. Dieser Mechanismus ist allerdings nicht standardisiert.

Verzeichnisse mit Lisp-Dateien

Die meisten Emacs-Funktionen und insbesondere alle Emacs-Erweiterungen und Bearbeitungsmodi sind in der Emacs-internen Programmiersprache realisiert (einem Lisp-Dialekt). `.emacs` ist insofern also keine Besonderheit, sie gesellt sich einfach zum bereits vorhandenen Grundstock von Lisp-Dateien.

Die zahllosen Emacs-Lisp-Dateien befinden sich normalerweise in zwei Verzeichnissen. Das Emacs-Lisp-Verzeichnis enthält alle Lisp-Dateien, die unmittelbar zum Lieferumfang des Standard-Emacs gehören. Das Site-Lisp-Verzeichnis ist der geeignete Ort, um den Funktionsumfang des Emacs durch zusätzliche Dateien zu erweitern. (Prinzipiell könnten diese Dateien auch in das Emacs-Lisp-Verzeichnis installiert werden – dort gingen sie aber beim nächsten Update auf eine neuere Emacs-Version verloren.)

Der genaue Pfad dieser beiden Verzeichnisse ist distributionsabhängig – verwenden Sie gegebenenfalls `locate` zur Suche. Die folgenden Pfade gelten für die GNU-Emacs-Version 20.2, wie sie mit SuSE-Linux ausgeliefert wird.

```
/usr/share/emacs/20.2/lisp
/usr/share/emacs/site-lisp
```

Emacs-intern wird bei der Suche nach Lisp-Dateien die Variable `load-path` ausgewertet. Sie können sich den Inhalt dieser Variablen mit `(Strg)+(H)`, `(V)` `load-path` ansehen.

Um das Einlesen von Lisp-Dateien zu beschleunigen, können diese kompiliert werden. Das Kompilieren erfolgt direkt im Emacs mit dem Kommando `(Alt)+(X)` `byte-compile-file`. Der Emacs erzeugt dadurch aus der angegebenen `*.el`-Datei eine `*.elc`-Datei. Wenn der Emacs eine bestimmte Lisp-Datei sucht und sowohl eine `*.el`- als auch eine `*.elc`-Variante zur Verfügung steht, wird nur die `*.elc`-Variante berücksichtigt.

Bei den meisten Linux-Distributionen werden aus Platzgründen nur die kompilierten Lisp-Dateien installiert. Wenn Sie Interesse an den Quellcodes haben, müssen Sie die `*.el`-Dateien getrennt installieren. Die Quellcodes sind gute Beispiele für professionelle Lisp-Programmierung.

TIP

Neben den mitgelieferten Lisp-Dateien sind im Emacs-Lisp-Archiv Hunderte weitere Dateien gesammelt, mit denen dem Emacs besondere Funktionen beigebracht werden können. Bevor Sie sich also in die Lisp-Programmierung stürzen, sollten Sie einen Blick in dieses Archiv werfen. Vielleicht gibt es die Funktionen, die Sie brauchen, ohnedies schon.

<http://www.gnu.org/software/emacs/emacs.html>

<http://www.xemacs.org/elisp-archive.html>

<http://www.xemacs.org/elisp.html>

Kommandozeilenoptionen

Beim Starten des Editors erreichen Sie durch die Option `--no-init-file` oder kurz `-q`, daß die Konfigurationsdatei `.emacs` *nicht* geladen wird. Das ist sinnvoll, wenn bei der Erstellung der Konfigurationsdatei ein Fehler passiert ist und der Emacs nicht

<http://www.kofler.cc>

mehr normal bedient werden kann. Analog kann durch `--no-site-file` verhindert werden, daß `site-start.el` berücksichtigt wird.

Mit der Option `-u datname` wird statt `.emacs` eine andere Konfigurationsdatei geladen, mit `-l datname` eine zusätzliche Datei mit Lisp-Code. `-f funktion` führt unmittelbar nach dem Programmstart die angegebene Funktion aus.

Kommandozeilenoptionen

<code>-f funktion</code>	nach Start die angegebene Funktion ausführen
<code>-l datei</code>	zusätzlich zu <code>.emacs</code> weitere Datei ausführen
<code>--no-init-file</code> oder <code>-q</code>	Emacs starten, ohne <code>.emacs</code> und <code>default.el</code> auszuführen
<code>--no-site-file</code>	Emacs starten, ohne <code>site-start.el</code> auszuführen
<code>-u datei</code>	statt <code>.emacs</code> andere Datei ausführen

1.2 Makros

Makros stellen die Vorstufe zur richtigen Emacs-Programmierung dar. Mit Makros können Sie komplexe Kommandofolgen aufzeichnen und später wieder ausführen lassen. Die Aufzeichnung eines Makros beginnt mit `(Strg)+(X), (1)` und endet mit `(Strg)+(X), (1)`. Alle zwischen diesen beiden Kommandos durchgeführten Tastatureingaben werden aufgezeichnet. Mit `(Strg)+(X), (E)` kann dieses Makro ausgeführt werden.

Der Emacs speichert immer nur ein Makro. Sobald Sie ein neues Makro aufzeichnen, wird das alte gelöscht. Sie können aber vor dem Beginn einer neuen Aufzeichnung dem letzten Makro mit `(Alt)+(X) name-last-kbd-macro (←) name (←)` einen Namen geben. Dann wird das Makro unter diesem Namen gespeichert und kann jederzeit wieder mit `(Alt)+(X) name (←)` ausgeführt werden.

Allerdings werden auch benannte Makros nur so lange gespeichert, bis Sie den Emacs verlassen. Darüber hinaus ist es nicht möglich, einem eigenen Makro eine Tastenkombination zuzuweisen, um so einen bequemeren Aufruf zu ermöglichen. Diese Eigenschaften können Sie nur erreichen, wenn Sie die Definition des Makros und anschließend eine Verbindung zu einem Tastenkürzel in die Konfigurationsdatei `~/ .emacs` aufnehmen. Alternativ dazu können Sie auch eine eigene Makrodatei erstellen, die Sie dann aber mit `(Alt)+(X) eval-current-buffer (←)` manuell ausführen müssen (während `~/ .emacs` beim Start des Emacs automatisch ausgeführt wird).

Zur Übernahme von zuvor aufgezeichneten und benannten Tastaturmakros in eine Datei steht das Kommando `(Alt)+(X) insert-kbd-macro (←) name (←)` zur Verfügung. Dieses Kommando fügt den äquivalenten Lisp-Code zum Makro `name` in die aktuelle Datei ein. Das Kommando kann auch zum Einfügen des Lisp-Codes für alle anderen im Emacs definierten Kommandos eingesetzt werden. Allerdings sieht der

Code nicht gerade besonders übersichtlich aus, weil es darin nur so von unverständlichen Tastencodes wimmelt. Eine Veränderung vorhandener Makros ist daher kaum möglich.

Damit sieht die Vorgehensweise zur Erstellung eines Makros, das ständig zur Verfügung stehen soll, folgendermaßen aus:

- Makro mit $\overline{\text{Strg}}+\overline{\text{X}}$, $\textcircled{1}$ und $\overline{\text{Strg}}+\overline{\text{X}}$, $\textcircled{1}$ aufzeichnen.
- Makro mit $\overline{\text{Strg}}+\overline{\text{X}}$, $\textcircled{\text{E}}$ testen und eventuell nochmals aufzeichnen, wenn es nicht so funktioniert, wie es soll.
- Makro mit $\overline{\text{Alt}}+\overline{\text{X}}$ `name-last-kbd-macro` \leftarrow `name` \leftarrow benennen.
- Konfigurationsdatei `.emacs` aus dem Heimatverzeichnis laden.
- Makro dort mit $\overline{\text{Alt}}+\overline{\text{X}}$ `insert-kbd-macro` \leftarrow `name` \leftarrow einfügen, `~/.emacs` speichern.

1.3 Programmiertechniken

Einführung

Um Ihnen eine erste Vorstellung vom Aussehen einer Emacs-Konfigurationsdatei zu vermitteln, zeigen die folgenden Zeilen ein Beispiel für einen typischen Codeausschnitt: In den vier Zeilen wird eine neue Emacs-Funktion `jump-to-position` definiert, die den Cursor an die zuvor in Register Nr. 1 gespeicherte Position bewegt. Dieser Funktion wird das Tastenkürzel $\overline{\text{F6}}$ zugewiesen, um eine möglichst bequeme und effiziente Verwendung zu ermöglichen.

```
(defun jump-to-position()                ;Definition einer
  (interactive)                          ; neuen Funktion
  (jump-to-register 1))
(global-set-key [f6] 'jump-to-position) ;neues Kürzel F6
```

Während Sie die Datei `.emacs` bearbeiten, können Sie das Kommando $\overline{\text{Alt}}+\overline{\text{X}}$ `eval-current-buffer` \leftarrow ausführen. Durch dieses Kommando werden alle Lisp-Kommandos in der aktiven Datei ausgeführt. Das Kommando erspart Ihnen einen ständigen Neustart des Emacs, um eine geänderte Version von `.emacs` auszuprobieren.

VORSICHT

Wenn sich in `.emacs` ein Syntaxfehler befindet, ignoriert der Emacs die Datei ab der fehlerhaften Zeile. Unterhalb der Statuszeile wird zwar eine Fehlermeldung angezeigt, diese ist aber unsichtbar, wenn Sie beim Start eine Datei laden. (Dann wird in der Zeile eine Meldung über die letzte Aktion – z.B. über die erfolgreiche Syntaxhervorhebung – angezeigt.)

Mit anderen Worten: ein Syntaxfehler in `.emacs` bleibt leicht unbemerkt, und Sie wundern sich, warum weitere Änderungen in `.emacs` wirkungslos bleiben. Um dem Fehler auf die Spur zu kommen, laden Sie am besten `.emacs` in den Editor und führen die darin enthaltenen Kommandos mit `(Alt)+(X) eval-buffer` explizit aus. Die fehlerhafte Anweisung wird sofort identifiziert.

Sehr nützlich zum Testen einzelner Lisp-Anweisungen ist auch das Kommando `(Alt)+(X) eval-region` `(←)`. Damit werden nur die Anweisungen zwischen dem Markierungspunkt `(Strg)+(Leertaste)` und der aktuellen Cursorposition ausgeführt.

Lisp-Code ausführen

<code>(Alt)+(X) eval-current-buffer</code>	Lisp-Anweisungen im aktiven Puffer ausführen
<code>(Alt)+(X) eval-region</code>	Anweisungen zwischen Markierungspunkt und Cursor ausführen

Online-Hilfe zur Lisp-Programmierung

Die Online-Dokumentation des Emacs enthält eine Menge wertvoller Informationen zur Programmierung: `(F1)`, `(F)` *funktionsname* `(←)` gibt eine kurze Beschreibung der angegebenen Lisp-Funktion. `(F1)`, `(F)`, `(Tab)` liefert eine Liste *aller* zur Verfügung stehenden Lisp-Funktionen. Damit diese Liste betrachtet werden kann, muß das Kommando mit `(Strg)+(G)` abgebrochen werden. Anschließend können Sie mit `(Strg)+(X)`, `(B)*Com` `(←)` in den Puffer mit der Liste aller Funktionen wechseln. Wenn Sie die Liste öfter benötigen, sollten Sie den Puffer in einer Datei speichern `(Strg)+(X)`, `(W)`. Über das Tastenkürzel `(F1)`, `(V)` können Sie auf analoge Weise Informationen zu allen dem Emacs bekannten Variablen ermitteln.

Häufig möchten Sie wissen, wie eine Lisp-Funktion heißt, die mit einem bestimmten Tastenkürzel gestartet wird (etwa um diese Funktion in einer eigenen Funktion zu verwenden): In diesem Fall geben Sie `(F1)`, `(C)` *tastenkürzel* ein. Der Emacs zeigt dann in der untersten Bildschirmzeile den Namen der Funktion an.

Hilfe zu Funktionen und Variablen

<code>(F1)</code> , <code>(F)</code> <i>fname</i>	Informationen zur angegebenen Lisp-Funktion
<code>(F1)</code> , <code>(F)</code> , <code>(Tab)</code>	Liste aller bekannten Funktionen
<code>(F1)</code> , <code>(V)</code> <i>vname</i>	Informationen zur angegebenen Lisp-Variablen
<code>(F1)</code> , <code>(V)</code> , <code>(Tab)</code>	Liste aller bekannten Variablen
<code>(F1)</code> , <code>(C)</code> <i>tastenkürzel</i>	Name der zugeordneten Lisp-Funktion

Darüber hinaus gibt es zwei ausgezeichnete Programmierhandbücher zum GNU-Emacs: `elisp-manual` beschreibt auf rund 700 Seiten den GNU-Emacs-Lisp-Dialekt. `emacs-lisp-intro` gibt eine kompaktere Einführung in die Programmierung.

Die beiden Dokumentationspakete werden leider nur mit den wenigsten Distributionen mitgeliefert (und dann meistens in einer alten Version). Aktuelle Versionen in verschiedenen Formaten (PostScript, HTML) finden Sie aber natürlich im Internet:

<http://www.gnu.org/manual/manual.html>

Wenn Sie mit dem XEmacs arbeiten, ist eine umfangreiche Lisp-Sprachreferenz gleich in das Info-System integriert. Suchen Sie im Info-Startmenü nach `lispref!`

Aufruf von Lisp-Funktionen

Lisp-Funktionen müssen generell in runden Klammern angegeben werden. Der Funktionsname und die Parameter werden einfach durch Leerzeichen voneinander getrennt. Zur Lisp-typischen Anhäufung von Klammerebenen kommt es, wenn als Parameter einer Funktion wiederum andere Funktionen übergeben werden.

```
(set-input-mode (car (current-input-mode))
                (nth 1 (current-input-mode)) 0)
```

Diese Anweisung ermöglicht es, im Emacs deutsche Sonderzeichen über die Tastatur einzugeben. Der Funktion `set-input-mode` werden drei Parameter übergeben, von denen die beiden ersten wiederum durch andere Funktionen ermittelt werden: `current-input-mode` liefert eine Liste von vier Elementen, aus der `car` das erste und `nth` das zweite Element extrahiert (die Zählung beginnt mit 0; `(car x)` ist eine Kurzform für `(nth 0 x)`). In einer C-ähnlichen Programmiersprache würde das obige Kommando etwa so aussehen:

```
set-input-mode(car(current-input-mode()),
              nth(1, current-input-mode()), 0);
```

Alle Operationen werden generell in der Schreibweise (kommando parameter1 parameter2 ...) ausgeführt, auch ganz einfache Rechenoperationen, Zahlenvergleiche etc. Kommentare werden mit einem Semikolon eingeleitet.

```
(+ 2 3)           ; entspricht 2+3
(/ (+ a b) c)    ; entspricht (a+b)/c
(= c d)          ; entspricht c=d (gleich)
(<= e f)         ; entspricht e<=f (kleiner gleich)
(/= h i)        ; entspricht h!=i (ungleich)
```

Mehrere Anweisungen werden normalerweise einfach aneinandergereiht (jeweils in runden Klammern). Bei manchen Sprachelementen (insbesondere bei der Bildung von Verzweigungen mit `if`) ist es allerdings erforderlich, die Anweisungen in einem Block zusammenzufassen. Dazu wird die Funktion `progn` verwendet:

<http://www.kofler.cc>

```
(progn
  (funktion1 parameter1 parameter2 ...)
  (funktion2 parameter1 parameter2 ...)
  (funktion3 parameter1 parameter2 ...))
```

Ähnlich wie `progn` funktioniert auch `save-excursion`. Der Unterschied besteht darin, daß nach der letzten Anweisung im Block der Cursor an seine ursprüngliche Position zurückgestellt wird.

Variablen

Lokale Variablen werden mit der Funktion `let` eingeführt. Die Gültigkeit der Variablen beschränkt sich dabei auf die Klammerebene um `let`. Es handelt sich also um lokale Variablen. Die beiden Syntaxvarianten unterscheiden sich dadurch, daß die Variablen im ersten Fall nicht initialisiert werden.

```
(let (var1 var2 var3 ...) ;Variablen nicht
  (funktion1 parameter1 parameter2 ...) ; initialisieren
  (funktion2 parameter1 parameter2 ...)
  (funktion3 parameter1 parameter2 ...))

(let ((var1 wert1) (var2 wert2) ...) ;Variablen
  (funktion1 parameter1 parameter2 ...) ; initialisieren
  (funktion2 parameter1 parameter2 ...)
  (funktion3 parameter1 parameter2 ...))
```

Im Gegensatz zu anderen Programmiersprachen gibt es keine Variablentypen. Die Variablen können ohne Einschränkungen Integer-Zahlen, einzelne Zeichen, Zeichenketten, Listen etc. speichern. Nachdem Variablen definiert wurden, können sie ohne besondere Kennzeichnung in Anweisungen verwendet werden. Variablenzuweisungen werden mit der Funktion `setq` durchgeführt. Statt `setq` kann auch `set` verwendet werden, allerdings muß dem Variablennamen dann ein Hochkomma vorangestellt werden.

```
(setq var wert) ;entspricht var=wert
(set 'var wert) ;wie oben
(setq var (+ var1 var2)) ;entspricht var=var1+var2
```

Wenn mit `setq` einer nicht durch `let` eingeführten Variablen ein Wert zugewiesen wird, dann gilt diese Variable als global. Allerdings ist bei der Benennung globaler Variablen Vorsicht geboten, damit die Variable nicht mit einer bereits vorhandenen Emacs-Variablen kollidiert.

Definition eigener Funktionen

Eigene Funktionen werden mit `defun` definiert. Parameter werden in Klammern hinter dem Funktionsnamen angegeben. Das Schlüsselwort `interactive` macht die

Funktion allgemein verwendbar, so daß ein Funktionsaufruf via $(\text{Alt})+(\text{X})$ funktionsname möglich ist.

```
(defun funkl()                ;Funktion ohne Parameter
  (interactive)
  (funktion1 parameter1 parameter2 ...)
  (funktion2 parameter1 parameter2 ...)
  (funktion3 parameter1 parameter2 ...))
```

Wenn der Anwender nach dem Start einer Funktion Werte für die Parameter eingeben soll, muß in der *interactive*-Anweisung eine Zeichenkette angegeben werden, die die Parameter beschreibt:

```
(defun funkl(p1 p2 p3)      ;Funktion mit Parametern
  (interactive "nZeilennummer: \nsText: \nfDatei:")
  (funktion1 parameter1 parameter2 ...)
  (funktion2 parameter1 parameter2 ...)
  (funktion3 parameter1 parameter2 ...))
```

Die *interactive*-Zeichenkette ist mit `\n` in drei Teile aufgegliedert, die jeweils aus einem Zeichen für den Eingabetypus und einem kurzen erklärenden Text bestehen. Im obigen Beispiel fordert der Emacs den Anwender nach dem Start der Funktion auf, eine Zeilennummer, einen beliebigen Text und einen Dateinamen einzugeben. In *interactive* sind unter anderem folgende Zeichen für Eingabetypen vorgesehen: `n` (Zahl), `s` (Zeichenkette), `b` (existierender Textpuffer), `B` (neuer Textpuffer), `f` (existierende Datei), `F` (neue Datei).

Der Rückgabewert einer Funktion ergibt sich aus dem Rückgabewert der letzten Anweisung, die innerhalb der Funktion ausgeführt wurde.

Verzweigungen

Verzweigungen werden am häufigsten mit `if` gebildet. Die Syntax sieht dabei folgendermaßen aus:

```
(if (bedingung)
    (progn                ;then-Block
     (funktion1 ...)
     (funktion2 ...) ...)
    (progn                ;optionaler else-Block
     (funktion1 ...)
     (funktion2 ...) ...)
  )
```

Wenn der *then-Block* nur aus einer einzigen Funktion besteht, kann man auf `progn` verzichten. Bedingungen werden zumeist durch Vergleiche gebildet, etwa `(< a 5)` für `a<5`. Bedingungen können durch `(or bed1 bed2 bed3 ...)`, `(and ...)` und `(not bed)` verknüpft werden.

Schleifen

Die einfachste Schleifenform wird mit `while` eingeleitet. Im Gegensatz zu `if` muß der Schleifenkörper nicht mit `progn` angegeben werden. Die Syntax sieht folgendermaßen aus:

```
(while (bedingung)
  (funktion1 ...)
  (funktion2 ...) ...)
```

Differenzierung zwischen GNU Emacs und XEmacs

Beide Emacs-Versionen werten die Datei `~/ .emacs` aus. Manche Lisp-Funktionen sind allerdings versionsabhängig, d.h. bei der parallelen Verwendung beider Versionen kann es zu Konflikten und Syntaxfehlern kommen. Um das zu vermeiden, können Sie den Code in `~/ .emacs` in drei Teile zerlegen: einen gemeinsamen Abschnitt und je einen Abschnitt für jeden Editor. Das erforderliche Codefragment sieht in etwa so aus:

```
; Datei ~/ .emacs
; gemeinsamer Teil
... code ...
(defvar running-xemacs (string-match "XEmacs|Lucid" emacs-version))
; nur GNU-Emacs
(cond ((not running-xemacs)
  ... code ...
))
; nur XEmacs
(cond (running-xemacs
  ... code ...
))
```

1.4 Eigene Tastenkürzel

Eigentlich ist es nicht schwierig, eigenen oder vordefinierten Kommandos ein neues Tastenkürzel zuzuweisen. Wenn Sie also ein Emacs-Kommando häufig benötigen und sich bisher über dessen umständlichen Aufruf mit `(Alt)+(X)` kommando geärgert haben, besteht jetzt Hoffnung auf Abhilfe.

Dem Kommando `global-set-key` werden zwei Parameter übergeben: das in eckige Klammern gestellte Tastenkürzel und der Funktionsname, dem ein einfaches Anführungszeichen vorangestellt wird.

```
(global-set-key [f1] 'goto-line)
```

Was hier ganz einfach aussieht, ist in der Praxis allerdings mit einer Menge Schwierigkeiten verbunden:

- GNU Emacs und XEmacs haben eine unterschiedliche Syntax für zusammengesetzte Tastenkürzel. $(\text{Strg})+(\text{X})$ wird im GNU Emacs als `[?\C-x]`, im XEmacs dagegen als `[control x]` dargestellt.
- Die Defaultbedeutung mancher Tastenkürzel ändert sich beinahe bei jeder Emacs-Version. (Besonders anfällig für Änderungen sind (Entf) und (Backspace) . Kaum glaubt man, endlich eine vernünftige Konfiguration gefunden zu haben, kommt die nächste Version und macht alle Mühen wieder zunichte ...)
- Welchen Code der X-Server beim Drücken eine Tastenkombination an den Emacs liefert, hängt stark von der X-Konfiguration ab (insbesondere von `xmodmap`). Aus diesem Grund kann es passieren, daß sich (Entf) selbst bei vollkommen gleicher Emacs-Konfiguration in zwei Linux-Distributionen unterschiedlich verhält!
- Manche Tastenkombinationen werden von Linux, dem X-Server oder vom X-Windowmanager beansprucht und können dann im Emacs nicht verwendet werden (etwa $(\text{Strg})+(\text{Alt})+(\text{Fn})$ zum Konsolenwechsel).
- Wenn der Emacs in einer Textkonsole verwendet wird, können viele Tastenkombinationen überhaupt nicht verarbeitet werden. Die Bedeutung der Tastenkürzel hängt jetzt zudem vom Programm `loadkeys` ab. (Tastenkürzel innerhalb von Textkonsolen werden hier nicht weiter behandelt. Das Kapitel geht davon aus, daß Sie den Emacs unter X verwenden.)
- Schließlich müssen die Restriktionen des Emacs beachtet werden: Viele Tastenkürzel sind schon vorbelegt, und es ist nicht immer sinnvoll, diese Kürzel einfach durch eigene Kürzel zu ersetzen. Manche Tasten aktivieren eine eigene Kürzelebene und dürfen nur in Kombination mit weiteren Tasten eingesetzt werden (etwa $(\text{Strg})+(\text{X})$, $(\text{Strg})+(\text{C})$, $(\text{Strg})+(\text{H})$).

Aus all diesen Gründen ist es vollkommen unmöglich, eine `.emacs`-Datei zu schreiben, die wirklich auf jedem Rechner gleich funktioniert.

TIP

Bitte werfen Sie auch einen Blick auf die folgende Webseite. Die dort angebotenen Informationen sind aktueller und vollständiger als dieser Text:

<http://tiny-tools.sourceforge.net/emacs-keys.html>

TIP

Um festzustellen, wie der Emacs intern auf das Drücken einer Taste reagiert, können Sie die Tastenkombination $(\text{Strg})+(\text{H}), (\text{C}) (\text{Taste})$ verwenden. Der Emacs zeigt daraufhin die interne Zeichenkette der gedrückten Taste sowie die aktuelle Bedeutung dieses Tastenkürzels an. Beispielsweise führt $(\text{Strg})+(\text{H}), (\text{C}), (\text{F5})$ zur Meldung `'f5 is undefined'`.

$(\text{Strg})+(\text{H}), (\text{C})$ ist übrigens auch eine gute Möglichkeit, das Emacs-interne Symbol für Sondertasten zu ermitteln. $(\text{Strg})+(\text{H}), (\text{C}), (\text{Pos1})$ liefert beispielsweise `'home runs ...'`. Damit wissen Sie, daß (Pos1) durch `[home]` repräsentiert wird.

GNU-Emacs-Syntax für Tastenkürzel

Das gesamte Kürzel wird in eckige Klammern gestellt. Innerhalb der Klammern können entweder einzelne Zeichen (mit vorangestelltem ?) oder Schlüsselwörter für Funktionstasten (ohne ?) angegeben werden.

Kombinationen mit **(Strg)** werden durch `\C-z` angegeben, wobei für *z* ein beliebiger Buchstabe eingesetzt werden muß (aber keine Ziffer, kein Sonderzeichen). Analog werden die Buchstaben S für **(Shift)**, M für Meta (**(Esc)**) und A für **(Alt)** verwendet. Bei Funktionstasten kann auf den Backslash (\) vor den Buchstaben C, S, M und A verzichtet werden (also `[S-f1]`). Einige weitere Beispiele sollen die Syntax verdeutlichen:

GNU-Emacs-Tastenkürzel			
<code>[?a]</code>	(A)	<code>[?\C-x ?y ?z]</code>	(Strg)+X, Y, Z
<code>[f1]</code>	(F1)	<code>[?\C-x ?Y ?\M-z]</code>	(Strg)+X, (Shift)+Y, (Alt)+Z
<code>[?\C-x]</code>	(Strg)+X	<code>[?\C-x f1]</code>	(Strg)+X, F1
<code>[?\M-w]</code>	(Alt)+W	<code>[S-right]</code>	(Shift)+→
<code>[?\e]</code>	(Esc)		

XEmacs-Syntax für Tastenkürzel







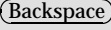

Beim XEmacs müssen Tastenkombinationen zusätzlich in runde Klammern gesetzt werden. Die Tasten von Buchstaben werden einfach durch den Buchstaben codiert, wobei die Groß- und Kleinschreibung relevant ist. Die Zustandstasten werden durch die Schlüsselwörter `shift`, `control` und `meta` (für **(Alt)**) ausgedrückt. Zwischen den Schlüsselwörtern ist wegen der Klammerung kein Bindestrich mehr erforderlich. Generell führen diese Regeln zu besser lesbarem Code. Auch hierzu einige Beispiele:

XEmacs-Tastenkürzel	
<code>[a]</code>	(A)
<code>[A]</code>	(Shift)+A
<code>[(meta a)]</code>	(Alt)+A
<code>[(control x) (meta a)]</code>	(Strg)+X, (Alt)+A
<code>[(shift end)]</code>	(Shift)+Ende

Tastencodes wichtiger Sondertasten

Um Ihnen allzuviel Arbeit beim Experimentieren mit Tastenkombinationen zu ersparen, gibt die folgende Liste eine Hilfestellung für die wichtigsten Sondertasten unter X. Die Liste gilt gleichermaßen für den GNU Emacs und den XEmacs. Beachten Sie, daß je nach Linux-Distribution und `xmodmap`-Einstellung und den verwendeten Versionen durchaus andere Ergebnisse möglich sind.

Tastencodes von Sondertasten

	[up]		[prior]
	[down]		[next]
	[left]		[home]
	[right]		[end]
	[delete] oder [deletechar] oder [DEL]		
	[backspace] oder [DEL]		
	[insert] oder [insertchar]		
	[f1] bis [f12]		

1.5 Beispiele zur Emacs-Programmierung

Zwei Buchstaben vertauschen

Mit der Tastenkombination $(\text{Strg})+(\text{T})$ können Sie den Buchstaben an der Cursorposition und den vorangehenden Buchstaben vertauschen. Das ist an sich in Ordnung – nur war ich es von einem anderen Texteditor seit Jahren gewöhnt, den Buchstaben an der Cursorposition mit dem *nächsten* Buchstaben zu vertauschen – und zwar nicht mit $(\text{Strg})+(\text{T})$, sondern mit (F11) . Um dieses Verhalten auch dem Emacs beizubringen, ist das folgende, leicht verständliche Makro entstanden:

```
(defun swap-char()                ;zwei Buchstaben an der Cursorposition
  (interactive)                  ;vertauschen
  (save-excursion
    (forward-char)
    (transpose-chars 1)))
(global-set-key [f11]             'swap-char) ;F11 unter X
```

Kurz zur Funktion: `save-excursion` speichert die aktuelle Cursorposition. Anschließend wird der Cursor mit `forward-char` um ein Zeichen weiterbewegt und die eingebaute Funktion `transpose-chars` zum Vertauschen der letzten beiden Buchstaben aufgerufen. Mit `global-set-key` wird die neue Funktion der Taste (F11) zugeordnet.

Abkürzungen automatisch laden und speichern

Mit $(\text{Alt})+(\text{X})$ `edit-abbrevs` können Sie Abkürzungen definieren. Die Abkürzungsdefinitionen werden allerdings nicht automatisch geladen bzw. gespeichert, wenn Sie den Emacs starten bzw. verlassen. Genau das bewirken die drei folgenden Zeilen in `.emacs`:

<http://www.kofler.cc>

```
(setq-default abbrev-mode t)           ;Abbrev-Mode aktivieren
(read-abbrev-file "~/.emacs-abbrev") ;~/.emacs-abbrev laden
(setq save-abbrevs t)                   ;Änderungen autom. speichern
```

`setq-default` weist einer Variablen einen Defaultwert zu, wobei die Zuweisung mit `t` dem Wahrheitswert `true` anderer Programmiersprachen entspricht.

Über die automatische Expansion von Abkürzungen kann man geteilter Meinung sein. Es kann ziemlich irritierend sein, wenn der Emacs irgendwelche Zeichenkombinationen plötzlich unbeabsichtigt als Abkürzungen erkennt. Wenn Sie Abkürzungen lieber manuell erweitern möchten, können Sie die erste Anweisung aus dem obigen Listing entfernen und statt dessen ein leicht erreichbares Tastenkürzel für die vorhandene Funktion `expand-abbrev` definieren. Damit werden Abkürzungen nur noch manuell durch das Drücken von **(F3)** erweitert.

```
(global-set-key [f3] 'expand-abbrev) ;Abkürzung manuell erweitern
```

Noch attraktiver wird der Umgang mit Abkürzungen durch das folgende Makro: `expand-abbrev-or-dabbrev` versucht zuerst, eine Abkürzung aus der Abkürzungstabelle zu verwenden. Wenn dort keine geeignete Abkürzung gefunden wird, ruft das Makro `dabbrev-expand` auf, um eine dynamische Expansion durchzuführen. Damit kann **(F3)** gleichzeitig für vordefinierte Abkürzungen und zur dynamischen Expansion verwendet werden.

```
(defun expand-abbrev-or-dabbrev()      ;Expansion von Abkürzung mit F3
  (interactive)
  (if (not (expand-abbrev))           ;falls keine Abkürzung existiert
      (progn
        (dabbrev-expand nil)))       ;dynamische Expansion
      (global-set-key [f3] 'expand-abbrev-or-dabbrev))
```

Das Beispiel zeigt gleichzeitig die Definition einer neuen Lisp-Funktion: Die Funktion wird mit dem Schlüsselwort `defun` eingeleitet. Durch das Kommando `interactive` wird die Funktion als Emacs-Kommando angemeldet, das durch **(Alt)+(X)** wie alle anderen Kommandos aufgerufen werden kann.

Groß- und Kleinschreibung ändern

Der Emacs kennt unzählige Kommandos, um einen Buchstaben, ein Wort oder einen anderen Bereich in Groß- oder in Kleinbuchstaben umzuwandeln. Was fehlt, ist jedoch ein Kommando, das die vorhandene Schreibweise einfach umkehrt: Wenn ein Buchstabe bisher groß geschrieben war, soll er durch einen kleinen Buchstaben ersetzt werden und umgekehrt.

Die resultierende Funktion `change-case` ist schon ein wenig komplexer. Mit `let` wird die Variable `zeichen` definiert, in der mit `char-after(point)` das Zeichen an der aktuellen Cursorposition gespeichert wird. In der folgenden `if`-Abfrage wird

getestet, ob der Code dieses Zeichens größer als 64 ist. Das ist für alle Buchstaben (und natürlich auch für einige Sonderzeichen) der Fall. Wenn dem so ist, wird der mit `progn` geklammerte Block ausgeführt.

Dort wird das sechste Bit des Zeichencodes geändert (XOR 32). Damit wird aus A (Code 65) ein a (Code 97) und umgekehrt. Dieses Zeichen wird anschließend mit `insert-char` an der Cursorposition eingefügt. Das ursprüngliche Zeichen wird gelöscht.

Der `else`-Teil der `if`-Abfrage wird nur dann ausgeführt, wenn das Zeichen unterhalb des Cursors einen Code kleiner gleich 64 hatte (Ziffern, Leerzeichen, Zeilenumbruch, Punkt, Komma etc.). In diesem Fall wird der Cursor um ein Zeichen nach vorn bewegt. Da der `else`-Block aus nur einer einzigen Funktion besteht, ist kein `progn`-Block erforderlich.

Die neue Funktion wird anschließend der Funktionstaste `(F12)` zugeordnet. Wenn `(F12)` mehrfach hintereinander gedrückt wird, bewegt sich der Cursor vorwärts über den Text und ändert alle Buchstaben. Um eine solche Änderung rückgängig zu machen, muß `(F12)` über dem betreffenden Buchstaben einfach nochmals gedrückt werden. Das Kommando funktioniert übrigens auch für deutsche Sonderzeichen.

```
(defun change-case()
  (interactive)
  (let ((zeichen (char-after (point))))
    (if (> zeichen 64)
      (progn
        (setq zeichen (logxor zeichen 32))
        (insert-char zeichen 1)
        (delete-char 1)
        )
      (forward-char 1)
      )
    )
  )
(global-set-key [f12] 'change-case)
```

Die folgende Funktion zeigt, daß eigene Funktionen auch in anderen Funktionen aufgerufen werden können. Das Kommando speichert mit `point-to-register` die aktuelle Cursorposition im Register mit der Nummer 2, springt an den Beginn eines Wortes, ändert die Groß-/Kleinschreibung des ersten Buchstabens und springt anschließend mit `jump-to-position` zurück an die gespeicherte Position. Statt der beiden Funktionen `point-to-register` und `jump-to-position` hätte natürlich wie in einem der vorangegangenen Beispiele ein `save-excursion`-Block gebildet werden können.

```
(defun change-word-case()
  (interactive)
  (point-to-register 2)
  (backward-word 1)
```

```
;Groß- und Kleinschreibung des 1.
;Zeichens eines Wortes verändern
;aktuelle Position speichern
;an den Wortanfang springen
```

```
(change-case) ; obige Funktion aufrufen
(jump-to-register 2) ; zurück zur gespeicherten Pos.
(global-set-key [f9] 'change-word-case)
```

Farbdarstellung

Im Fontlock-Modus ordnet Emacs Textausschnitte gemäß ihrer Syntax verschiedenen Textattributen wie 'bold', 'font-lock-keyword-face', 'font-lock-comment-face' usw. zu. Mit diversen `set-face-xxx`-Kommandos können Sie das Aussehen dieser Textattribute verändern: Sie können beispielsweise einstellen, daß Texte mit dem Attribut 'bold' vom Emacs grün und kursiv dargestellt werden. Auf diese Weise können Sie die zum Teil recht blassen Defaulteinstellungen von Emacs durch kräftigere Farben ersetzen.

Die Bedingung `(and window-system` bewirkt, daß die nachfolgenden Kommandos nur ausgeführt werden, wenn der Emacs unter X läuft (nicht in einer Textkonsole). Diese Sicherheitsmaßnahme ist für den GNU Emacs gedacht, der in Textkonsolen keine Farben darstellen kann.

```
(and window-system ; nur unter X
  (progn
    (set-face-foreground 'bold "red") ; fett rot
    (set-face-underline-p 'underline nil) ; nicht unterstreichen
    (set-face-foreground 'underline "blue"))) ; dafür blau
```

Für die Einstellung der Farben im Info-Modus gilt eine andere Syntax. Die folgenden Zeilen wurden mit Hilfe von `(Alt)+(X) customize-browse` erstellt.

```
(custom-set-faces
 '(info-xref ((t (:bold t :foreground "blue"))))
 '(info-node ((t (:bold t :italic t :foreground "red")))))
```

Veränderung eines Bearbeitungsmodus

Wenn Sie eine bestimmte Änderung am Verhalten des Emacs nicht generell, sondern nur für einen bestimmten Bearbeitungsmodus erreichen möchten, müssen Sie mit einer `mode-hook`-Variablen arbeiten. Der in diesen Variablen gespeicherte Code wird automatisch ausgeführt, sobald der Bearbeitungsmodus aktiviert wird. `add-hook` fügt an den schon vorhandenen Inhalt einer Hook-Variable eine neue Funktion an.

Die folgenden Zeilen verändern den $\text{T}_{\text{E}}\text{X}$ -Modus des Emacs (gilt nicht für AUC $\text{T}_{\text{E}}\text{X}$): Mit `auto-fill-mode` wird jedesmal, wenn der $\text{T}_{\text{E}}\text{X}$ -Modus aktiviert wird, auch der Fließtextmodus aktiviert. Damit der Zeilenumbruch in diesem Modus nicht bereits in der 70. Textspalte erfolgt (Defaulteinstellung), wird die Variable `fill-column` auf 79 gesetzt. Die letzte Veränderung betrifft die Eingabe des Anführungszeichens ("):

Der Emacs hat im \TeX -Modus die unangenehme Eigenschaft, daß er dieses Zeichen abwechselnd in `'` und ``` umwandelt. Um das zu vermeiden, wird den Variablen `tex-open-quote` und `tex-close-quote` das Zeichen `"` (Code 34) zugewiesen.

Jetzt bleibt nur noch das Schlüsselwort `lambda` zu erklären: Die Anweisung definiert ähnlich wie `defun` eine Funktion. Der Unterschied besteht darin, daß die Funktion nicht benannt werden muß. Der vorangestellte Apostroph `'` bewirkt zudem, daß die nachfolgenden Zeilen nicht sofort ausgewertet, sondern als Code gespeichert werden.

```
(add-hook 'tex-mode-hook          ;Einstellungen für den TeX-Modus
  (function (lambda ()
    (auto-fill-mode)              ;Fließtextmodus
    (setq fill-column 79)         ;volle Breite
    (setq tex-open-quote 34)      ;kein Schabernack mit "
    (setq tex-close-quote 34)
    (and window-system
      (progn
        (font-lock-mode 1)        ;Farbdarstellung
        (set-face-foreground 'font-lock-comment-face "blue")
        (set-face-foreground 'font-lock-string-face "red")
        (set-face-foreground 'font-lock-keyword-face "magenta")))
      )
    )))
```

Prozentuelle Cursorbewegung in großen Texten

Der Emacs zeigt in der Statuszeile ständig an, in welchem prozentuellen Bereich des Textes Sie sich befinden (also etwa 50 Prozent für die Mitte). Leider fehlt im Textmodus ein Kommando, um den Cursor rasch an eine Position zu bewegen, die durch einen Prozentwert ausgedrückt wird. (Bei der Verwendung des Emacs unter X fehlt dieses Kommando weniger als in der Textversion, weil dort eine komfortable Navigation in umfangreichen Dokumenten mit der vertikalen Schiebepalken erfolgt.)

Bei der Ausführung des Kommandos `goto-percent` wird der Anwender zuerst aufgefordert, einen Prozentwert (0 bis 100) einzugeben. Der Wert wird in der Variablen `p` (dem Parameter der Funktion) gespeichert. `goto-char` bewegt den Cursor anschließend an die gewünschte Position, die durch die Berechnung $(\text{point-max}) * p / 100$ ermittelt wird. Dabei liefert die Funktion `point-max` das letzte Zeichen des Textes.

```
(defun goto-percent (p)                ;goto Prozentwert (0-100)
  (interactive "nProzent (0 bis 100): ") ;[Idee: Cameron, Rosenblatt:
  (goto-char (/ (* p (point-max)) 100)) ; Learning GNU Emacs]
  (beginning-of-line))                ;Cursor an Zeilenanfang
(global-set-key [f8] 'goto-percent)
```

Die obige Funktion hat den Nachteil, daß sie bei Texten, die größer 80 kBytes nicht mehr zuverlässig funktioniert. Da unter Emacs-Lisp Zahlen nur mit 24 Bit Genauigkeit dargestellt werden, kommt es bei der Berechnung der neuen Position zu einem Überlauf. Daher muß die Berechnungsreihenfolge umgedreht werden: $(\text{point-max})/100*p$. Diese neue Formel hat allerdings den Nachteil, daß bei kleinen Texten starke Rundungsfehler auftreten. Die neue Funktion verwendet aus diesem Grund je nach Textlänge unterschiedliche Formeln:

```
(defun goto-percent (p)                                ;goto Prozentwert (0-100)
  (interactive "nProzent: ")
  (if (> (point-max) 80000)
      (goto-char (* (/ (point-max) 100) p)) ;kein Überlauf: (max/100)*p
      (goto-char (/ (* p (point-max)) 100))) ;kein Fehler.: (max*p)/100
  (beginning-of-line))                                ;Cursor an Zeilenanfang
(global-set-key [f8] 'goto-percent)
```

LaTeX-Fontlock-Problem

Wenn der Fontlock-Modus gemeinsam mit dem LaTeX-Modus verwendet wird, tritt manchmal das Problem auf, dass der Emacs bei der Syntaxanalyse durcheinander gerät und große Textbereiche falsch kennzeichnet. Die Ursache ist zumeist ein einzelnes $\$$ -Zeichen, das in einer verbatim-Umgebung oder in einem einzelnen `\verb`-Kommando verwendet wird.

Das Problem lässt sich lösen, wenn Sie in der Datei `font-lock.el` eine Zeile ändern (das Zeichen `\` wird durch ein Leerzeichen ersetzt) und damit auf die Hervorhebung von Formeln in LaTeX-Texten verzichten. Damit die Änderung wirksam wird, müssen Sie `font-lock.elc` umbenennen bzw. die geänderte Datei `font-lock.el` neu kompilieren. Falls in Ihrem Lisp-Verzeichnis nur `*.elc`-Dateien installiert sind, müssen Sie vorher die Emacs-Lisp-Quelldateien installieren. (Dabei handelt es sich bei vielen Distributionen um ein eigenes Paket.)

```
; Änderung in /usr/X11R6/lib/xemacs-21.1/lisp/packages/font-lock.el
; ursprünglich:
; (put 'tex-mode 'font-lock-defaults
;      '(tex-font-lock-keywords nil nil ((? $ . "\"))))
; neu:
(put 'tex-mode 'font-lock-defaults
     '(tex-font-lock-keywords nil nil ((? $ . " "))))
```

Stichwortverzeichnis

Die Einträge im Stichwortverzeichnis sind nach dem ersten Buchstaben sortiert. Daher finden Sie die Einträge zum \LaTeX -Kommando `\end`, zu den Dateien im Verzeichnis `/etc` und zur Konfigurationsdatei `.emacs` alle unter dem Buchstaben E.

D

`default.el` 4

E

Elisp 4

 Beispiele 15

 Programmiertechniken 7

Emacs

 Abkürzungen 15

 Beispiele zur Programmierung 15

 Compiler 5

 eigene Tastenkürzel 12

 Farben 18

 Konfiguration 4

\LaTeX -Modus (Fehler) 20

 Lisp 7

 Lisp-Archiv 5

 Lisp-Verzeichnis 4

 Makros 6

 Programmierung 4

 prozentuelle Cursorbewegung 19

`site-lisp-Verzeichnis` 4

`.emacs` 4

`.emacs` 4

G

`.gnu-emacs` 4

I

Info

 Emacs-Farben 18

Internet

 Emacs-Lisp-Archiv 5

K

Konfiguration

 Emacs 4

L

\LaTeX

 Emacs (Fontlock-Modus) 20

M

Makros

 Emacs 6

S

`site-start.el` 4

`site-lisp-Verzeichnis` 4

T

Tastatur

 im Emacs 12

Tastenkürzel

 im Emacs 12

`tex-mode-hook` 18

X

`.xemacs-custom` 4

`.xemacs-options` 4