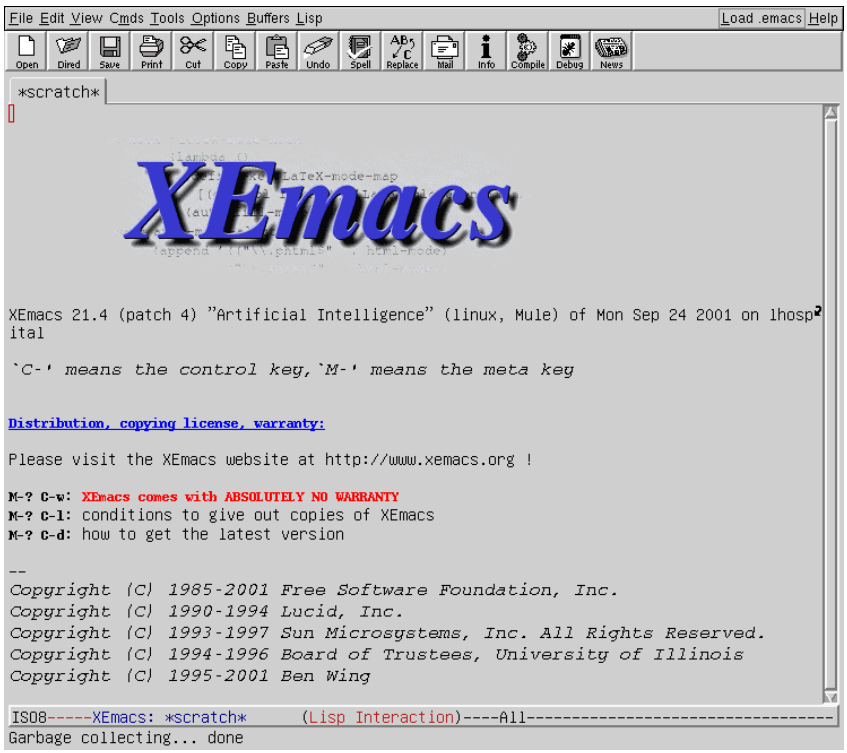
Eine pragmaprogrammatische Einleitung mit vielen Beispielen und bunten Bildern in etwas mehr als nur einen Texteditor. http://www.skamphausen.de/xemacs



Version – 2.04

XEmacs auf Deutsch

Stefan Kamphausen

13. Februar 2005

Gesetzt, getippt und gestaltet mit Freier Software (LaTeX, XEmacs und Gimp).

Inhaltsverzeichnis

1. Vorneweg

1.1. Er war's, er war's – Ich war's, ich war's

Lasst mich ganz zu Beginn eine wichtige Sache loswerden. Verantwortlich für den ganzen Kram hier bin *ich*, Stefan, und alles, was ich hier so wiedergebe, funktioniert für mich auf meinen Systemen.

Wenn es bei der Verwendung der hier dargebotenen Code-Kunststückchen zu irgendwelchen Problemen kommt, stürmt bitte nicht gleich zu den jeweiligen Newsgroups oder (noch schlimmer) direkt zu den Entwicklern. Ich muss zumindest so weit Verantwortung übernehmen, dass ich eventuell überholten oder schlicht falschen Informationen und Beispielen hier selber nachgehen muss. Also: schickt erstmal eine EMail an mich.

Dazu auch gleich noch: ich setze nur ein einziges Betriebssytem ein: Linux (ich habe Erfahrungen mit SuSE, RedHat und Gentoo). Fragen nach der Anwendbarkeit und Konfiguration auf anderen Systemen kann ich nicht beantworten. Ich nehme aber gerne entsprechende Kapitel hier mit auf, wenn sie jemand bereitstellt und dafür auch gerade steht.

1.2. Er war's, er war's – Du warst's, du warst's

Nur für die Leser, die es nicht ohnehin als selbstverständlich ansehen …

Jeglicher Code und jegliche Anleitung hier wird ohne Verantwortung oder Sicherheit für die Datenintegrität auf Seiten des Lesers abgegeben. Jeder Anweder und jede Anwenderin ist ganz alleine selber dafür verantwortlich, dass seine/ihre Dateien keinen Schaden nehmen, er/sie keine Mails verliert oder sonstwie inhaltlichen Schaden davon trägt. Wer das nicht einsieht oder versteht, sollte am Ende dieses Satzes aufhören zu lesen und das Thema XEmacs einfach ganz schnell vergessen.

So, und jetzt kann es losgehen.

2. `xemacs &`

Der Titel diese Kapitels zeigt eine folgenschwere Eingabezeile in einer Shell. Die Eingabe des Befehls `xemacs &` startet (im Hintergrund) einen XEmacs und damit eine Maschinerie, die ihresgleichen sucht[1] Das Ergebnis (sofern nicht bereits irgendwelche Konfigurationen vorgenommen wurden) sieht so aus, wie auf dem Titelblatt dieser Dokumentation.

Der XEmacs ist im Kern eigentlich ein LISP-Interpreter und eine Anzeige-Maschine, die einige wenige grundlegende Texteditor-Funktionalitäten liefert. Es mag nicht weiter verwundern, dass es eine solche grundlegende Funktion gibt, die bei Drücken der Q-Taste den Buchstaben „Q“ an aktueller Position einfügt, dass jedoch die Funktion zum Springen in die nächste Zeile *nicht* zu diesen Grundfunktionen zählt sollte schon ein wenig verwundern. Die Erklärung ist einfach die, dass diese Funktion in LISP, genauer in Emacs Lisp *Elisp* (noch genauer in der XEmacs-Version von Elisp) geschrieben ist.

Und eben das zeichnet den XEmacs aus. Möglichst viele Funktion sollen in ELisp realisiert werden. Das ergibt eine Flexibilität, die sich dem angehenden XEmacs-Adepten noch nicht erschließt, auf die der alte Guru aber unter keinen Umständen jemals wieder verzichten mag.

Aus diesem Grunde an dieser Stelle ein ernstzunehmende Warnung: Hat man einmal den nicht ganz einfachen Einstieg genommen, wird sich so ziemliche jede andere Komponente, die ein Editieren von Text zulässt (Formulare in Browsern, spezialisierte IDEs für diverse Programmiersprachen, EMail-Programme o. ä.) mickrig bis unterträglich anfühlen.

In der vorliegenden Dokumentation wollen wir uns den XEmacs einmal genauer anschauen und im Vorbeigehen noch einiges über Lisp lernen.

2.1. Die Maschine läuft an

Zu Beginn nehmen wir jedoch den Start des XEmacs mal etwas genauer unter die Lupe. Der Kern des XEmacs ist in C geschrieben und somit für den unbedarften Anwender eher uninteressant. Das macht aber auch nichts weiter, denn wir werden damit nicht großartig in Berührung kommen. Es soll reichen, zu wissen, dass dieser Kern am Anfang, also direkt nach dem Abfeuern obiger Kommandozeile gestartet wird und sich ganz viele Systembibliotheken einliest. Dieser Kern ist es auch, der auf verschiedene Architekturen, wie Linux, BSD oder auch Microsoft Windows, portiert werden muss – ein weiterer Grund dafür, ihn möglichst schmal zu halten.

Nachdem sich der XEmacs in unserem Betriebssystem häuslich eingerichtet und einen scharfen Blick auf seine Umgebung und einige dort vorkommende Programme, Bibliotheken und andere Resourcen geworfen hat, startet als nächstes die ELisp-Maschinerie, und schaut sich ihrerseits im System um. Sie findet all die Sachen, die sie verwenden könnte, wenn es drauf ankommt und lädt die grundlegenden Funktionen ein. Sozusagen die *nicht ganz so grundlegenden Funktionen*, oder auch die *grundlegenden Funktionen 2*.

Der erste Zugriffspunkt für uns ist die Datei `XEM/site-packages/lisp/site-start.el`. [2]. Dort wird ein Systemadministrator all die Einstellungen vornehmen, die er gerne allen Anwendern auf dem System als Voreinstellung anbieten möchte.

[1] OK, ihresgleichen ist leicht gefunden: der GNU Emacs, der Vater und Bruder des XEmcas zugleich ist. Im hier vorliegenden Schriftstück wenden wir uns jedoch dem XEmacs zu, wenngleich Vieles in gleicher oder ähnlicher Form auch auf den GNU Emacs angewandt werden kann. Die Trennung der beiden Emacsen bildet in der Geschichte eine der unschönen und nicht positiven Trennungen im Bereich der freien Software. Vielleicht sogar die bekannteste dieser blöden Trennungen, die vermutlich positivste und damit das Gegenstück dürfte die Zwischenzeitliche Trennung und spätere Wiedervereinigung mit deutlichen Vorteilen des GCC sein. Aber das hier ist ja kein Buch über die Geschichte der Freien Software.

[2] Ich verwende hier `XEM` als Abkürzung für das Basis-Verzeichnis, in dem der XEmacs installiert wurde. Üblicherweise wird das sowas sein wie `/usr/share/xemacs/21.4.4`.

Wer diese Datei nun ändert, wird sich vermutlich beim nächsten Systemstart wundern, dass die Änderungen gar nicht aufgetaucht sind. Dass liegt vermutlich daran, dass es eine Datei fast gleichen Namens, nur mit der Endung `.elc` gibt. Eine solche Datei ist eine kompilierte Lisp-Datei, die einfach etwas schneller geladen werden kann. Wann immer eine solche Datei vorliegt, wird XEmacs sie beim Laden vorziehen. Wie solche erstellt werden, beschreibt Abschnitt 8.5.

Im darauffolgenden Schritt der Initialisierung kommt der große Moment, auf den der versierte XEmacs-Anwender gewartet hat: die Benutzer-eigene Initialisierungs-Datei `$HOME/.xemacs/init.el` wird geladen![3]

Dieses ist der wohl wichtigste Punkt beim Starten des XEmacs, denn ab dieser Stelle kann jeder Anwender seine eigenen Definitionen vornehmen. Wie das genau geht, erklärt Abschnitt 4.

In einer weiteren Phase der Anwender-spezifischen Einrichtung wird das Customize-System mit seinen Einstellungen geladen (siehe Abschnitt 4.1).

Abschließend wird noch eine weitere System-Datei geladen, mit der der Administrator nach belieben einige Einstellungen vornehmen kann, die nach dem Laden der Anwender-spezifischen Dateien in Kraft treten: `XEM/lisp/default.el` Es solle sich jedoch kein Administrator der irrigen Annahme hingeben, auf diesem Wege Einstellungen erzwingen zu können, da jeder User das Laden dieser Datei unterdrücken kann.

Und kurz danach steht der XEmacs dem Anwender voll zur Verfügung. Es ist Voreinstellung, dass jetzt zunächst ein Hilfe-Fenster mit den wichtigsten Details eingeblendet wird, das beim ersten Tastendruck verschwindet und durch den `*scratch*`-Buffer ersetzt wird.

Dieser Buffer ist jedoch nicht als echtes Text-Dokument gedacht (wie ja auch ein kleiner Hinweis ganz oben erläutert), sondern für kurze Notizen und zum Evaluieren von Lisp-Code.

2.2. C-x C-c

Die vielleicht wichtigste Aktion nach dem Starten ist das Beenden des Programms. Da der Emacs entworfen und programmiert wurde, als es heute gängige Programme und Systeme noch lange nicht gab, hält er sich schlicht und ergreifend nicht an die Schemata, die durch solche gängigen Programme eingeführt und verbreitet wurden. Und beim Versuch, das Programm zu beenden, kann man dieses auch gleich spüren: das verbreitete (C-g), also das Drücken der (Control) oder auch (Strg)-Taste zusammen mit (g) bewirkt jedenfalls nicht das gewünschte. Stattdessen erzielt man das Programmende mit (C-x) (C-c), oder etwas ausführlicher: (M-x) `kill-emacs`.

So ganz im Vorbeigehen haben wir dabei auch gleich die typische Schreibweise für XEmacs-Tastatur-Sequenzen kennengelernt. Dabei steht ein (C-) immer für das gleichzeitige Drücken der Steuerungstaste und ein (M-) für die Meta-Taste. Nun habe die wenigsten Tastaturen eine echte Meta-Taste eingebaut[4], und dort wird üblicherweise die (Alt)-Taste diese Funktion übernehmen. Ist das einmal nicht der Fall, kann man sich immer noch durch Drücken von (Esc) und drauf folgendem Drücken der Folgetasten helfen.

2.3. Buffer, Windows, Frames

Wie bereits erwähnt, gilt für XEmacs vieles, was heute Standard ist, *nicht*. So ist ein XEmacs-*Window* nicht das, was man sich gemeinhin unter einem „Window“ vorstellt. Daher wollen wir uns kurz einen Überblick über die in diesem Bereich gängige Nomenklatur verschaffen und en passant auch gleich noch einige wichtige Tastenkürzel dabei kennenlernen.

Zur Verdeutlichung kann die Abbildung 2.1 herangezogen werden, die einige Elemente beschreibt.

Frame Ein Frame ist ein XEmacs-Fenster, also das, was woanders meist Fenster/Window heißt. Zu einem einzelnen XEmacs-Prozess können viele Frames gehören, die sich mit (C-x) (5) (2) erstellen und mit (C-x) (5) (0) wieder schließen lassen. Ein Frame hat keinerlei Einfluss darauf, welche Dateien im XEmacs gerade geöffnet sind, aber ein Frame kann eine gruppierende Wirkung haben. Auch gelten manche Einstellungen immer Frame-spezifisch,

[3] Eigentlich ist es ja schade, dass es nicht mehr die sagenumwobene `$HOME/.emacs` ist, weil damit solche Sprüche wie “HOME is where .emacs is” zumindest für XEmacser nicht mehr angebracht sind. Da aber in der Vergangenheit die `Dotemacs` doch oftmals nur noch zum Unterscheiden der beiden großen Emacs-Versionen und entsprechendem Nachladen separater Initialisierungsdateien diente, ist der Schritt nur konsequent.

[4] Ich kenne die eigentlich nur von Rechnern von SUN

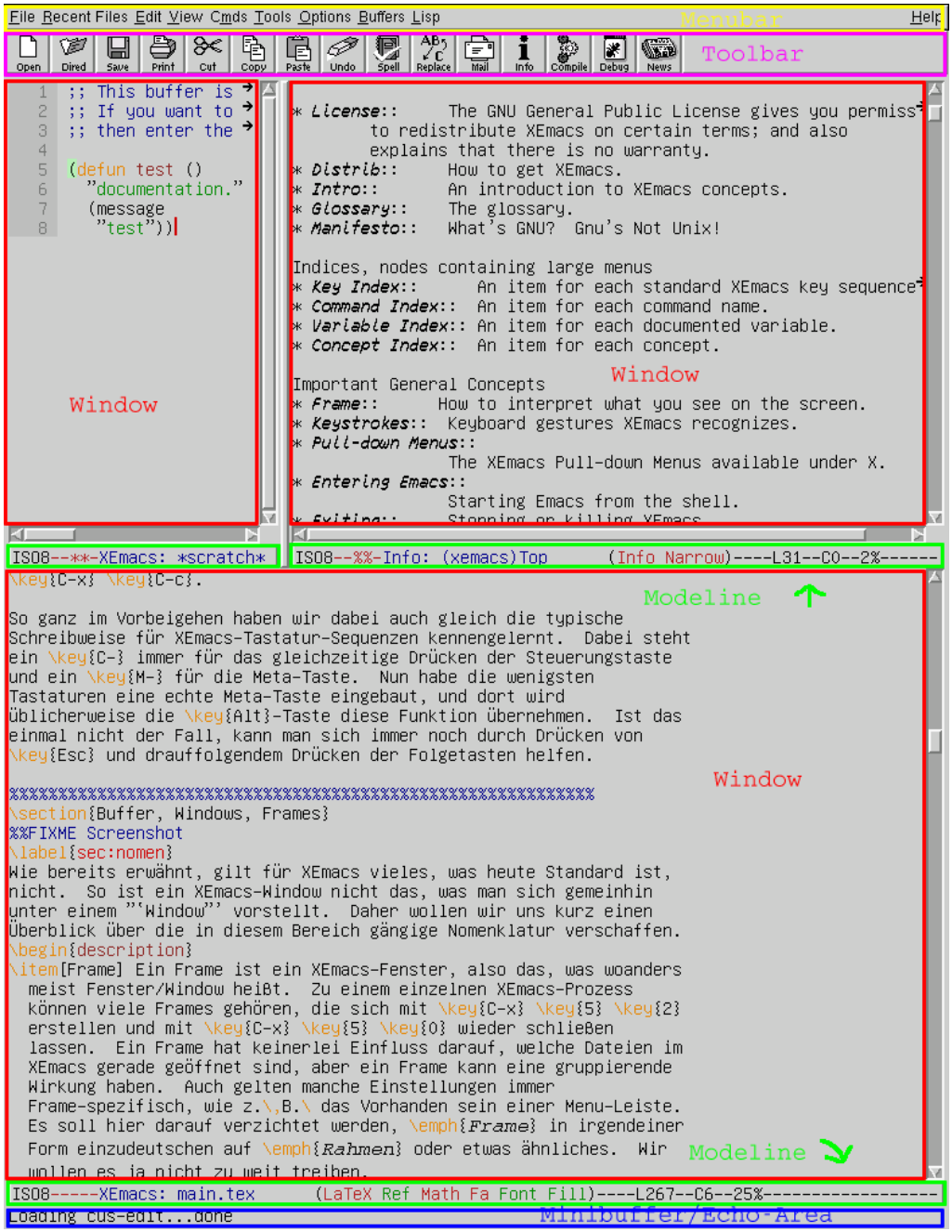


Abbildung 2.1.: Ein XEmacs-Frame (das ganze Fenster) mit einigen markierten Bereichen: *Gelb:* die Menubar, *lila:* Toolbar, *rot:* drei Windows, teilweise durch horizontale, teilweise durch vertikale Teilung entstanden, *grün* die zu den Windows gehörenden Modelines sowie *blau:* die im Moment als Echo-Area agierende letzte Zeile (nur eine Ausgabe), die aber auch als Minibuffer zu Eingaben auffordert.

wie z. B. das Vorhanden sein einer Menu-Leiste. Es soll hier darauf verzichtet werden, *Frame* in irgendeiner Form einzudeutschen auf *Rahmen* oder etwas ähnliches. Wir wollen es ja nicht zu weit treiben.

Window Wenn ein Frame ein Window ist, was ist dann ein Window? Im XEmacsjargon bezeichnet ein Window gewissermaßen ein Unterfenster. Jeder Frame kann mit der Eingabe (C-x) (2) bzw. (C-x) (3) horizontal oder vertikal gesplittet werden. Die so entstandenen einzelnen Bereiche sind zwei (oder mehr) Windows. Bei manchen Gelegenheiten entstehen solche Windows ganz automatisch, z. B. wenn eine Vervollständigung aufpöppt[5], oder wenn eine Hilfe angezeigt wird. Es ist gut, zu wissen, dass man das Fenster, in dem man sich gerade befindet mit (C-x) (0) wieder los wird, oder aber mit (C-x) (1) zum einzig abendfüllenden Window macht.

Buffer Ein Buffer nun bezeichnet das Konzept, mit dem der XEmacs seine Daten präsentiert. In den allermeisten Fällen wird ein Buffer eine Datei enthalten, und auch entsprechend benamt sein. Es gibt aber durchaus auch andere Buffer, wie die Buffer, die in den bereits erwähnten Vervollständigungswindows angezeigt werden. Jeder Text, jeder Prozess, jedes Wasauchimmer erhält einen eigenen Buffer. Der Buffer ist das zentrale Konzept im XEmacs. Für Programmierer geht das sogar soweit, dass es manche Funktion gibt, Text in einem Buffer zu bearbeiten, die man für Text in reiner String-Form nicht hat.

Zugriff auf die Buffer-Liste liefern unter anderem das Buffer-Menu in der Menu-Leiste, aber auch die Eingabe (C-x) (C-b)

Menu-Leiste Nicht ganz überraschend weil überraschend standardkonform ist die Menu-Leiste am oberen Rand des Frames. Sie verändert sich oft in Abhängigkeit vom Inhalt des aktuellen Buffers und kann spezifische Menus enthalten. Wer gerade mal viel Zeit hat, kann hier einfach durchstöbern und sich inspirieren lassen, was es nicht alles so gibt. Freundlicherweise werden Tastenkürzel –so definiert– mit angezeigt. Ein Fass ohne Boden ist das Customizemenu.

Toolbar Auch im XEmacs findet sich eine Leiste mit grafischen Knöpfchen, die den schnellen Zugriff auf einzelne Funktionen liefern soll. Für die meisten langjährigen Benutzer ist die wichtigste Eigenschaft der Toolbar, dass man sie auch loswerden kann ;-)

Andererseits verwenden Applikationen wie VM oder Gnus die Toolbar ganz gezielt, und da kann es durchaus angenehm sein, sie vorzufinden.

Im Vergleich zu anderen Programmen, die eher GUI-orientiert sind, wirkt die Toolbar auch wirklich ein wenig altbacken. Es gilt ja heutzutage[6] bereits als unfein und hemmungslos überholt, wenn die Toolbarelemente noch 3D-Effekte aufweisen.

Modeline Das ist nun wieder eine Eigenart des XEmacs. Am unteren Rand jedes Windows findet sich ein einzeiliges Display, das uns mit einer Übersicht über aktuelle Einstellungen und Eigenschaften des angezeigten Buffers versorgt. Hier finden sich nicht nur der Name des Buffers und Informationen über den Änderungszustand oder das Codier-Schema, auch die Zeilen- und Spaltennummern und eine Liste der aktivierten Modi werden angezeigt[7].

Ziehen mit der Maus kann ein Window ößern oder verkleinern.

Minibuffer Die allerletzte Zeile, ganz am unteren Rand, wenn man gerade eine Eingabe machen soll. Jeglicher Prompt, sofern der Befehl mit einer Tastatur-Eingabe bewirkt wurde, wird dort erledigt. Beispielsweise, wenn man (C-x) (C-f) verwendet, um eine Datei zum Bearbeiten zu öffnen. Ruft man jedoch via Maus einen Dialog zum Dateiöffnen auf, wird auch ein Maus-Fenster verwendet. Ebenfalls im Minibuffer stehen auch die bisher gedrückten Tasten (nach einer kurzen Verzögerung), wenn man eine längere Tastatur-Sequenz eingibt.

Nach jahrelanger Erfahrung wundern sich viele Leute, dass ein so einfaches Interface, nur eine Zeile, so mächtig sein, so gut funktionieren kann.

[5] Ich finde „aufpöppen“ ist eine prima Übersetzung des englischen *pop-up*.

[6] Im Jahre 2004

[7] Es sollte an dieser Stelle wohl erwähnt werden, dass die Zeilen- und Spaltennummern kein Standard sind, man muss sie erst mit (M-x) `line-number-mode` bzw. (M-x) `column-number-mode` einschalten. Aber da passt die Information doch gut hin, oder?

Echo Area Ebenfalls die allerletzte Zeile, aber dann wenn XEmacs gerade eine Ausgabe macht.

Scrollbar Hui, eine Scrollbar ist aber ganz schön normal. Zeigte sich noch die Scrollbar des GNU Emacs wenigstens vollkommen ungewohnt, wir die Scrollbar des XEmacs wenig Überraschungen bereithalten. Eine wichtige Eigenschaft aber gibt es schon: In der Scrollbar vieler X Programme (Motif, GTK+, KDE, teilweise einstellbar) findet sich ein Verhalten, das man bei MS Win Anwendungen vergeblich sucht: Wenn man mit der mittleren Maustaste auf eine freie Stelle der Scrollbar drückt, springt man direkt dorthin, ohne erst von ganz vorne bis dorthin scrollen zu müssen.

Region Die Region ist der aktuell markierte Text. Textmarkieren kann, ganz wie gewohnt, mit der Maus vonstatten gehen. Einfach mit gedrückter linker Maustaste über den Text ziehen. Erwähnenswert ist da höchstens noch, dass DoppekKlicks ganze Worte anwählen (bei Weiterziehen auch in ganzen Wort-Schritten weitergehen) und Dreifachklicks ganze Zeilen. Soweit nix neues. Etwas ungewohnt ist es da schon, wenn man mal die Shift/Umschalt-Taste benutzt: ist eine Region bereits markiert, kann man mit Shift-Mausziehen den Bereich sowohl vorne als auch hinten noch verändern[8].

Das bekannte Verhalten, Text mit Shift und den Cursor-Tasten zu markieren erhält man nur durch Verwenden des CUA-Moduls. Der XEmacsstandard lohnt aber auf jeden Fall einen Blick, auch wenn's zunächst ungewohnt ist (wie so vieles im XEmacs). Mit (C-SPACE) sagt man dem XEmacs einfach, dass es jetzt losgeht mit dem Markieren. Jede weitere Cursorbewegung, zu der man also auch all die tollen Funktionen verwenden kann, die sich irgendwie in größeren Schritten bewegen (`forward-word`, `forward-sexp` `end-of-buffer`, etc.), markieren ab nun.

Will die Anwenderin die aktuelle Region wieder loswerden, hilft XEmacs' Escape: (C-g).

Point und Mark Genaugenommen setzt der Befehl (C-SPACE) das *Mark* auf die aktuelle Position.

XEmacs verwaltet für jeden Buffer zwei besondere Stellen: Point und Mark. *Point* ist immer die Stelle unter dem aktuellen Cursor. *Mark* ist so etwas ähnliches, nämlich auch eine Stelle, die genauso gut ein Point sein könnte, aber üblicherweise verschieden vom Point. Die Region dazwischen, ist halt die Region, und die ist meistens markiert. Will man einfach Point und Mark vertauschen, also vom Ende einer Markierung zum Anfang gehen (aber es gibt auch noch andere Fälle, in denen dies sinnvoll ist, z. B. wenn man eine Suche gemacht und abgebrochen hat, kann man damit zum Ausgangspunkt zurück gehen), verwendet man (C-x) (C-x).

Eine wichtige Eigenschaft des Marks (und der *Marker*, ein Datentyp, der weitere Marks ermöglicht) ist, dass sie mitwandern, wenn weiter vorne Text verändert wird. Es ist also keineswegs eine feste Byte-Position sondern eine kontextbezogene Position.

Kill und Yank Hierbei handelt es sich keineswegs um eine unfeine Attacke gegen einen Amerikaner, sondern um die XEmacs-Beschreibung für „Ausschneiden und Einfügen", auch „Cut 'n' Paste" genannt. „Copy" (Kopieren) bleibt übrigens auch im XEmacs „Copy". Die Tasten dazu? (C-w) für kill, (M-w) für copy und (C-y) für yank.

Modes Jeder Buffer befindet sich in genau einem *Major-Mode*[9]

Dieser Mode bestimmt die grundlegende Funktionalität des Buffers und entspricht in den meisten Fällen in eleganter Form dem Inhalt des Buffers. So wird eine Ruby-Datei im `ruby-mode` sein, eine C++-Datei eher im `c++-mode` und eine Perl-Datei im `cperl-mode`. Huh? Wieso denn *c*perl? Naja, der XEmacs ist schon nicht mehr so ganz jung und es wurde bereits einiges an Geschichte geschrieben. So gab es dereinst den `perl-mode`, der jedoch vom `cperl-mode` abgelöst wurde, der von der Geschichte einfach bevorzugt behandelt wurde. Ähnliches gilt für den `LaTeX-mode`, der aus dem AUCTeX-Paket stammt und sehr, sehr leistungsfähig ist, der aber den älteren `latex-mode` abgelöst hat. Major-Modes gibt es für so ziemlich alle gängigen[10] Programmiersprachen, aber auch für Shells oder zum Maillesen und -schreiben.

[8] können das die ganzen modernen GUI-Programme eigentlich auch?
[9] Es sei denn man verwendet das mmm-Module, das *M*ultiple-*M*ajor-*M*ode-Modul.
[10] sowie für weitere nicht so gängige

Zu einem solchen Mode gehören immer besondere Tastatursequenzen. Es ist üblich, dass die Major-Mode--Tastatursequenzen immer mit dem Präfix C-c beginnen. Da findet man dann häufig Befehle zum Starten wichtiger Programme (z. B. C-c C-c im `LaTeX-mode`), oder zum Senden eines Textes an einen Prozess (`sql-mode`, `gnuplot-mode`), zum Kommentieren der aktuellen Region oder zum Einfügen von Text-Bausteinen (C-c C-e im `LaTeX-mode`, C-c C-c im `xml-mode`, C-z 1 im `html-mode`[11]). Einen Überblick über den aktuellen Major-Mode sowie die aktuellen Keybindings, also die aktuell definierten und verfügbaren Tastatursequenzen, erhält man mit C-h m.

Zudem kann ein Major-Mode noch eigene Menus in der Menuleiste, einen ganzen Haufen eigener Funktionen oder auch ein Kontextmenu auf der rechten Maustaste (Ja, soweit sind hier auch weit verbreitete Standards eingeflossen ;) bereitstellen, oder auch die Toolbar verändern und überhaupt ganz viele tolle Sachen machen.

Die ganz grundlegenden Major-Modes sind der `fundamental-mode` und der `text-mode`.

Zusätzliche Funktionalität, die vielleicht auch in verschiedenen Major-Modes Sinn macht, weil sie nicht auf eine Programmiersprache begrenzt ist, wie beispielsweise das automatische Umbrechen langer Zeilen nach dem Erreichen einer bestimmten Spalte (`auto-fill-mode`) oder aber das syntaktische Einfärben (`font-lock-mode`) wird von sogenannten *Minor-Modes* bereitgestellt, und es können jederzeit mehrere davon aktiv sein. Ein Liste von möglichen Minor-Modes, gibt ein Rechtermausklick auf die Modeline, in genau dem Bereich, in dem die Modes angezeigt werden: etwa mittig, in runde Klammern eingefasst.

Auch die Minor-Modes werden in der Mode-Hilfe, hinter C-h m erwähnt.

2.4. (defun () “und andere Klammern”)

Eigentlich will der Leser ja am liebsten alles schon ausprobieren, aber dennoch soll an dieser Stelle ein wenig die Programmiersprache LISP erklärt werden. HALT! Nicht gleich weglaufen. Zwei Versprechen:

1. Diese LISP-Einleitung wird echt kurz.
2. Danach geht es mit einem interaktiven Tutorial in medias res.

Gut. Die Grundlage von Lisp wird von lediglich vier einfachsten Regeln gegeben:

1. *Listen* werden in runde Klammern eingeschlossen.
2. Der erste Eintrag einer solchen Liste ist ein Funktionsname, alle weiteren sind Argumente zu dieser Funktion.
3. Wenn vor einer runden Klammer ein Hochkomma steht werden die Einträge der Liste nicht evaluiert, d. h. der erste Eintrag nicht als Funktion und die weiteren nicht als Variablennamen betrachtet.
4. Achja: Variablennamen evaluieren zum Inhalt der Variable.

So, mit diesem Grundwissen, kann man nun schon sehr viel von dem Lispcode, dem man so begegnen kann, einigermaßen lesen. Zu beachten sind dann im wesentlichen nur noch die *Special Forms*, die daran zu erkennen sind, dass die vom XEmacs anders eingefärbt werden. Zu diesen gehören `if`, `defun` oder auch `while`. Für diese Special Forms gelten etwas andere Regeln als für Funktionsaufrufe, die hier aber zunächst nicht interessieren sollen.

2.5. Tutorial

Dieses Kapitel wird in dieser Anleitung sehr kurz ausfallen, andererseits aber dem Leser eine Menge beibringen. Der Grund für dieses merkwürdige Verhalten dieses Kapitels ist, dass andere sich schon die Mühe gemacht haben, es zu schreiben[12].

[11] OK, ok, große Ausnahme hier. Der HTML-Mode selber verwendet auf C-c die Tastaturkürzel von PSGML und belegt das selten benutzte C-z mit Hilfe des `hm-html-mode` mit vielen Tasten zur Eingabe von HTML-Tags

[12] Genaugenommen war auch schon dieses Geschreibsel nur Füllstoff

Im, und das meint durchaus wirklich *im*, XEmacs gibt es bereits ein interaktives Tutorial, das die Anwenderin gemächlich an verschiedene Konzepte heranführt und die grundlegenden Funktionen des XEmacs erläutert.

Vorneweg nur noch einige kurze Anmerkungen, dann kann es losgehen.

- Keine Angst, man gewöhnt sich schnell daran!
- Das Tutorial bezieht sich auf den GNU Emacs, so dass sich einige Unterschiede ergeben.
- Die Scrollbarbeschreibung bezieht sich auf die ungewöhnliche aber sehr durchdachte Scrollbar des GNU Emacs, die hier nicht weiter betrachtet werden soll, nicht auf die des XEmacs.
- Bei allen Dingen, die mit der Oberfläche und dem Aussehen des Emacs zu tun haben (Menus, Maus, etc) wird das Tutorial ebenfalls häufig vom XEmacs abweichen.
- Natürlich funktionieren auch die Cursor-Tasten und die Bildlauftasten.
- Vielleicht gibt es aber auch jetzt schon eine angepasste Version des Tutorials?

So, jetzt aber los. Tutorial starten. Wie? Einfach im Menu unter Hilfe im Untermenu Tutorial die gewünschte Sprache aussuchen. Oder aber (M-x) `help-with-tutorial` oder aber (C-u) (M-x) `help-with-tutorial` (für die Sprachauswahl).

Achso: und eines nicht vergessen: Kein Mensch muss wirklich (M-x) (h) (e) (l) (p) (-) (w) (i) (t) (h) (-) (t) (u) (t) (o) (r) (i) (a) (l) tippen, es gibt da TAB-Vervollständigung:
(M-x) (h) (e) (l) (p) (-) (w) (TAB), gleiches gilt für die Auswahl der Sprache[13].

2.6. (describe-...) – Hilfe zur Selbsthilfe

Zum Abschluss der Einleitung soll noch ein wenig Hilfe zur Selbsthilfe gegeben werden. Ein System wie den XEmacs kann man eigentlich gar nicht nur einfach anwenden. Man lebt darin und lernt im Laufe der Zeit immer mehr Winkel und Kammern kennen, in denen Nützliches steckt. Bei der Suche nach all diesen Ecken steht die Anwenderin aber nicht ohne Hilfe da.

Als erstes zu nennen ist dann die Vervollständigung mit der (TAB)-Taste. Wer einfach nur mal eine Liste aller aktuell verfügbaren Funktionen sehen möchte, tippt nur kurz (M-x) TAB (TAB). Es öffnet sich ein Buffer, mit viiielen Zeilen.

Ist dann eine Funktion bekannt und das Keybinding wird gesucht, so hilft (C-h) (w) gefolgt von dem Funktionsnamen weiter. Umgekehrt erzählt uns (C-h) (k) gefolgt von einer Tastatursequenz, welche Funktion sich dahinter verbirgt. Die Sequenz (C-h) (m), haben wir schon besprochen, sie beschreibt die aktuellen Major- und Minormodes.

Eine unglaubliche Stütze kann `apropos` sein. Einfach mal (C-h) (a) eingeben und nach irgendeinem Text suchen, der interessant sein könnte. Z. B. „buffer“. Es folgt ein Buffer (mit eigenem Major-Mode ;-), der alle Funktionen und Variablen anzeigt, in deren Namen „buffer“ vorkommt. Die apropos-Suche funktioniert aber auch für die *Inhalte* von Variablen oder sogar für die Dokumentation von Funktionen und Variablen: einfach mal (M-x) `apropos-` (TAB) (TAB) eingeben ....

Schlussendlich findet man mit (C-h) (i) zum Info-Stand, und dort findet man nicht nur die nahezu komplette Beschreibung von XEmacs und ELisp (und GNU Emacs und eine Lisp-Einleitung) sondern auch die Beschreibung zu vielen, vielen anderen Programmen.

Diese Einleitung kann und will gar nicht alles erzählen, das wäre vermessen. Es soll immer versucht werden, wichtige (Grundeinstellungen) oder interessante Aspekte eines Themas (nette Funktionen, kleine Tricks) herauszupicken. An dieser Stelle sollte auch erwähnt werden, dass diese Dokumentation zwar in Deutsch erstellt wurde, man jedoch als XEmacsanwender nur mit der deutschen Sprache nicht allzuweit kommen wird. Die meisten XEmacspakete und -majormodes kommen mit teils sehr ausführlichen Dokumentationen (zu finden im Infosystem) auf englisch, die Webseiten sind englisch, die angesagte Newsgroup `comp.emacs.xemacs` ist englischsprachig. Diese Dokumentation möchte nur

[13] So, jetzt ist aber genug gefüllt hier!

- den Einstieg erleichtern,
- nette Trickse zeigen und
- angenehm, vielleicht sogar spaßig zu lesen sein.

Bei der Selbsthilfe kommt es immer wieder darauf an, all die möglichen Resourcen aufzusuchen. Diese sind, zusammenfassend noch einmal gesammelt:

- Das Infosystem, auf der Kommandozeile mit `info`, im XEmacs mit C-h i,
- die Newsgroup `comp.emacs.xemacs`,
- das Apropossystem, zu erreichen mit C-h a, M-x `apropos`, M-x `hyper-apropos`,
- die eingebaute Hilfe zu Funktionen und Variablen mit C-h f und C-h v,
- die TAB-Vervollständigung bei der Eingabe von Funktionen sowie
- `grep -r` im XEmacsinstallationsverzeichnis (auch nicht zu unterschätzen).

3. Anwendungen und Major-Modes

In diesem Kapitel wollen wir uns möglichst vielen mehr oder minder bekannten Anwendungen zuwenden. Es erfolgt dabei ein kleiner Vorgriff auf das Kapitel 4, in dem erst erläutert wird, wie man denn den angegebenen Konfigurationscode ein sein System geben soll. Andererseits haben wir ja schon eingangs in Kapitel 2.1 gelernt, dass es sich dabei um verschiedene Dateien handelt. Also nicht verwirren lassen, einfach überlesen, was nicht verstanden werden kann und ggf. später wieder zurück kommen.

3.1. AUCTeX

Einer der wichtigsten Befehle, um mit dem XEmacs komfortabel LaTeX-Texte zu tippen, lautet:

```
(require 'tex-site)
```

Durch diesen Code schaltet man den XEmacs vom alten und oftmals noch immer ebenso standard- wie -mäßigen `latex-mode` auf die Verwendung des AUCTeX-Paketes zur Bearbeitung von LaTeX-Dateien um. Und das machst sich bemerkbar. Zunächst einmal durch neue Menueinträge, die einerseits den Start diverser Programme (wie LaTeX, BiBTeX, aber auch `xdvi` und `dvips`) ermöglichen (Menu: „Command“), andererseits durch das Menu „LaTeX“, in dem die Eingabe von Umgebungen, Schriftarten und Makros sowie die Navigation und andere nette Kleinigkeiten angeboten werden (siehe dazu auch Abbildung 3.1).

Ersteres braucht man eigentlich nur selten. Denn mit der Tastenkombinaten C-c C-c ist eigentlich alles gesagt und getan. XEmacs geht in folgenden Schritten vor:

1. Nachschauen, ob alle Dateien gespeichert sind und ggf. nachfragen.
2. Erstmal LaTeX aufrufen.
3. Wenn LaTeX noch nach mehr fragt, wie z. B. einem zweiten Lauf, weil die Querverweise noch nicht vollständig angelegt sind, oder aber nach BiBTeX, weil Zitate fehlen, werden diese Programme aufgerufen.
4. Ist das Dokument soweit vollständig, bietet XEmacs den Befehl *View* an, mit dem `xdvi` aufgerufen wird.

Will man die Ausgabe von LaTeX mitverfolgen, bietet es sich an, dem Hinweis in der Echo-Area Folge zu leisten, und C-c C-l noch hinterher zu schicken. Ist immer noch schneller als zu einer Konsole zu wechseln und dort die Befehle einzugeben oder aus der Historie zu holen.

Bei Auftreten eines Fehlers springt XEmacs nach Drücken von C-c ` (Backtick) direkt an die fehlerhafte Stelle. Allerdings sei erwähnt, dass diese Stelle nicht immer die richtige sein muss, weil LaTeX selber manches Mal ganz schön in die Irre geführt wird.

3.1.1. RefTex

Zum Verwalten von Zitaten und Referenzen (`\cite{}`) und `\ref{}`) bietet sich das RefTeX Paket ([4]) an. Es wird mittels

```
(turn-on-reftex)
```

Abbildung 3.1.: AUCTeX in Aktion. Die Ausgabe von LaTeX ist ebenso dargestellt wie ein Auswahlfenster von RefTeX. Im Menu wurde gerade überlegt, ein Umgebung einzufügen.

in der Konfiguration aktiviert und die wichtigsten Tastenkombis sind einerseits C-c) für einen Querverweis und C-c] für ein Zitat aus einer Literaturdatenbank. Diese muss RefTeX bekannt sein, was man leicht durch Setzen der Variablen `reftex-default-bibliography` (am einfachsten im Customize Menu: customize-> emacs-> wp-> TeX-> RefTex->Citation Support) auf die entsprechende .bib-Datei erledigt.

Beim Zitieren fragt RefTeX nach einem Suchmuster für die Datenbank und bietet alle gefundenen Einträge in einem neuen Buffer zum Anwählen mittels Cursortasten an. Es gibt in diesem Buffer auch noch weitere Navigationsmöglichkeiten, die man sich mittels Beschreibung des RefTeX-Modes beschaffe. Bei dem Erstellen von Querverweisen ist RefTeX ziemlich intelligent: Wenn man hinter dem Wort *Gleichung* C-c) drückt, weiß es, dass es um eine Gleichung geht und bietet folgerichtig nur alle Gleichungen zur Auswahl an. Dabei gilt es noch einiges zu beachten: lässt man kein Leerzeichen nach *Gleichung*, erstellt RefTeX ein (vor Umbruch) geschütztes Leerzeichen (~), lässt man eines, bleibt alles so. Damit das funktioniert, muss man alle seine Gleichung mit `\label{eq:...}` bezeichnen. Entsprechend bezeichnet man Abbildungen mit `\label{fig:...}` usw... Kann RefTeX nicht erkennen, worum es sich bei dem Querverweis handelt, wird zunächst danach gefragt. Die gefundenen Label stammen natürlich aus allen zum Dokument gehörenden Files (vgl. Abschnitt 3.1.2).

3.1.2. LaTeX-Projekte

Viele LaTeX-Projekte werden groß. Da liegt es dann nahe, den Inhalt auf mehrere Dateien zu verteilen, die von einem zentralen Masterdokument mit ins Boot geholt werden.

Ein Beispiel für eine solche Masterdatei könnte in Kurzform so aussehen:

```
\documentclass[10pt,a4paper,DIV15,titlepage,headsepline,footsepline,bibtotoc,idxtotoc]{scrbook}
% ... rest der Praeambel
\begin{document}
\include{einleitung}
\include{einkapitel}
% ... weitere Dateien
\end{document}
% Make this the Master File for Emacs-AUC-TeX:
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
```

Auffällig daran sind die letzten Zeilen. In der einzelnen Zeile `% Master` wird festgelegt, dass diese Datei als Zentraldokument zu betrachten ist, auf das sich diverse Aufrufe wie der Start von `latex` beziehen.

Alle anderen Dateien dieses Projektes, die mit dem `include`-Befehl eingebunden werden, sollten folgende Zeilen am Ende haben.

```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: "main"
%%% End:
```

Mit diesen *local Variables*, oder auch *Filevariablen*, (siehe Abschnitt 7.1) wird XEmacs angewiesen, die angegebene Datei als Master zu verwenden.

Damit sollte es im XEmacs transparent sein, ob ein LaTeX-Projekt aus nur einer großen oder vielen kleinen Dateien besteht. Man sollte allerdings, wie mir gut unterrrichtete Quellen aus dem LaTeX-Lager berichtet haben, nicht unbedingt arg viele `\input` Befehle verwenden, weil LaTeX selber dann irgendwann Stress macht.

3.1.3. Mathe DeLuxe

Wer im `TeX-mode-hook` auch noch die Zeile

```
(LaTeX-math-mode)
```

unterbringt, wird mit einem weiteren Menu, das ganz, ganz viele Einträge für mathematische Symbole enthält, belohnt. Zudem gibt es eine mathematische Präfix-Taste (Standard: Backtick), die die verschiedenen Tastaturkürzel zur Eingabe der Symbole einleitet. An dieses Präfix kann man auch noch weitere, eigene Ergänzungen anhängen:

```
(define-key LaTeX-math-keymap
  (concat LaTeX-math-abbrev-prefix "/")  'LaTeX-math-frac))
```

3.1.4. Preview LaTeX

Hier ist Platz für all die GUI-Maniacs, die das im Titel dieses Abschnitts genannte Paket einsetzen, um bereits im XEmacs selber eine Idee von der späteren visuellen Gestaltung zu erhalten.

3.1.5. Whizzytex

Auch bei diesem Paket kann ich nur auf die Mithilfe begeisterter Anwender hoffen und bleibe bis dahin (oder bis ich es selber benutze) weitere Erläuterungen schuldig.

3.1.6. Konfiguration

Die hier vorgestellte Konfiguration kann so u. U. nur für LaTeXverwendet werden, weil ihr jeglicher Praxiseinsatz mit anderen Systemen, die von AUCTeX ebenfalls abgedeckt werden, fehlt.

```
(require 'tex-site)
(setq TeX-parse-self t) ; Enable parse on load.
(setq TeX-auto-save t) ; Enable parse on save.

(add-hook 'TeX-mode-hook
          '(lambda ()
             (setq ispell-parser 'tex)
             ))

(add-hook 'LaTeX-mode-hook
          '(lambda ()
             ;; meine keybindings (braucht Extrafunktionen)
             ;;(ska-coding-keys TeX-mode-map)
             ;;(ska-latex-mode-keys)
             (LaTeX-math-mode)
             (turn-on-reftex)
             (fume-add-menubar-entry)
             (setq fume-display-in-modeline-p nil)
             ))
```

Besondere Erwähnung verdienen hier noch `TeX-parse-self`, das selber geschriebene Kommandos erkennt und bei der Eingabe anbietet, sowei `TeX-auto-save`, das das Abspeichern solcher Informationen zum schnelleren späteren Einbinden erlaubt.

3.1.7. Beispiel

Abschließend soll in diesem Kapitel ein Beispiel für ein Masterdokument gegeben werden, das bereits viel Erfahrung mit diversen Aspekten von LaTeX beinhaltet. Es kann als Anregung verstanden werden, wie man eigene Dokumente gestalten könnte.

```
\documentclass[10pt,a4paper,DIV15,titlepage,headsepline,footsepline,bibtotoc,idxtotoc]{scrbook}
%% PLEASE NOTE, that the following lines to detect whether we are
%% running as pdflatex or latex are error prone!  I only use them
%% because my LaTeX-installation does not contain ifpdf.sty!
%% BITTE BEACHTET, das die folgenden Zeilen anfaellig fuer Fehler
%% sind.  Die aktuelle und richtige Loesung waere die Verwendung des
```

```
%% Pakets ifpdf.sty, das mir jedoch derzeit nicht zur Verfuegung
%% steht.
\newif\ifpdf \ifx\pdfoutput\undefined
\pdffalse % we are not running pdflatex
\else
\pdfoutput=1 % we are running pdflatex
\pdfcompresslevel=9 % compression level for text and image;
\pdftrue
\fi

\usepackage[latin1]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{textcomp}
\usepackage{ngerman}
\usepackage{fancyhdr}
\usepackage{makeidx}
\usepackage{verbatim}

\makeindex

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       VERBATIM
\makeatletter
\def\verbatim@font{\normalfont\ttfamily\footnotesize
                   \hyphenchar\font\m@ne
                   \let\do\do@noligs
                   \verbatim@nolig@list}
\makeatother
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       TEX4HT
%% LaTeX to HTML
%% http://www.cis.ohio-state.edu/~gurari/TeX4ht/
%% einkommentieren und mit 'htlatex ...tex' TeXen
%% \usepackage[html,sections+,3,png]{tex4ht}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       FONTS
\font\fc=cmss10   scaled 800
\font\ft=phvr8r   scaled 800


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     HEADER/FOOTER
\pagestyle{fancy}
% first define for printing style
\fancyhead[LO,RE]{\slshape \leftmark}
\fancyhead[LE,RO]{\slshape \rightmark}
\fancyhead[C]{}
\fancyfoot[LE,RO]{\bfseries \thepage}
\fancyfoot[LO,RE]{\small\slshape XEmacs auf Deutsch}
\fancyfoot[C]{}
\renewcommand{\headrulewidth}{0.4pt}
\renewcommand{\footrulewidth}{0.4pt}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 PDF or not to PDF
\ifpdf
\makeatletter
\@twosidefalse
\makeatother
% die naechsten drei zeilen sind die empfohlene ersetzung fuer
% times.sty
\usepackage{mathptmx}
\usepackage[scaled=.90]{helvet}
```

```
\usepackage{courier}
\usepackage{thumbpdf}
\usepackage[pdftex,xdvi]{graphicx}
\usepackage[pdftex,colorlinks=true,backref]{hyperref}
\usepackage[pdftex]{color}
\else
\usepackage{graphicx}
\usepackage[colorlinks=false,backref]{hyperref}
\fi

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   META INFORMATION
\newcommand{\version}{2.04}
\author{Stefan Kamphausen}
\title{XEmacs auf Deutsch}
\titlehead{%
  \begin{minipage}[b]{0.4\textwidth}
    Eine pragmaprogrammatische Einleitung mit vielen Beispielen und
    bunten Bildern in etwas mehr als einen Texteditor.
    \href{http://www.skamphausen.de/xemacs}
    {\nolinkurl{http://www.skamphausen.de/xemacs}}
  \end{minipage}
  }

\lowertitleback{%
  Copyright \copyright\ 2000-2004 Stefan Kamphausen\\[2ex]

  Gesetzt, getippt und gestaltet mit Freier Software (\LaTeX,
  XEmacs und Gimp).\\[2ex]

  Permission is granted to copy, distribute and or modify this
  document under the terms of the GNU Free Documentation
  License, Version 1.1 or any later version published by the
  Free Software Foundation; with no Invariant Sections,
  with no Front-Cover Texts, and with no Back-Cover
  Texts. A copy of the license can be obtained from \cite{fdl}.
}


\begin{document}
\maketitle

\tableofcontents

\include{datei1}
\include{datei2}
\include{datei3}
% ...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\nocite{*}
\renewcommand\refname{Links und andere Quellen}
\bibliography{quellen}
\bibliographystyle{mybibstyle}

\printindex

\end{document}
% Make this the Master File for Emacs-AUC-TeX:
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
```

3.2. psgml-mode – SGML und XML und so

SGML ist die Mutter aller Markupsprachen. Nagut, ich weiß es eigentlich gar nicht, ob sie wirklich die *Ur*mutter ist, aber SGML ist mal alt und erprobt. HTML und XML sind sicherlich ihre prominentesten Kinder, aber auch Docbook hat eine gewisse Popularität erlangt. Lange bevor XML-Schema am Horizont erschienen, wurden Dokumentenstrukturen mit DTDs, den *Document Type Definitions*, festgelegt. Diese legen fest, welches *Tag*, also ein in spitze Klammern eingefasster Strukturierungsbefehl, wann kommen darf. Wer also die DTD verstehen kann, kann sagen, ob ein entsprechendes Dokument *valide* ist, oder nicht.

Das XEmacspaket, das DTDs parsen kann und Dokumente vom Typ SGML bearbeiten hilft, heißt PSGML. Die Dokumentation und die Konfigurationsmöglichkeiten sind massiv.

PSGML kommt mit einem eigenen Syntaxhighlighting daher, das aber erst aktiv wird, wenn die DTD geparst wurde. Dazu aktiviert man mit

```
(setq sgml-auto-activate-dtd t)
```

das automatische Parsen der DTD beim Laden einer Datei. Ggf. muss man noch einmal (TAB) drücken, damit der Text eingefärbt wird.

Weitere Einstellungen entnehme die geneigte Leserin bitte meiner Beispielkonfiguration im Abschnitt 4.5.

3.2.1. Benutzung

Während man also seine XML- oder SGML-Dateien tippt, gibt es einige Funktionen, die man einfach immer wieder benötigt:

(M-x) **sgml-insert-end-tag**, (C-c) (/) Beendet das aktuelle Tag.

(M-x) **sgml-insert-element**, (C-c) (C-e) Fügt ein Tag mit TAB-Completion auf die an der aktuellen Position erlaubten Tags ein.

(M-x) **sgml-split-element**, (C-c) (C-RET) Fügt vom aktuellen Tag ein Ende und ein Anfangstag ein. Praktisch, um einen neuen Paragraphen zu beginnen.

(M-x) **sgml-next-trouble-spot**, (C-c) (C-o) Springt die nächste Position an, an der PSGML eine Unstimmigkeit mit der DTD findet.

Ich persönlich habe immer mal wieder Probleme damit, wie PSGML neue Elemente einrückt, aber mit einigen Griffen zur (TAB)-Taste ist das schnell erledigt.

3.2.2. Zukunft

Leider ist es wohl nicht mehr vorgesehen, PSGML zum Verarbeiten der neueren XML-Schema zu bewegen, so dass dieses Paket wohl langsam in den nächsten Jahren veralten wird. Es gibt da ein neues Paket names nxml (FIXMEREF) aber das wird nur für den GNU Emacs entwickelt und eine Portierung dürfte aufgrund der enormen Größe des Paketes schwierig sein. Es bleibt abzuwarten, was die Zeit bringt.

3.3. html-mode – PSGML- -

Da liegt mit PSGML eine richtige SGML-Entwicklungsumgebung vor und ihre Hauptanwendung liegt in der Bearbeitung von HTML-Dateien[1]. Schade eigentlich. Aber dadurch werden auf jeden Fall HTML-Dateien, die aus dem XEmacs kommen, automatisch besser. Nicht nur, dass hier für richtiges (und konfigurierbares) Einrücken gesorgt wird, auch die Korrektheit eines Dokuments bezüglich seiner DTD wird wahrscheinlicher, sofern der Entwickler nur mal (C-c)

[1] Daher der Name dieses Kapitel: PeEssGeEmmEllMinusMinus

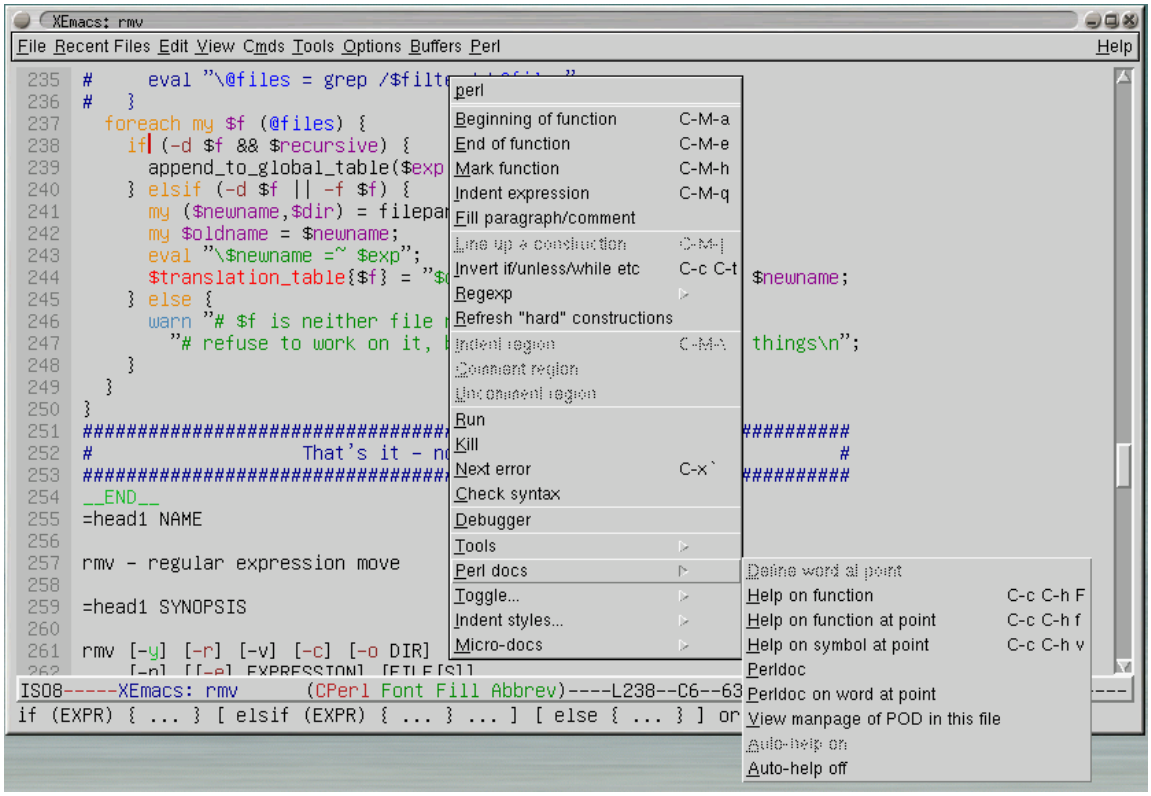


Abbildung 3.2.: Ein Buffer im cperl-mode. In der EchoArea ist die automatische Kurzhilfe zum if-Befehl zu erkennen, das Kontextmenu zeigt weitere Möglichkeiten, an Hilfe zu kommen.

C-o drückt. Ist PSGML einmal so eingestellt, dass notwendige Tags automatisch eingefügt werden, ist eine weitere Fehlerquelle abgestellt.

Zudem gibt es noch einen Minor-Mode, der Abkürzungen für viele HTML-Tags definiert. Ein mutiges C-h m sorgt für eine lange Liste von vordefinieren Tastenkombinationen.

Die ausführlichen Menus des Modes, sobald eine HTML-Datei geöffnet wurde, bieten genügend Stoff zum lange drin Stöbern.

3.4. CPerl, also Perl im cperl-mode

„Nur perl kann Perl parsen“ so sagt man sich unter Perlprogrammierern. Leider stimmt das auch für den XEmacs. Er tut sein Bestes und schafft eine Menge, aber ab und an trifft man doch auf falsche Berechnungen der Einrückung oder falsche Syntaxeinfärbungen. Ansonsten liefert der cperl-mode eine hervorragende Plattform, um Perlprogramme zu entwickeln.

Zunächst sei die Navigation erwähnt, mit der man durch den Buffer navigieren kann. CPerl arbeitet sowohl mit der Speedbar als auch mit dem functions-menu (FIXMEREFWRITE) und bietet neben dem eigene Funktionen, um z. B. an den Anfang einer Funktion zu springen (siehe dazu das Kontextmenu auf der rechten Maustaste).

Eine weitere wichtige Stütze ist die automatische Kurzhilfe von vielen Funktionen und Operatoren. Einmal aktiviert (im Menu unter „Perl Docs“), zeigt sie in der EchoArea bei kurzem Verweilen an einer Stelle eine kontextbezogene Hilfe an. Dieses ist im gezeigten Screenshot in Abbildung 3.2 zu erkennen. Auch der Zugriff auf die vollständige Perldokumentation kann aus dem XEmacs erfolgen. Früher musste man sich ein separat gepflegtes info-File mit der

Dokumentation aller Perlbefehle besorgen, heute verwendet man einfach das von Perl mitgelieferte `perldoc`, wie es das Kontextmenu in der Abbildung bereits andeutet. Damit muss man also nicht mehr zur Shell wechseln, dort

```
shell> perldoc -f unpack
```

angeben, weil man wieder mal die Verwendung von `unpack` vergessen hat und dann zurück in den XEmacs gehen. Stattdessen schreibt man einfach `unpack` in seinen Buffer, wartet kurz, ob die Kurzhilfe vielleicht schon aussreicht, um der Erinnerung auf die Sprünge zu helfen und –wenn nicht– tippt dann kurz mal auf (M-x) `cperl-perldoc-at-point`. Wer da häufig verwendet, wird es sicherlich auf eine kurze Taste legen wollen (siehe Abschnitt 4.2).

Geschmackssache ist die Fähigkeit des cperl-modes, beim Tippen diverser Schlüsselwörter oder -zeichen, den Versuch zu starten, besonders pfiffig zu sein und Tipparbeit abzunehmen. Es ist schon nett, wenn man einfach nur

```
while SPACE
```

tippt, und dann steht danach

```
while () {
}
```

im Buffer und der Cursor zwischen den Klammern. Will man allerdings eine eigene Laufvariable angeben:

```
for my $i (1..10) {
    print "$i\n";
}
```

...kann das auch als störend empfunden werden. Generell ist dieses Verhalten (auch in anderen Modes) als *electric* bekannt. Man kann einstellen, ob man seine Klammern und das Semikolon (fügt dann automatisch Zeilenumbrüche ein) und gewisse Schlüsselwörter elektrisch haben möchte. Wer ohnehin alle Regler nach rechts gedreht sehen möchte, setzt einfach `cperl-hairy` auf `t`, dann sind alle elektrischen Features ebenso angeschaltet, wie noch einige andere Kleinigkeiten.

Eine sehr häufig auftauchende Frage gilt den neueren CPerl-Modes und deren Verhalten, überflüssige Leerzeichen durch einen Unterstrich zu markieren. Meistens wird gefragt, wie man das wieder loswird. Abgesehen davon, dass man einfach die Leerzeichen entfernen kann, kann man auch die Schriftart für `cperl-invalid-face` auf einen passenden Wert setzten, indem man per (M-x) `customize` (RETURN) `cperl-faces` (RETURN) im aufpöppenden Buffer „Invalid Face“ auf `'(quote default)` setzt. Dann Abspeichern für ein dauerhaftes Abschalten und gut ist.

Wie man nun den Perlinterpreter gleich selber startet oder aber einen Syntaxcheck ausführen lässt, dessen Ergebnis XEmacs in einem eigenen Buffer zum direkten Anspringen der Fehlerstellen anzeigt, kann man erneut dem Kontextmenu entnehmen.

Jetzt noch schnell ein paar hausgemachte Skeletons definiert (Abschnitt 3.6.2) und dann kanns losgehen.

3.4.1. Konfiguration

Beispielkonfiguration findet sich im langen Abschnitt, der meine `personal.el` zeigt (4.5).

3.5. Java mit JDE – hypeorientiertes Programmieren

Mitte 2004 kam der Zeitpunkt, als ich dann doch nochmal mit Java begonnen habe. Weil es vor Ort das eingesetzte Instrument war und weil ich zwar um JDE wusste, es jedoch nicht konfiguriert war, habe ich ausnahmsweise zu einem anderen Tool gegriffen: NetBeans. Fancy stuff, mit CVS und GUI Builder und `ant` und Onlinehilfe zu den Funktionen sowie einer kontextsensitiven Vervollständigung von Funktionen, die jederzeit weiß, welche Funktionen an der aktuellen Cursorposition erlaubt sind. Nett. Allein, der Editor hat weh getan. Da fehlte so viel ...

Nun gut, zwei Tage später habe ich begonnen, JDE zu konfigurieren und einen weiteren Tag später hatte ich all die genannten Features ebenfalls, nur der GUI Builder fehlte noch.

Vervollständigung Mal ganz abgesehen von dabbrev, das ich auch in Java viel nutze, gibt es die Funktionen `jde-complete` und `jde-complete-in-line`, die entweder direkt im Code die Funktionen einfügen (zweiteres) und bei weiteren Aufrufen durch die Möglichkeiten gehen, oder aber eine Auswahl anbieten, in dem bequem die Funktion ausgesucht werden kann.

Dokumentation Ist erst einmal bekannt, wo die HTML-Dokumentation für das JDK zu finden ist (Variable `jde-jdk-doc-url` oder aber die Ausgabe der Funktion `jde-get-jdk-dir`), kann man mit `jde-help-browse-jdk-doc` die Hilfe in einem einstellbaren Webbrowser anzeigen lassen. Wer sich dann noch mit einem internen Browser im XEmacs zufrieden gibt (w3 oder w3m) hat ein ziemlich „integriertes“ Feeling. Eine kontextsensitive Hilfe liefert die Funktion `jde-help-symbol`.

JavaDoc Nach wenigen Tagen sah mein Code auch besser aus als anderer aus dem gleichen Projekt. Nicht nur weil er korrekt eingerückt war, sondern weil jede Klasse, jede Funktion ihre JavaDocs hatte. Das geht aber auch so einfach: (M-x) `jde-javadoc-checkdoc` und es wird überall geschaut, was den gängigen Regeln nach in Ordnung ist und was nicht. Tja, und wenn mal was fehlt: (M-x) `jde-javadoc-autodoc-at-line` hilft da weiter.

Buildsystem: ant Mit `ant` kommt für Java eine Alternative zu `make`. Im JDE kann man mit der Variablen `jde-build-function` ((M-x) `customize-variable` (RET) `jde-build-function`) einstellen, welche Funktion zum Bauen eines Projektes verwendet werden soll. Neben den genannten Möglichkeiten kann man auch eine ganz eigene Funktion schreiben. Ein Aufruf von `jde-build` ruft dann `ant` mit den (hoffentlich) passenden Parametern auf.

Programmstart Und auch den Umweg über die Shell zum Starten des Programmes kann man sich sparen. Gleich aus dem XEmacs heraus (M-x) `jde-run` aufgerufen, sofern `jde-run-executable` oder aber `jde-run-application-class` gesetzt wurde. Das einzig unschöne daran ist, dass dabei die aktuelle Windowconfiguration zerstört wird. Da man die ja aber vorher schnell in ein Register speichern kann ((M-x) `window-configuration-to-register` (1), später dann (M-x) `jump-to-register` (1)) stört das nicht so arg.

3.5.1. JDE Konfiguration

Anregungen für eine gute Konfiguration. Allerdings nutzt JDE eine Eigenschaft von customize, so dass die eigentliche Konfiguration darüber erledigt wird. Customizevariablen haben einen Defaultwert, der zwar überschrieben werden kann, aber immer vorhanden ist. Somit kann JDE für jedes Javaprojekt eigene Konfigurationen vornehmen.

Zunächst die entsprechenden Einträge in meiner customize-Datei:

```
(custom-set-variables
 '(jde-project-name "MySuperCoolProject")
 '(jde-build-function (quote (jde-ant-build)))
 '(jde-complete-function (quote jde-complete-minibuf))
 '(jde-setnu-mode-enable t)
 '(jde-complete-signature-display-time 3)
```

Dann eine mögliche Konfiguration für Tastenkombinationen und ähnliches:

```
;; If you want Emacs to defer loading the JDE until you open a
;; Java file, edit the following line
(setq defer-loading-jde t)

(if defer-loading-jde
    (progn
      (autoload 'jde-mode "jde" "JDE mode." t)
      (setq auto-mode-alist
        (append
         '(("\\.java\\'" . jde-mode))
         auto-mode-alist)))
  (require 'jde))
```

```
;; Note that the customization for JDE is done via customize, because
;; JDE wants it that way
(defun ska-jde-mode-keys ()
  "Setting local keybindings for major mode: JDE."
  (local-set-key '[(control b) (control b)]      'jde-build)
  (local-set-key '[(control b) (control h)]      'jde-help-symbol)
  (local-set-key '[(control b) (control space)]  'jde-complete-in-line)
  (local-set-key '[(control b) (space)]          'jde-complete)
  (local-set-key '[(control b) (control r)]      'jde-run)
  )

(defun my-jde-mode-hook ()
  (turn-on-setnu-mode)
  (turn-on-font-lock)
  (auto-fill-mode 1)
  (setq indent-tabs-mode nil)

  (setq c-basic-offset 2)

  (ska-coding-keys jde-mode-map)
  (ska-jde-mode-keys)
  (make-local-hook 'write-contents-hooks)
  (add-hook 'write-contents-hooks #'ska-untabify
            nil t))

(add-hook 'jde-mode-hook 'my-jde-mode-hook)
```

Die projektspezifische Konfiguration erfolgt über eine spezielle Datei: `prj.el` im Projektverzeichnis. Eine solche Datei könnte folgenden Inhalt haben.

```
(jde-project-file-version "1.0")
(jde-set-variables
 '(jde-global-classpath
    (quote ("."
            "/home/user/projekte/java/tollebibliothek"
            "/home/user/projekte/java/wichtigeklassen/"))))
```

JDE ist eine ziemlich dicke Bibliothek mit einer Vielzahl an Fähigkeiten und Möglichkeiten. Die hier beschriebenen Konfigurationen und Funktionen können nur der obere Zipfel des Eisberges sein. Es reicht aber bereits, um NetBeans abzulösen.

3.6. Skeletons

So ziemlich jeder halbwegs ernstzunehmende Editor bietet die Fähigkeit, bestimmte Textbausteine einfügen zu lassen. Natürlich bekommen wir XEmacs-User noch viel mehr geboten. Mit den Skeletons steht uns ein sehr leistungsfähiger Mechanismus zur Verfügung.

Die Dokumentation dazu liefert der XEmacs mit

```
C-h f skeleton-insert
```

Wer noch tiefergehendes Verständnis erwerben möchte, sollte mal einen Blick die Dateien `skeleton.el` und `sh-script.el` werfen.

3.6.1. Definition eines Skeletons

Um ein Skeleton zu definieren dient die Funktion `define-skeleton`:

```
(define-skeleton ska-skel-ruby-def
  "Inserts a new ruby function definition at point."
  "name: "
  < "def " str
  "("
  ("Argument, %s: "
   str & ", ") & -2 & ")" | -1
   \n _ "\n"
  "end" (progn (ruby-indent-command) ""))
```

Ihr gibt man den Namen mit, den das Skeleton bekommen soll (hier `ska-skel-ruby-def`), und den man später als vollwertige Funktion zur Verfügung hat. Es folgt der obligatorische Dokumentations-String: „Inserts a new ruby function definition at point.“.

Das nächste Argument ist entweder `nil` oder aber im Beispiel der Fragetext für eine Anfrage beim Benutzer: `"name: "`. Im zweiten Fall bekommt man den vom Benutzer eingegebenen String in der Variablen `str` geliefert, die man später verwenden kann.

Alle Strings, also Zeichenketten, die in der Skeleton-Definition in doppelte Anführungszeichen (") eingefasst werden, werden einfach so, wie sie dort stehen, eingefügt.

Möchte man gerne elisp-Code ausführen, fasst man jenen Code in einen progn-Aufruf ein und liefert am Ende einen leeren String zurück: `(progn (ruby-indent-command) "")`.

Eine normale runde Klammer beginnt ein rekursives Skeleton: Hier kann erneut eine Abfrage formuliert werden, deren Antwort man in der Variablen `str` erhält. Im Beispiel werden die Argumente für eine Ruby-Funktion erfragt. Ein solcher Block wird solange durchlaufen, bis eine leere Eingabe erfolgt. Für die weitere Diskussion des Beispiels s. unten.

Sonderzeichen

Es existieren noch einige Zeichen mit besonderer Bedeutung:

Zeichen	Bedeutung
`>`	Einrücken der aktuellen Zeile.
`\n`	Einrücken der aktuellen Zeile und Wechsel in die nächste.
`_`	Festlegen der Cursorposition nach dem vollständigen Einfügen des Templates. Achtung, das klappt nicht immer so ganz richtig, speziell wenn man elisp-Code ausführt.
`>`	Einrücken der aktuellen Zeile.
`-3`	Das Minuszeichen gibt an, dass von der aktuellen Position die angegebene (3) Anzahl Zeichen gelöscht werden sollen.

Bedingungen

In obiger Tabelle fehlen noch einige Sonderzeichen, die jedoch gesondert behandelt werden sollen: `&` und `|`. Sie tauchen auch bereits in obigem Beispiel auf und dienen zum Testen von Bedingungen.

Falls der vorherige Ausdruck die Cursorposition verändert hat (z. B. durch Einfügen des Inhalts der Variablen `str` in den Buffer), wird ein mit `&` angeschlossener Teil behandelt. Falls nicht, entfällt diese Bearbeitung. Somit ist also `&` ein logisches UND. Entsprechendes gilt für `|`, das logische ODER.

Im Beispiel dient das dazu, für jedes weitere Argument, das eingegeben wurde ein `", "` in den Text einzufügen, und wenn die Eingabe beendet wurde, jene Zeichen zu löschen und dafür eine schliessende Klammer mit einem Zeilenumbruch zu erzeugen. Falls hingegen keine Argumente angegeben wurden, werden die Klammern entfernt.

Somit ist also der Ausdruck

```
("Argument, %s: " str & ", ") & -2 & ")" | -1
```

wie folgt zu lesen:

1. Erfrage ein Argument: `("Argument, %s: "`
2. Schreib es in den Buffer und füge ein Komma und ein Leerzeichen an: `str & ", ")`
3. Führe diese Aktion solange aus, bis der Anwender eine leere Eingabe macht: das wird durch die Klammerung erreicht.
4. Wenn es eine Eingabe gegeben hat, lösche zwei Zeichen (das Lerzeichen und das Komma): `& -2`
5. Und füge gleich noch die schliessende Klammer für die Argumentenliste ein: `& ")"`.
6. Hat es hingegen keine Eingabe gegeben, lösche lediglich ein Zeichen, nämlich die öffnende Klammer der Argumentenliste: `| -1`

3.6.2. Beispiele

Eine Perl-Subroutine, die nach beliebig vielen Argumenten fragt:

```
(define-skeleton ska-skel-perl-sub
  "Insert a perl subroutine with arguments."
  "Subroutine name: "
  "sub " str " {"
  \n "my (" ("Argument name: " "$" str ", ") -2 ") = @_;"
  "\n"
  \n _
  \n "}" '(progn (indent-according-to-mode) nil)
  \n)
```

Und hier bereiten wir uns den Code für ein größeres Perl-Projekt vor. Es sollte beachtet werden, dass hier davon ausgegangen wird, dass die Datei mit `auto-insert` bereits vorbereitet wurde (vergleiche dazu Abschnitt 3.7).

```
(define-skeleton ska-skel-perl-project
  "Insert much perl code, preparing a real world project."
  (nil)
  "use Getopt::Long;\n"
  "use Pod::Usage;\n"
  "#############################################################################\n"
  "##                              OPTIONS\n"
  "#############################################################################\n"
  "GetOptions("
  \n "\"help|h!\" => \\my $help,"
  \n "\"version|v!\" => \\my $version"
  \n ") or pod2usage("
  \n "verbose => 0,"
  \n "exitstatus => 1"
  \n ");"
  \n "if ($help) {"
  \n "pod2usage("
  \n "verbose => 1,"
  \n "exitstatus => 0"
  \n ");"
  \n "}"
  \n "if ($version) {"
  \n "print $Version;"
  \n "exit 0;"
  \n "}"
  \n "#############################################################################"
  \n "##                                MAIN"
  \n "#############################################################################"
  \n ""
  \n "#############################################################################"
  \n "##                                SUBS"
```

```
\n "#############################################################################"
\n "__END__\n"
"#############################################################################\n"
"##                                Now Docs...\n"
"#############################################################################\n"
"=head1 NAME"
"\n"
\n (file-name-nondirectory buffer-file-name) " - DESCRIBE ME"
"\n\n"
"=head1 SYNOPSIS"
"\n"
\n (file-name-nondirectory buffer-file-name) " [-h] [-v]"
"\n\n"
"=head1 OPTIONS"
"\n\n"
"=over 1"
"\n\n"
"=item B<-h|--help>"
"\n"
\n "Print help message and exit successfully."
"\n\n"
"=item B<-v|--version>"
"\n"
\n "Print version information and exit successfully."
"\n\n"
"=back"
"\n\n"
"=cut\n"
"")
```

3.7. Auto-Insert

Wann immer XEmacs eine leere Datei zur Bearbeitung öffnet, kann man sich einiges an Text bereits einfügen lassen. Das passende Paket dazu heißt AutoInsert, und es enthält mehr als nur einen Mechanismus dafür. Die wichtigste Variable in diesem Zusammenhang ist `auto-insert-alist`. Sie hat die Form

```
((CONDITION1 . ACTION1)
 (CONDITION2 . ACTION2)
 (CONDITIONn . ACTIONn))
```

Dabei ist jede einzelne CONDITION entweder ein regulärer Ausdruck, der auf den Dateinamen passt, z. B. `"\\.c$"` für C-Dateien, die gewöhnlich auf ein `.c` enden. Alternativ darf hier auch ein Ausdruck der Form `(CONDITION . DESCRIPTION)` verwendet werden, also beispielsweise

```
("\\.h*$" . "C Header")
```

Die zweite Möglichkeit für CONDITION ist ein Symbol, das auf den jeweiligen Majormode passt, z. B.

```
(ruby-mode . "Ruby Program")
```

Derart konfiguriert kann dann die Funktion `auto-insert` entscheiden, welche ACTION ausgewählt werden soll. Auch hier existieren wieder zwei Möglichkeiten:

1. ACTION kann ein Dateiname sein, der die Datei angibt, die den einzufügenden Text enthält. Der Dateiname kann absolut angegeben werden oder aber relativ zum Verzeichnis, das in der Variablen `auto-insert-directory` definiert wurde.

2. Alternativ kann ACTION aber auch ein echtes Skeleton (s. Abschnitt 3.6) sein, womit einem dann der komplette Funktionsumfang von XEmacs zur Verfügung steht.

Abschließend ein Beispiel für die Skriptsprachen Perl und Ruby:

```
(setq auto-insert-alist
      '(
        ((perl-mode . "Perl Program")
         nil
         "#! /usr/bin/perl\n#\n"
         "# File: " (file-name-nondirectory buffer-file-name) "\n"
         "# Time-stamp: <>\n"
         "# $Id: $\n#\n"
         "# Copyright (C) " (substring (current-time-string) -4)
         " by " auto-insert-copyright "\n#\n"
         "# Author: "(user-full-name) "\n#\n"
         (progn (save-buffer)
                (shell-command (format "chmod +x %s"
                                       (buffer-file-name)))
                "")
         "# Description:\n# " _ "\n"
         "use strict;\n"
         "use warnings;\n"
         "use Data::Dumper;\n"
         (when (yes-or-no-p "Is this a real project (or just a script)? ")
           (ska-skel-perl-project)
         ))
        ;;}}}
        ;;{{{ Ruby Programm
        ((ruby-mode . "Ruby Program")
         nil
         "#! /usr/bin/ruby -w\n#\n"
         "# File: " (file-name-nondirectory buffer-file-name) "\n"
         "# Time-stamp: <>\n"
         "# $Id: $\n#\n"
         "# Copyright (C) " (substring (current-time-string) -4)
         " by " auto-insert-copyright "\n#\n"
         "# Author: "(user-full-name) "\n#\n"
         "# Description: " _ "\n#\n\n"
         "class " (replace-in-string
                   (upcase-initials
                     (file-name-nondirectory
                     (file-name-sans-extension
                       buffer-file-name)))
                   "_"
                   "")
         "\n"
         "def initialize()" (progn (ruby-indent-command) "")
         "\nend" (progn (ruby-indent-command) "")
         "\n end" (progn (ruby-indent-command) "")
         )
```

Hier werden die richtigen Shebangs eingesetzt, der aktuelle Dateiname, wichtige Bibliotheken[2] und einiges mehr. Beim Code für Perl sind noch zwei Sachen hervorzuheben:

1. Das Skript wird auch gleich ausführbar gespeichert. Es ist doch wirklich lächerlich, wie oft man in seinem (Programmierer-) Leben schon `chmod +x` getippt hat, wenn man ein neues Skript ins Leben gerufen hat!

2. Der Code verwendet noch eine weitere Hilfsfunktion. Die Zeile mit der `(yes-or-no-p ...)`-Abfrage fragt nach, ob man hier nur ein schnelles Skript erstellen möchte oder aber ein richtiges Projekt. In letzterem Falle verwendet es noch eine Hilfsfunktion, die weitere angenehme Codefragmente einsetzt, und im Abschnitt 3.6 vorgestellt wurde.

Einige Vorraussetzungen, damit dieser Code richtig funktionieren kann, sind:

[2] Die Manpage zu Perl definiert es als einen Bug in der Sprache, dass der Schalter `-w` nicht absolut notwendig ist. Dieses Thema wäre mit dieser Konfiguration bereits erledig

```
;; expliziter Copyright-Holder
(setq my-copyright-holder "Stefan Kamphausen")

;; ...
;; expliziter oder aber default Copyright-Holder
(if (boundp 'my-copyright-holder)
    (setq auto-insert-copyright my-copyright-holder)
  (setq auto-insert-copyright (user-full-name)))
```

3.8. C und C++ – Old-School Coding

Manchmal fragt man sich, warum selbst im XEmacs das Arbeiten mit C und C++ so archaisch anmutet. Es existiert beispielsweise kein Kontext-Menu im c-mode. Aber dann kommt doch wieder die Erkenntnis, dass C halt alt ist. Dennoch gibt es eine Menge angenehmer Dinge, die Compilation, Debugging, Navigation und Bearbeitung in und von C/C++-Projekten erleichtern:

- speedbar – Nicht zu verzichten! Eines der besten Navigationswerkzeuge im XEmacs.
- ECB – Der Emacs Code Browser, ein mächtiger Klassen-Browser.
- functions-menu – Zum Navigieren in der aktuellen Datei.
- `paren-backwards-message` – Bei (zu) großen Blöcken wird der Text bei der passenden Klammer angezeigt.
- ctypes – erkennt selbsdefinierte Typen und Klassen und sorgt für das Syntax-Highlighting
- GUD – Der Grand Unifying Debugger
- (M-x) `compile` – ruft `make` oder ein anderes Programm zum Kompilieren, parst die Ausgabe und lässt die Fehlerstellen direkt anspringen.
- (M-x) `mode-compile` – Erstellt die Kommandozeile für (M-x) `compile`.
- setnu – Zeilennummer.
- etags – TAGS-Interface, ebenfalls zum Navigieren.

3.9. SQL – Ein Sprache für Manager

Dem Vernehmen nach wurde SQL tatsächlich dafür konzipiert, dass Manager ihre Reports (Umsatzzahlen etc.) aus einer Datenbank holen können. Für den EDV-Mann in einem Unternehmen in den 70ern war das bestimmt eine gute Sache. Heute stecken wir immer noch mit dieser Sprache in der Datenbankerei fest[3]. Immerhin kann man sich das Leben mit XEmacs ein wenig angenehmer gestalten.

Für SQL exisiteren die üblichen Angenehmlichkeiten wie Syntaxhighlighting sowie Einrückung. Zudem kann sich XEmacs zu einem Datenbank-Prozess verbinden und dann Teile des SQL-Buffers dorthin schicken. Spezialisierungen für gängige Datenbanksysteme sind vorhanden.

Ein wenig Code ...

```
(autoload 'sql-mode "sql" "SQL Editing Mode" t)
(setq auto-mode-alist
      (append
       '(("\\.sql$" . sql-mode))
       auto-mode-alist))
(add-hook 'sql-mode-hook
          '(lambda ()
```

[3] Die zwischen den Zeilen leicht mitschwingende Abneigung gegen SQL spiegelt natürlich nur die persönliche Meinung des Autors wieder

```
;; lokale keybindungs hierhin
;; ...
;; speziell fuer Oracle...
(sql-highlight-oracle-keywords)
))
```

Mit (M-x) `sql-mysql` wird beispielsweise eine Verbindung zu einer MySQL-Datenbank erzeugt. Danach kann in einem Buffer, der sich im sql-mode befindet, über das Menu der SQLi-Buffer auf den vorher geöffneten gesetzt werden. Danach können Teile des Buffers an die Datenbank gesendet werden.

3.10. CVS – Immer auf Stand

Immer die aktuellen Quellen des Lieblings-Editors oder aber die Projekt-Verwaltung mit dem Bekannten auf einer Open-Source-Entwicklungs-Seite? Oftmal lautet die Lösung dazu CVS, das Concurrent Versions System. Neben dem klassischen Kommandozeilen-Programm tummeln sich allerhand andere Werkzeuge, unter anderem auch *PCL-CVS* im XEmacs.

Ist ein Programm-Baum erst einmal lokal gespeichert[4], reicht es, im XEmacs einmal (M-x) `cvs-status` aufzurufen. Sofort wird der XEmacs nach dem zu untersuchenden Verzeichnis fragen und nach Eingabe dessen die dort hinterlegte Methode zum Abgleich mit dem Server verwenden. Der neu entstandene CVS-Buffer erklärt sich eigentlich von selber, wenn man einmal die rechte Maus auf einem Eintrag bemüht.

Das Wichtigste in Kürze:

(O)	Update (sprich Opdate ;-). Holt die aktuellste Version vom Server.
(=)	Zeige die Unterschiede als Ausgabe des `diff`-Programms an.
(C)	Commit: spiele die eigene Version auf den Server.

Es sei an dieser Stelle darauf hingewiesen, dass es auch für die Versionsmanagementsysteme `arch` und `svn` entsprechende XEmacs-Frontends gibt, die eine ähnliche Struktur aufweisen (siehe[16] und FIXMEREF).

3.11. sawfish-mode

Sawfish ist ein Window-Manager, also ein eher kleines und schlankes Programm, das Rahmen um die Programm-Fenster auf dem Bildschirm zeichnet und dafür sorgt, dass man die Fenster auch bewegen kann (natürlich noch etwas mehr als das). Interessant wird Sawfish dadurch, dass auch er im Wesentlichen in einem LISP-Dialekt (librep) geschrieben ist. Damit geht dann einher, dass man in einem Sawfish-Client LISP-Code evaluieren kann, der sofort im laufenden Window-Manager aktiv wird und eine Aktion durchführt oder aber eine Funktion zur Verfügung stellt.

Es gibt nun einen XEmacs-Mode, der uns genau diese Verbindung zum Sawfish liefert ([14]).

Konfigurations-Code:

```
;; needs sawfish.el from
;; <URL:http://www.davep.org/emacs/#sawfish.el>
(autoload 'sawfish-mode "sawfish" "sawfish-mode" t)
(setq auto-mode-alist
      (append
       '(("\\.sawfishrc$"  . sawfish-mode)
         ("\\.jl$"         . sawfish-mode)
         ("\\.sawfish/rc$" . sawfish-mode))
       auto-mode-alist))
```

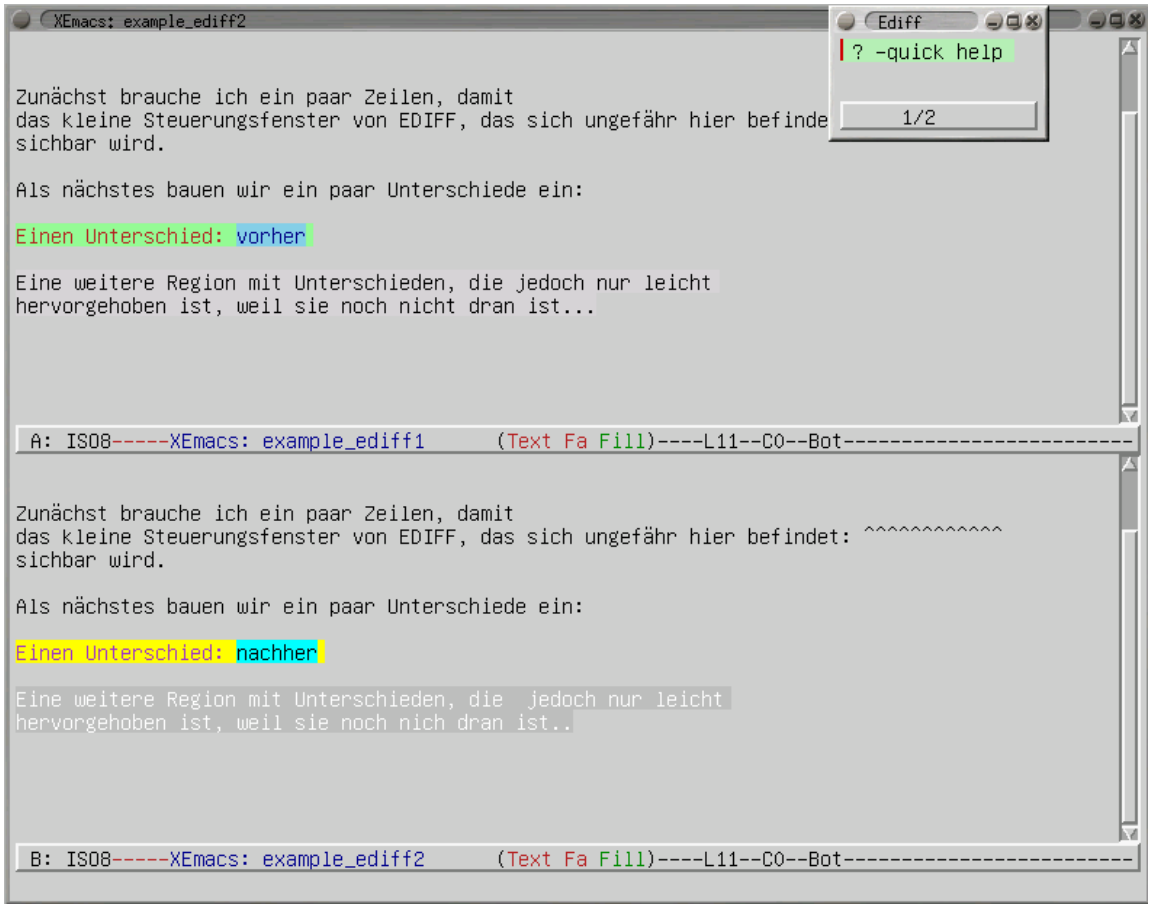


Abbildung 3.3.: Ein Ediffbeispiel. Die Dateien werden im Tarball mitgeliefert. Oben rechts sieht man den kleinen Kontrollframe, in dem die Eingabe eines „?“ bewirkt, dass er sich in einen größeren mit einer Angabe der meisten Tastaturbelegungen verwandelt. Im Hauptframe werden die beiden Dateien in je einem Fenster dargestellt und die Unterschiede sogar nochmal genau in der Zeile hervorgehoben. Die untere Region ist leicht hervorgehoben, da sie der nächste Unterschied sein wird. Aber das steht ja eigentlich auch so im Text der Beispiele.

3.12. ediff – Was macht es für einen Unterschied?

Viele Uni*x-Anwender kommen früher oder später in die Situation, das Programm `diff` zu verwenden, um Unterschiede in Dateien aufzuspüren. Es ist ein Kommandozeilen-Tool, das auch für das *Patchen* (Flicken) von Programm-Quellen verwendet wird.

Selbstredend möchte man die Ergebnisse solcher Untersuchungen auch oft gleich in Dateien einbauen. Oder am besten gleich ganze Verzeichnis-Bäume rekursiv bearbeiten. Und selbstredend geht das im XEmacs :)

3.12.1. Unterschiede in Dateien

Mit (M-x) `ediff-files` startet man eine Session (Sitzung) in ediff, dem XEmacs-Front-End zu `diff`. Alternativ findet man ediff in den Menus unter Tools->Compare-> ...

XEmacs erwartet als nächstes zwei Dateinamen (im Minibuffer), nämlich die zu vergleichenden Dateien. Diese werden auch als Buffer A und Buffer B bezeichnet. Es öffnet sich dann ein neuer, sehr kleiner Frame, der zwei Informationen enthält (vgl. auch Abbildung 3.3):

1. Einen Hinweis darauf, das sich selbiges Fenster durch Drücken von (?) in ein Hilfe-Fenster verwandeln lässt.

2. Eine sehr verkleinerte Modeline, die die Anzahl an Unterschieden und den aktuell angezeigten Unterschied wiedespiegelt.

Die Bedienung ist denkbar einfach durch Tasten und ihre Belegungen zeigt Tabelle 3.1.

Tasten	Wirkung
(SPACE)	Springt zum nächsten Unterschied ohne irgendwelche Änderungen vorzunehmen.
(A) ((B))	Übernimmt die Version aus Buffer A in Buffer B, bzw. umgekehrt.
(q)	Beendet die ediff-Sitzung mit vorheriger Abfrage. Lässt beide Buffer geöffnet.
(rb)/(ra)	Rückgängig im jeweiligen Buffer („r“ für „restore“)
(wb)/(wa)	Speichern des jeweiligen Buffers („w“ für „write“)
(|)	Ändert das Layout von *übereinander* zu *nebeneinander*.
(m)	Zieht das XEmacs-Fenster auf volle Größe auf.
(##)	Ingoriert Unterschiede, die lediglich Whitespaces (also SPACE, TAB, NEWLINE) sind.

Tabelle 3.1.: Bedienung und Tastenbelegung von Ediff.

Weitere Kommandos entnehme der begeisterte Anwender dann der Dokumentation (erst das (?), ausführliche Info-Dokumentation mit (E).

3.12.2. Farbcodes

Die mittels `diff` entdeckten Unterschiede werden im Buffer farblich codiert dargestellt und sind in Tabelle 3.2 aufgelistet.

[4] Wie das geht, wird üblicherweise auf den allermeisten Webseiten, die CVS-Downloads anbieten, so genau beschrieben, dass man den Code einfach von dort in die Kommandozeile kopieren kann

Farbcode	Bedeutung
Grau	Unterschied, aber nicht der aktive
Gelb	Nur Whitespace-Unterschiede
Blau	Hervorgehoben in einer ohnehin schon farbigen Zeile steht der tatsächliche Unterschied, sofern er festgestellt werden konnte.
Weitere Farben	Weiss ich auch nicht, hab ich aber auch noch nie gebraucht ;-).

Tabelle 3.2.: Farbcodes von Ediff.

3.12.3. Verzeichnisse

Um ganze Verzeichnis-Bäume rekursiv zu untersuchen, benötigt man (M-x) `ediff-directories` Dann wird ein Buffer geöffnet, der eine Liste von Dateien und Verzeichnissen mit einigen Informationen zu selbigen enthält. Jedes Verzeichnis und jede Datei sind eine *Session*, die mit (RETURN) gestartet werden kann.

Dann verhält sich ein geöffnetes Verzeichnis genauso wie das erste Verzeichnis, und eine Datei so, wie oben beschrieben. Wenn eine Datei-Session beendet wurde, geht der Cursor gleich auf die nächste Session, so dass man gleich wieder starten kann. Dazu empfiehlt es sich, bei der Frage nach einem Filter am Anfang, alle Backup-Dateien auszublenden, wenn man welche hat.

3.12.4. Weiteres

Das ediff-Paket liefert noch viel mehr Funktionalität, aber ich bin nur so weit eingestiegen, wie hier beschrieben, und habe bisher noch nicht mehr gebraucht.

Mit (M-x) `ediff-files3` lassen sich auch drei Dateien vergleichen. Dann existieren noch Funktionen auf der (C)-Taste, für den dritten Buffer. Auch drei Verzeichnisse lassen sich vergleichen ...

3.13. VM

3.13.1. Einleitung

„You've got email." „New Mail arrived.". Ein Icon ändert sein Aussehen von einem leeren Briefkasten zu einem gefüllten. So oder so ähnlich geschieht es tagtäglich auf mindestens ganz vielen Rechnern -eher mehr- weltweit. Hat sich gandalf aus dem Chat wieder gemeldet? Ist meine Bestellung beim Online-Shop aufgenommen worden? Oder ist es Traffic auf der XEmacs-Mailingliste? Aufklärung schafft hier nur ein User Mail Client, und genau dort wird es wahnsinnig. Manchmal muss man sich fragen, ob es von irgendeiner Gattung Programm mehr Variationen gibt als von Mailreadern[5]. Angefangen im Terminal mit `mail`, `pine` oder `elm` über diverse eigene Clients bis hin zu den integrierten Mailreadern in Gnome und KDE, *Evolution* und *KMail*. Mozillas Mailprogramm *Thunderbird* zählt sicherlich auch zu den massiv verbreiteten und für jedes verfügbare Toolkit (Tk, GTK+, QT, ncurses, etc) gibt es mehrere eigenständige Projekte. Hier soll nun ein weiteres vorgestellt werden: Der VM.

3.13.2. Vorbereitung

VM findet sich einerseits automatisch in den Distributionen andererseits auf der VM Homepage ([9]), und wurde von Kyle Jones geschrieben. VM funktioniert sowohl in einem Terminal als auch mit X, was für remote Verbindungen sehr nützlich sein kann. Man startet ihn am besten mittels

```
xemacs -f vm
```

[5] OK, IRC Clients, Textedioren und Thumbnail-Gallery-Generatoren gibt es ähnlich oft

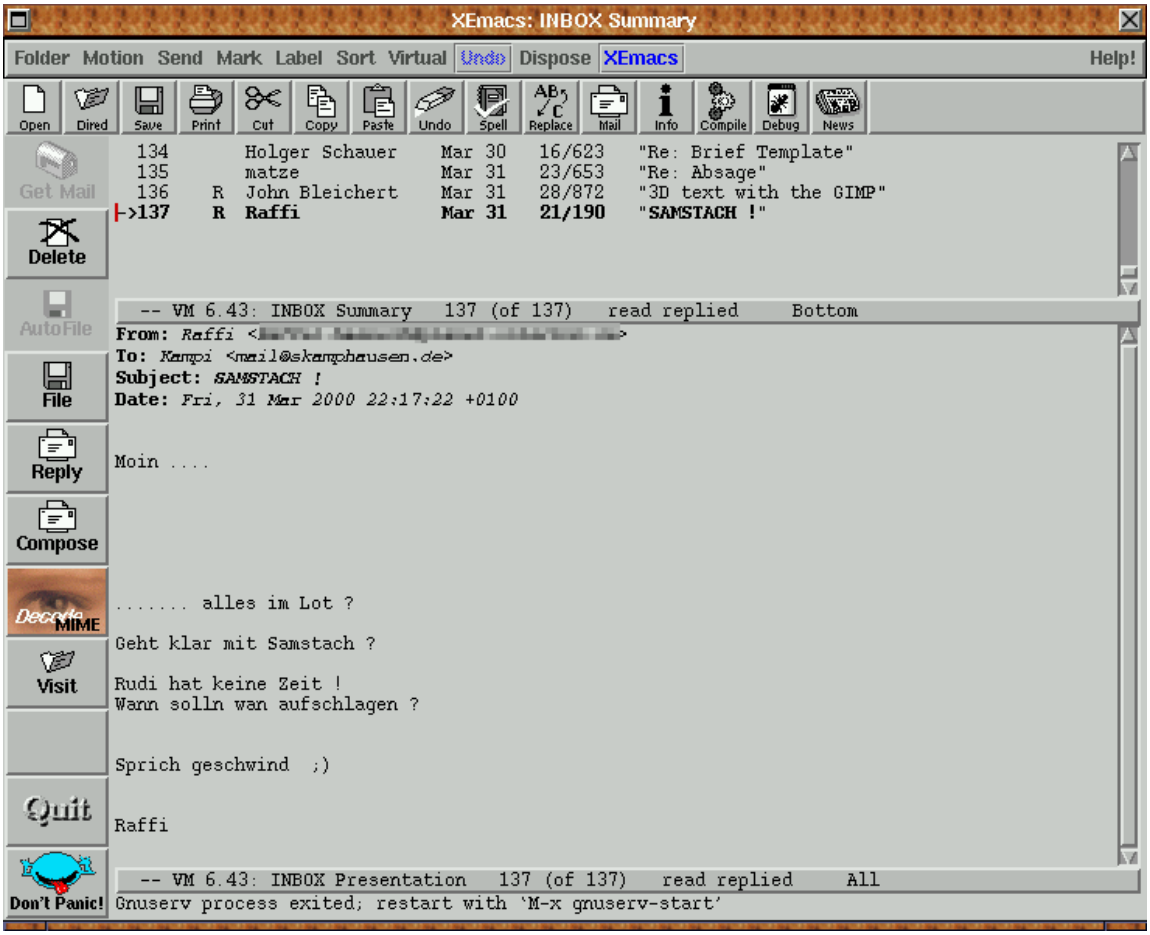


Abbildung 3.4.: Ein VM-Fenster wie es bei mir vor vielen Jahren mal aussah.

in einem dedizierten XEmacs-Prozess, den man sich bei heutiger Rechen- und Speicherleistung sicherlich leisten kann. Ansonsten wird er einfach im XEmacs durch

```
M-x vm
```

gestartet oder aber in der Toolbar durch einen Klick auf das Mail-Icon, falls jenes auf das Starten von VM gesetzt wurde (vermöge `(setq toolbar-mail-reader 'vm)`, oder aber unter Options-> Customize-> Emacs-> Environment-> Toolbar-> Mailreader). Dann öffnet sich ein Fenster wie in Abbildung 3.4 zu sehen.

Jetzt müssen wir als erstes mal an unsere Mail rankommen. Es mag sein, dass der Systemadministrator des lokalen Netzes bereits dafür gesorgt hat, dass all unsere Mail in `/var/spool/mail/[USERNAME]` zu liegen kommt, vielleicht besorgen wir das auch schon selber mittels `fetchmail`. Dann ist das all kein Problem mehr, weil es bereits automatisch funktioniert. Vielleicht haben wir aber auch irgendwo einen POP3-Account, von dem die Mail geholt werden soll. Dann heißt es erste Konfigurationen am VM vorzunehmen. Dieses kann in der `$HOME/.xemacs/init.el` geschehen, wovon aber abzuraten ist, denn VM hat auch seine eigene Initialisierungsdatei: `/.vm`. Diese Datei sollte mit der Zeile:

```
;; -*- Mode: Emacs-Lisp -*-
```

beginnen, damit XEmacs weiß, wie er mit der Datei umzugehen hat, weil sie ja nicht die Endung `.el` aufweist. Jetzt zeigen wir VM den Weg zu unserer Mail:

Abbildung 3.5.: Die Toolbar des Mailprogramms VM. Die Beschriftungen sollten soweit eigentlich selbsterklärend sein. Die einzelnen Funktionen werden in Tabelle 3.3 näher erläutert.

```
(setq vm-spool-files
      (list
       (concat "/var/spool/mail/"
               (getenv "USER"))
       (concat "pop.server1.net:110:pass:"
               (getenv "USER") "daspasswort")
       "pop.server2.org:110:pass:username2:*"))
```

Dieses Beispiel lässt VM erstens die lokale Mail abholen und zweitens bereits zwei verschiedene POP Mailboxen abfragen. Die entsprechenden Server sind natürlich noch einzustellen... Wer es nicht mag, dass das POP Passwort im Klartext lesbar im Heimatverzeichnis rumliegt [6], trägt hier einfach ein `*` wie beim zweiten Server im Beispiel ein. Dann fragt VM nach dem Passwort beim ersten Abholen und merkt es sich für den weiteren Verlauf der Sitzung.

Da wir gerade ein wenig konfigurieren, gleich noch ein paar Kleinigkeiten eingestellt:

```
(setq vm-mail-check-interval 300) ; schau nach post alle 5min
```

Das erklärt sich wohl von selber, oder? Die Zeitangabe erfolgt in Sekunden. Falls man nicht wünscht, dass VM nach Mail schaut (z. B. weil das bereits das kleine Tool im Panel tut), schafft

```
(setq vm-mail-check-interval nil)  ; nicht nachschauen
```

Abhilfe. Das ist vor allem dann sinnvoll, wenn man nur an einer Wählverbindung sitzt[7].

Soweit so gut; und wie oft soll VM die Mail dann auch abholen?

```
(setq vm-auto-get-new-mail 900) ; post alle 15mins abholen
```

Bleibt noch die Frage, wo die Mail gespeichert werden soll. Es ist sicherlich Geschmackssache, wie man dazu steht, wenn Programme selbsttätig irgendws im eigenen Heimatverzeichnis erstellen, aber Gerümpel liegt dort eigentlich schon genug herum, so dass man sein Mail-Verzeichnis sicherlich lieber unterordnet:

```
(setq vm-primary-inbox "~/Mail/INBOX") ; standard: ~/INBOX
```

3.13.3. Mail lesen

Gut, jetzt wird die Mail gelesen. Die Navigation durch die `Inbox Summary` (so heißt der Buffer, der alle Mails in der Inbox auflistet) ist ziemlich intuitiv (wenn man die XEmacs Standards kennt ;-): Scrollen mit der Scrollbar oder den Cursortasten, anwählen mit der mittleren Maustaste oder (RETURN) oder (SPACE), nochmal (SPACE) für das Weiterblättern im angezeigten Text der Mail (Buffername: `INBOX`). Zum Auslösen der diversen Aktionen kann man einerseits die Toolbar benutzen, anderseits aber auch die Tastaturkürzel oder das Menu auf der rechten Maustaste. Tabelle 3.3 und Abbildung 3.5 beschreiben eine kleine Sammlung diverser Aktionen und wie sie erreicht werden können.

[6] vielleicht auch mal Gedanken um die Leserechte anderer an der eigenen `.vm` machen...
[7] Solche Not leidet selbst im Jahre 2004 noch der Autor dieser Zeilen

Get Mail (g)	Holt die Mail aus allen Mailboxen, die in `vm-spool-files` angegeben sind.
Delete/Undelete (d)/(u)	Markiert Nachrichten als gelöscht bzw. hebt diese Markierung wieder auf. Die Nachrichten werden jedoch nicht sofort gelöscht; das geschieht erst durch Aufruf von `vm-expunge-folder`, was auf dreimal (#) liegt, um versehentliches Löschen zu vermeiden.
Autofile	Autofile (s. u., sehr praktisch!)
Speichern (s)	Speichert die aktuelle Nachricht auf Festplatte. Der erscheinende Dialog ist nicht schwierig zu verstehen. Falls man eine neue Datei einrichten möchte, klickt man einfach auf „Click here for Keyboard Interface“, falls man vorher mit der Maus unterwegs war. Speichert man ab, indem man (s) drückt, ist man von vorneherein im Keyboardmodus und kann einfach einen neuen Namen angeben (hier wird der auto-file-Eintrag, sofern vorhanden, oder der zuletzt eingegebene Wert als Default angeboten).
Antworten (r)/(R)	Öffnet einen Buffer, in dem man eine Antwort Mail auf die aktuelle verfassen kann. Dabei geht der Reply Button davon aus, dass man den Text der Mail als Zitat einfügen will, während man mit der Tastatur die Wahl zwischen (r): ohne und (R): mit Zitattext hat.
Mail schreiben (m)	Öffnet einen Buffer, in dem man eine neue Mail verfassen kann. Dabei stehen einem natürliche alle Funktionen eines ausgewachsenen XEmacs offen. Ich erwähne das, weil das seinerzeit für mich der Grund war auf VM umzusteigen.
MIME interpretieren (D)	VM kann diverse MIME Typen (wie zB jpg's) gleich mitten im Text anzeigen, oder aber einen kleinen Balken einblenden, der anzeigt, dass da ein MIME Objekt sitzt. Mit diesem Knopf kann man so durch die Anzeigemöglichkeiten für MIME blättern. Einfach testen...
Archive besuchen (v)	Will man alte mit dem File-Kommando abgespeicherte Mails nochmal lesen, wird man sie mit *Visit* besuchen. Jedes so besuchte File sieht aus wie eine eigene INBOX und lässt sich mit (g) einfach wieder beenden (ggf, muss man danach wieder in den Inbox-Summary-Buffer wechseln).
Quit/Help (q)	Der Quit Button (Tastatur: (q)) erklärt sich wohl von selbst. Falls nicht: Don't Panic ... Hilfe wartet schon :-)

Tabelle 3.3.: VM Aktionen

3.13.4. Toolbar für Fortgeschrittene

Sicherlich wird bei vielen die Toolbar etwas andere Buttons enthalten, als das bei meinem VM der Fall ist. Die Lösung ist einfach: Folgender Code in der /.vm stellt die Toolbar ordentlich ein:

;; Configure the toolbar: (setq vm-use-toolbar '(getmail delete/undelete autofile file reply compose mime visit nil quit help))

Die Beschreibung von vm-use-toolbar ist sehr aufschlussreich. Wichtig: das Symbol nil darf nur einmal auftauchen: Alle Buttons vorher werden linksbündig bzw. am oberen Rand der Toolbar ausgerichtet, alle danach rechtsbündig bzw. am unteren Rand, je nach Orientierung der Toolbar. Es mag sein, dass für das Abholen von Mail noch die Zeile (fset 'vm-toolbar-getmail-command 'vm-get-new-mail) in die .vm einzutragen ist.

Mittels

```
(setq vm-toolbar-pixmap-directory "/pfad/zu/den/vm/pixmaps/")
```

kann man ein anderes Verzeichnis als das Standardverzeichnis (`/xemacs/installations/dir/etc/vm/`) für die Bilder (nicht nur) in der Toolbar wählen, auch die Pixmaps für die MIME Typen sind dort abgelegt. Ich habe

dafür eigene Pixmaps erstellt, die dem VM ein Aussehen, wie im abgebildeten Screenshot verpassen. Die geneigte Systemadministratorin kann sie natürlich auch systemweit installieren.

3.13.5. Autofile

Der Autofile Mechanismus ist irre praktisch! Man kann Muster für jeden Mail-Header angeben und danach eine Standard-Datei angeben, in die man diese Nachrichten speichern kann. Das Einrichten dieser Muster sieht so aus:

```
(setq vm-auto-folder-alist
      '(("from"
    (".*kamphaus.*" . "Perso/own")
    (".*chef.*" . "Wichtig/el-chefe")
    )
   ("to"
    (".*xcf.*" . "Listen/gimp")
    (".*emacs.*" . "Listen/xemacs")
    )))
```

Hier werden zwei Muster für den Absender (from) angegeben. .* meint eine beliebig lange Zeichenkette (auch null) aus beliebigen Zeichen. Zwei weitere Muster für den Adressaten (to) sorgen dafür, dass sich die bemerkenswerten Nachrichten auf der Gimp- und der XEmacs-Mailingliste mit einem einfachen Mausklick abspeichern lassen. Da Wer meist mit der Tastatur navigiert, braucht diese Funktion vielleicht auf dem einfachen a:

```
(local-set-key "a" 'vm-toolbar-autofile-message)
```

3.13.6. Verschiedenes

Weitere Einstellungen entnehme die interessierte Leserin bitte meinem Beispiel für eine vm-init.el

(M-s) durchsucht übrigens alle Nachrichten in einem Folder ganz so, wie man es von incremental-search kennt.

Das Rechte-Maus-Menu ist auf einem MIME Objekt anders als sonst: hier kann man z. B. abspeichern.

Es gibt noch einen großen Bruder: Gnus ist sowohl Mail- als auch Newsreader mit einer unglaublichen Anzahl von Features.

Die Tasten (f) (bzw. (F)) agieren wie (r) (bzw. (R)), senden aber an alle Empfänger.

3.14. Gnus

3.14.1. Einleitende Worte

OK, längere Zeit stand an dieser Stelle der lapidare Hinweis, dass ich keine Ahnung von Gnus habe, und dieses Kapitel nur geschrieben werden würde, wenn es andere schreiben oder ich dereinst auf Gnus umsteigen würde. Natürlich ist der zuvorderst beschriebene Fall nicht eingetreten. Der zweite hingegen mittlerweile doch.

Daraus ergibt sich aber auch gleich ein Warnhinweis: in Sachen Gnus bin ich sicherlich noch ein Anfänger, auch wenn ich so einiges inzwischen für mich gelöst habe.

Was ist Gnus? Gnus ist ein XEmacs-Paket zum Lesen von Usenetnews, ein Newsreader. In einem zweiten Schritt transferiert Gnus das Konzept der Newsgruppen auf das Lesen ganz normaler EMails. Das ist konzeptionell ein himmelweiter Unterschied zu allen (?) anderen EMailprogrammen, die eingehende Post einsammeln und in einem oder mehreren Postfächern darstellen! Zudem gilt Gnus als sehr standardkonform, kann Gnus vielerlei bereinigen, was andere verbocken und ist, naturgemäß, da in ELisp geschrieben, grenzenlos konfigurierbar.

Ich möchte für dieses Kapitel noch einige Einschränkungen geben:

- Es ist lange noch nicht fertig.
- Die Beispiele und Konfigurationen sind noch nicht lange genug im harten Alltagseinsatz getestet.

- Ich bin ein Gnus-Neuling, vielleicht mache ich hier auch echten Quatsch.
- Es wird nie eine vollständige Doku von Gnus werden, das wäre echt zu lang, aber es zeigt *einen* Weg durch den Dschungel.

3.14.2. Einrichtung

Wer sich auf die Suche nach Dokumentationen zu Gnus begibt, wird immer wieder beschrieben finden, wie man Gnus zuerst zum Newslesen einrichtet und dann die EMailfunktion nachlegt. Dazu sei an dieser prominenten Stelle erwähnt, dass man auch EMail als primäre Funktion wählen kann, ja sogar gänzlich auf News verzichten kann. Die primären Verbindungen, die mit der Variablen `gnus-select-method` meist auf News eingerichtet werden, können ebensogut mit den meist unter `gnus-secondary-select-methods` eingerichteten Mailverbindungen belegt werden. Die sekundären Verbindungen sind eine Mischung aus der primären sowie *fremden* (foreign), verhalten sich aber –so sagt die Dokumentation– im Wesentlichen so wie die primären.

News

Nachdem dieses gesagt ist, kann auch ich einschwenken auf den üblichen Pfad und zunächst den Newsgruppenzugriff konfigurieren:

```
(setq gnus-select-method '(nntp "news.newsserver.example"))
(add-hook 'nntp-server-opened-hook 'nntp-send-authinfo)
```

Dabei sorgt die zweite Zeile dafür, dass bereits ganz zu Beginn die Authentifizierung am Newsserver erfolgt, was bei solchen Servern, die bereits für das Lesen eine Authentifizierung verlangen, erforderlich ist. Bleibt die Frage, woher kennt Gnus meinen Usernamen und Passwort? Die stehen in der Datei `/.authinfo`[8], deren Format sich wie folgt darstellt:

```
machine news.newsserver.example login meinusername password sehrgeheim
machine mein.imapserver.example login ichbinimap password weisskeiner
```

Dabei können noch einige weitere Schlüsselwörter verwendet werden. Im Falle einer Newsserververbindung kann man manchmal noch `force yes` in einer Zeile gebrauchen, eine IMAP-Verbindung wird oft nur durch Angabe von `port imap` automatisch von Gnus detektiert.

Das Funktionieren der Authentifizierung merkt man leicht daran, ob man nach Benutzername und Passwort gefragt wird, oder eben nicht.

Eine häufige Fehlerquelle sind falsche Benutzerrechte: diese Datei darf **nur** für den Benutzer selber lesbar sein:

```
shell> chmod 600 ~/.authinfo
```

Mail

Für unser EMail-Setup verwenden wir mal ein etwas ausführlicheres Beispiel:

- Es gibt einen IMAP-Server (ein weit verbreitetes, kommerzielles Produkt) im lokalen Netz.
- Dort existiert ein Postfach für uns, dessen Inhalt wir aber gerne auf unsere eigene Platte bringen wollen (auch um den Server ein wenig zu entlasten).
- Für eine Arbeitsgruppe liegt dort ein weiteres Postfach (kein öffentlicher Ordner, sondern ein eigenes Postfach), auf das wir gerne zugreifen möchten. Dort wollen wir aber keine Mails löschen oder ähnliches.
- Über das Hauptpostfach empfangen wir auch unsere private EMail, die von all unseren privaten Accounts weitergeleitet wird.

[8] Genauer gesagt in der Datei, die in `nntp-authinfo-file` beschrieben ist.

Natürlich muss am Ende auch noch die Jonglage mit den verschiedenen Identitäten funktionieren, aber das schieben wir mal (3.14.4).

Für die Hauptmailbox verwenden wir das (empfohlene) nnml-Mailbackend. Es gibt noch andere, aber nnml scheint mir das angesagte zu sein: es speichert eine Mail pro Datei, so dass große *Gruppen* keine Performance kosten (und die Suche mit `grep` gut funktioniert). Dazu konfigurieren wir:

```
(setq gnus-secondary-select-methods
      '((nnml "")))
```

und definieren die Mailbox mit einer weiteren Variablen:

```
;; IMAP als POP++
(eval-after-load "mail-source"
  '(add-to-list 'mail-sources
                '(imap
                  :server "imapserver"
                  :user "username"
                  :password "mypassword"
                  :fetchflag "\\Seen" ;; don't delete
                  )))
```

Die Zeile mit dem `fetchflag` bedeutet, dass die Mail auf dem Server belassen wird. Das ist am Anfang praktisch, wenn man Gnus im Parallelbetrieb zu seinem bisherigen Mailprogramm laufen lässt, später kommt das natürlich weg. Dieser Modus von IMAP wird in der Doku zu Gnus als *POP++* bezeichnet, weil es eigentlich nur ein fortgeschrittenes Abholen von EMail ist, ohne auf die ganzen Funktionen von IMAP zuzugreifen. Die eingehende Post werden wir gleich von Gnus untersuchen lassen und Gnus wird uns dafür verschiedene *Gruppen* anlegen (siehe Abschnitt 3.14.3). Es wurde ja eingangs bereits darauf hingewiesen, dass Gnus auch EMaillesen wie Newslesen behandelt. Eine Mailbox ist also nur eine Newsgruppe, vielleicht mit anderen Eigenschaften, aber die kann man auch für Newsgruppen einzeln vergeben. Das dahinterliegende Backend spielt keine Rolle mehr.

Wenden wir uns nun dem Zugriff auf das andere Postfach zu. Das hierfür gewählte Backend ist *nnimap* und es konfiguriert sich so:

```
(setq gnus-secondary-select-methods
      '((nnml "")
        (nnimap "imap-server"
                ;; somehow these settings don't show up?
                (nnimap-address "imap-server")
                (nnimap-authinfo-file "~/.imap-otherbox")
                ;; hopefully these do...
                (nnimap-list-pattern ("INBOX.*"))
                (nnimap-expunge-on-close never)
                )
        ))
```

Man kann an den Kommentaren bereits erkennen, dass bei mir noch nicht alles ganz so funktioniert, wie es soll ...

Auch hier taucht wieder ein `authinfo`-File auf und dort wartet auch ein wichtiger „Trick“. Zur Anwahl des anderen Postfachs muss der Benutzername sehr ausführlich angegeben werden:

```
machine mein.imapserver.example login "domaene/username/postfach" password egal
```

Dabei zeigt meine Erfahrung[9], dass für den Postfachnamen der *Kurzname* des Serversystems gewählt werden sollte. Ich hatte einen Fall vorliegen, in dem der ausführliche Name der Form „Unsere Gruppe [Ich&Du]“ folgte. Das hat nicht funktioniert. Der Kurzname „unseregruppe“ hingegen führte zum gewünschten Ergebnis.

Im Prinzip lassen sich so und durch die Angabe von `nnimap-authinfo-file` beliebig viele Postfächer abfragen, bei mir persönlich klappt das noch nicht, was aber sicher an mir liegt.

[9] ohne tieferes Verständnis der Materie

In den Kapiteln zur Mailverteilung (3.14.3) und zur Klärung der Identität (3.14.4) werden wir dafür sorgen, dass die verschiedenen (privaten) EMailadressen ordentlich behandelt werden.

Es ist zu beachten, dass man die zugänglichen Gruppen per nnimap nicht sofort angezeigt bekommt. Um die wichtigen auszuwählen, geht man zunächst in die Liste der Server (^) und wählt dort mit (RET) den IMAP-Server aus. Es dauert dann eine Weile, bis schließlich eine Liste der verfügbaren Postfächer angezeigt wird. Diese Liste kann durch öffentliche Order sehr lang werden, man kann aber die Liste auch filtern:

```
(nnimap-list-pattern ("INBOX.*"))
```

Dort kann man dann mit (u) beliebig viele Gruppen auswählen, die dann später auch angezeigt werden.

3.14.3. Mailverteilung

Eingehende Mail soll gleich in passende Gruppen verteilt werden. Dazu bietet das nnmail-Backend zwei Mechanismen: einen einfachen, der hier nicht weiter besprochen wird, sowie einen komplexeren, *fancy* genannt. Um zweiteren auszuwählen, brauchen wir:

```
(setq nnmail-split-methods 'nnmail-split-fancy)
```

Häufig begegnet man in den Beispielen auch der Zeile

```
(setq nnmail-crosspost nil)
```

Das soll verhindern, dass beim Mailsplitten mehrere Newsgroups befüllt werden (entsprechend kann es mit `t` erlaubt werden). Im Falle des Fancy-Modes können wir das aber auch gezielt in der Beschreibung der Splitregeln tun.

Betrachten wir einige Regeln:

```
(setq nnmail-split-fancy
      '(&
        (from "mail@agoodfriend\\.example" "mail.private.thegoodfriend")
        (subject ".*[SPAM].*" "mail.spam")
        (subject ".*[cvs.host]" "mail.code.cvs")
        (to "mail@meinprivate\\.example" "mail.private.misc")
        (from ".* <?\\(.+\\)@theoffice.example" "mail.office.\\1")
        (to "me@theoffice\\.example" "mail.office.all")
        "mail.inbox"
        ))
```

Das erste Element dieser Liste ist ein &, damit wird gesagt, dass von den folgenden beliebig viele Regeln zutreffen dürfen. Ein | würde den Verteilprozess abbrechen, sobald eine Regel getroffen hat. Ich möchte halt noch eine old-style Mailbox in `mail.inbox` haben, wo letztlich alle Mails landen.

Wie man sieht kann verschiedene Regeln definieren:

- Einen einfachen String (hier „mail.inbox“). Dort wird eine Mail einsortiert.
- Ein Regel der Form `HEADER MATCH GRUPPE`. Dort wird der Inhalt des EMail-Headers `HEADER` mit dem regulären Ausdruck `MATCH` geprüft. Trifft der `MATCH` zu, wird die Mail (auch) in die Gruppe `GRUPPE` einsortiert.
- Der Beschreiber `GRUPPE` kann auf die Ergebnisse des regulären Ausdrucks zugreifen: wenn man mit runden Klammern eine Gruppe im regulären Ausdruck definiert, liegt deren Inhalt später in `\\1` vor. Weitere Klammergruppen werden dann hochgezählt. In obigem Beispiel benutze ich das, um den Benutzernamen meiner Arbeitskollegen aus dem Absendermailheader auszuschneiden und eine entsprechende Gruppe anzulegen.

Gerade bei Leuten, die ihre Mails mit Outlook schreiben, erhält man oft verschiedene Großkleinschreibweisen. Beispielsweise kann Hans Benutzer den Benutzernamen „hbenutzer“ senden oder auch „HBenutzer“. Damit dann nicht für jede Schreibweise eine neue Gruppe angelegt wird, setzt man

```
(setq nnmail-split-lowercase-expanded t)
```

Danach ist alles klein geschrieben.

3.14.4. Identitätenwechselei

Gnus ist in der Lage durch Befüllen der entsprechenden Mailheader verschiedene Identitäten vorzugeben. So kann man als Privatmann mit verschiedenen EMailadressen hantieren sowie als Arbeitstier aus verschiedenen Teams heraus Mails beantworten. Ich verwende hier eine Lösung, die oft in der Newsgruppe `gnu.emacs.gnus` zitiert wird. Eine Whitelist gibt die Liste der verfügbaren Adressen an, ein weitere Funktion schaut in die zu beantwortende Mail (so man denn gerade beantwortet) und die Variable `gnus-posting-styles` erledigt den Rest:

```
(setq my-mail-addresses
      (list "meine@newsgruppenadresse.example"
            "haupt@privateadresse.example"
            "team@beiderarbeit.example"))

(defun ska-message-get-from-address ()
  (save-excursion
    (if (and (boundp 'gnus-article-buffer)
             (gnus-buffer-live-p gnus-article-buffer))
        (progn
          (set-buffer gnus-article-buffer)
          ;; fixme: this doesn't work when I'm cc'ed
          (let ((address (message-fetch-field "to")))
            (if (and address
                     (string-match (regexp-opt ska-mail-addresses)
                                   address))
                (let ((matched-address (match-string 0 address)))
                  (cond
                   ((string-match "otheridentity" matched-address)
                    (concat "Iam Someoneelse [At The Company] <" matched-address
                            ">"))
                   (t
                    (concat "Mein Name <" matched-address ">"))))
              "Mein Name <ich@beiderarbeit.example>")))
      "Mein Name <ich@beiderarbeit.example>")))

(setq gnus-posting-styles
      '((".*"  ;;default
         (name "Mein Name")
         (organization "The Company")
         (signature-file "~/.signature")
         (address "ich@beiderarbeit.example")
         ;; for debugging:(eval (message "NEWS %s" (message-news-p)))
         )
        ((message-news-p)  ;;Usenet news?
         (organization "")
         (signature-file "~/.signature-priv")
         (address "meine@newsgruppenadresse.example")
         )
        ("mail\\.private.*"
         (organization "")
         (address ''haupt@privateadresse.example")
         (signature-file "~/.signature-priv")
         (message-user-fqdn "mail.privateadresse.example")
         )
        ("nnimap\\+imap-server.*"
         (name "Das Team [Bei Der Arbeit]")
         (organization "The Company")
         (address "team@beiderarbeit.example")
         )
        ("nnml:.*" ;; dynamically detect address using functions above
         (From (ska-message-get-from-address))
         )
        ))
```

Dort werden die Gruppen, in denen man gerade seine Artikel (Mail oder News) schreiben will, getestet. Der erste Ausdruck definiert einen Default, der zweite testet nicht gegen den Namen, sondern ob es sich um einen Newsartikel handelt, der dritte Ausdruck testet, ob eine Mail im privaten Ordner beantwortet wird. Der vierte Ausdruck bedient die nnimap-Gruppen, die wir ja für eine andere Arbeitsgruppe im Büro ausgewählt hatten. Die letzte Gruppe nun setzt noch den `From`-Header dynamisch in Abhängigkeit davon, unter welcher Adresse uns diese Mail erreicht hat.

3.14.5. Lesen

- Gruppen-Buffer
- Topics
- Artikelbuffer
- MIME siehe display...
- Screenshots!

3.14.6. Schreiben

- Reply vs Followup
- MIME/MML erzeugen
- ?

3.14.7. Archivieren

- Outbox

3.14.8. Cryptokram

- PGG
- Mailcrypt
- SMIME
- MML

3.14.9. Anzeige

- Artikel/MIME
- Gruppen/Summary Zeilen
- `gnus-add-configuration`

3.15. EShell

Mit der eshell gibt es eine vollkommen in ELisp geschriebene Shell. Aufgrund der mangelnden Erfahrungen mit derselben des Autors kann dieses Kapitel derzeit nur rudimentär ausfallen. Doch ist es einfach angenehm, wenn man sich mit wenigen Tastendrücken schnell in einer Shell im aktuellen Arbeitsverzeichnis befinden kann. Zumal diese Shell dann, da sie in einem XEmacsbuffer abläuft, den Vorteil hat, dass man Textabschnitte von dort ohnen den Umweg über das XClipboard in einen anderen Buffer kopieren kann. Aufzurufen ist die EShell ganz einfach via (M-x) `eshell`. Sinnvoll mag auch ein Tastaturkürzel der folgenden Art sein:

```
(global-set-key '(f3) '(lambda ()
                                (interactive)
                                (eshell)
                                (insert "ls")
                                (eshell-send-input)))
```

Interessant ist noch die Variable `eshell-visual-commands`, die all jene Befehle in einer Liste versammelt, die in einem Terminal etwas mehr machen, als nur auf die Standardausgabe zu schreiben. Für diese bemüht EShell dann eine eigene Terminalemulation des XEmacs (s. Abschnitt 3.16).

3.16. M-x term . . . ls . . . kaputt

Ok, diese Nummer hat mich lange Jahre verfolgt und ich habe oft versucht, sie zu bereinigen. Ich will eines vorweg nehmen: die Geschichte hat ein Happy End.

Betrachten wir gemeinsam Abbildung 3.6. Im unteren Bereich findet sich eine Fehlermeldung aus den Tiefen des Codes im oberen Bereich eine recht wirre Anzeige. Dort wurde folgendes nacheinander ausgeführt:

1. (M-x) `term`
2. `/bin/bash` (RET)
3. `cd <PFAD/ZU/DIESER/DOKU RET`
4. `ls RET`

Na, kann man in der Anzeige mit etwas gutem Willen sogar noch nachvollziehen, was uns die Shell eigentlich sagen wollte. Wirklich, über Jahre hinweg habe ich immer mal wieder versucht, diesem Problem beizukommen. Ohne Erfolg. Auch neuere XEmacsen brachten nie eine Lösung.

Bis Kommissar Zufall um die Ecke kam. Es begab sich, dass ich Ende 2004 auf ein Gentoo-System umstieg, und siehe da! dort lief die Terminalemulation[10].

Wenige Tage später bekam ich eine Datei von einem anderen Betriebssytem, das seine Zeilenenden immer noch mit einem Wagenrücklauf (
`r` oder auch `^M`) versieht. Und zum ersten Mal in vielen Jahren der XEmacsbenutzung sah ich mich diesen Zeilenenden gegenüber, um die ich mich noch nie kümmern musste. Eine entsprechende Suche in den Newgroups ergab erstaunliches: MULE, die Multilanguage Extension des XEmacs war bisher dafür verantwortlich und ich hatte einen XEmacs ohne MULE. Also habe ich XEmacs fix neu ge-emerge-t, diesmal mit MULE und einige weitere Tage später fiel mir auf, dass das Terminal nicht mehr ging.

Mit diesem neuen Wissen um die Korrelation dieser beiden Dinge ging ich erneut auf die Suche in den Newgroups und siehe da, es fand sich eine Lösung. MULE erkennt auch im Falle eines Terminalmodes automatisch irgendwelche Sachen und reagiert schlicht falsch. Die einfache Lösung dieses alten Problems besteht also darin, im Falle einer Terminalemulation explizit den Modus des Buffers zu setzen:

[10] Ich setze hier ganz frech voraus, dass die Leser wissen, was eine *Terminalemulation* ist. Wenn nicht: das ist das Fenster, in dem Eure Shell abläuft, mal ein `xterm`, mal eine `konsole` oder ein `gnome-terminal`, um nur einige zu nennen.

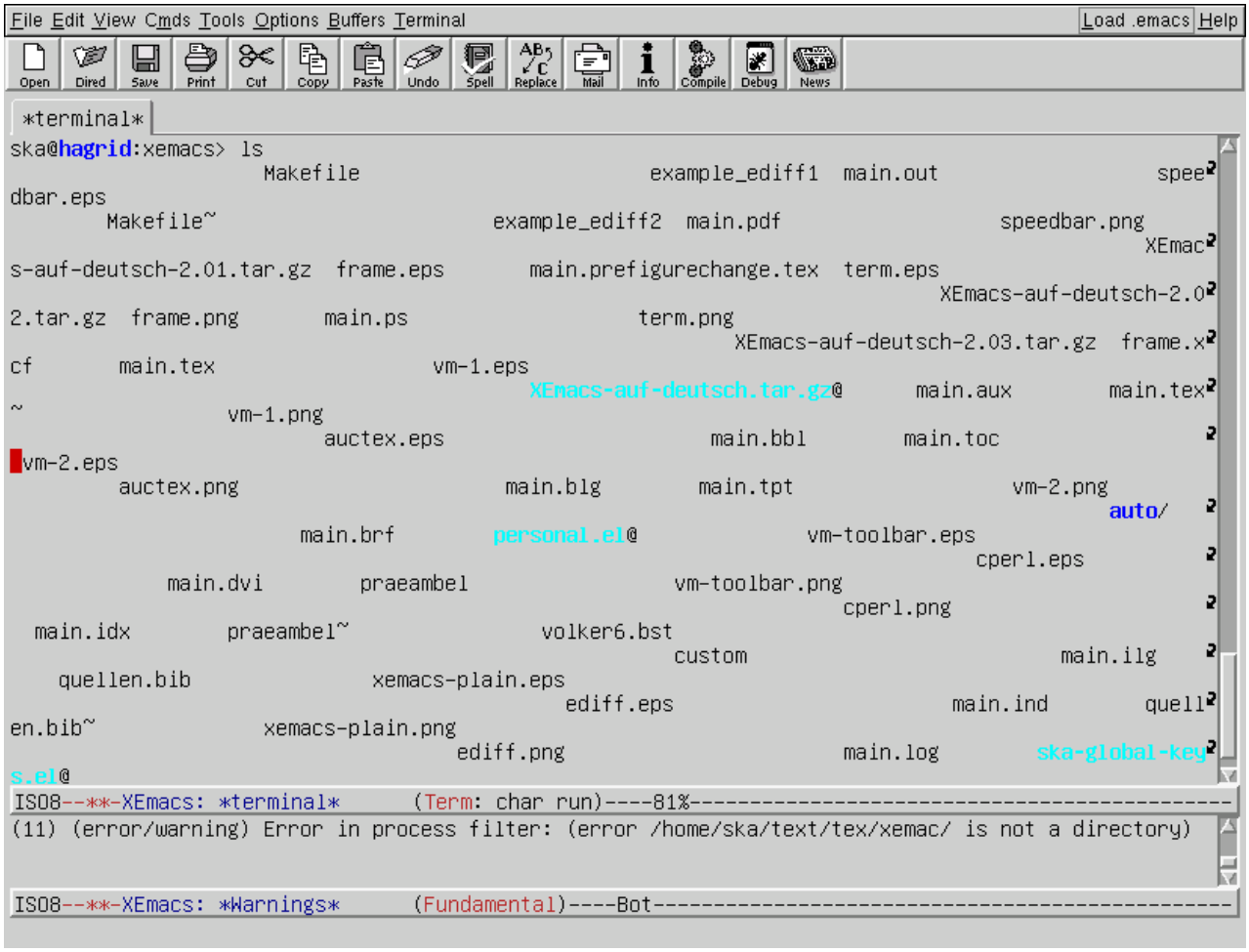


Abbildung 3.6.: Ganz schön kaputt die Terminalemulation im XEmacs. Verflixung!

```
(defun bug-evil-term-process-coding-system ()
  "Fix a term bug; the 'process-coding-system' should always be 'binary'."
  (set-buffer-process-coding-system 'binary 'binary))

(add-hook 'term-exec-hook 'bug-evil-term-process-coding-system)
```

Ist das nicht schön?

4. Konfiguration

Hier geht es in weiten Teilen um echten ELisp-Code. Es wird aber aber an dieser Stelle auf die syntaktischen Erklärungen des Codes verzichtet und stattdessen auf das Kapitel 8 verwiesen.

4.1. Customize

Der Customize-Mechanismus des XEmacs ist ein mächtiges Werkzeug zur Konfiguration. Bei einem Programm, das die Größe des XEmacs erreicht hat, reichen eine einfache ASCII-Datei und ein paar Menüpunkte einfach nicht mehr aus.

4.1.1. Bedienung

Der (zunächst) leichteste Weg zu Customize ist über das Menü Options->Customize->Emacs. Dort lassen sich entweder einzelne Punkte oder ganze Gruppen anwählen. In jedem Fall erhält man einen eigenen Buffer, in dem man sich die Dokumentation des angewählten Punktes und den aktuellen Wert ansehen kann. Oftmals werden einzelne Einstellungen angeboten, immer kann man sie erstmal nur in dieser aktiven Instanz des XEmacs testen (Set), oder sie gleich in der entsprechenden Datei speichern (Save for future sessions). Gespeichert wird das Ganze in der Datei, deren Name sich in der Variabeln `custom-file` wiederfindet.

Das Anklicken von Buttons und das Durchsuchen von Menus muss wohl nicht explizit dokumentiert werden. Es soll für den Moment genügen, darauf hinzuweisen, dass vermöge

```
M-x customize-apropos RET [muster] RET
```

Optionen konfiguriert werden können, deren Name auf das angegebene Muster (Regulärer Ausdruck!) passt und dass man mittels (M-x) `customize` (TAB) weitere Varianten des customize-Befehls finden kann, um z. B. einzelne Variablen ((M-x) `customize-variable`) oder ganze Gruppen ((M-x) `customize-group`) zu konfigurieren.

Es soll davon ausgegangen werden, dass die Bedienung von Customize im Wesentlichen selbsterklärend sein sollte. Es ist neben der eigentlichen Funktion, die Konfiguration des XEmacs zu bewerkstelligen auch eine super Fundgrube für neue Sachen und ein Ort zum Stöbern. ...

4.2. Eigene Tastaturbelegungen

Wenn ich auf ein fremdes System komme und dort einen XEmacs bemühe ist es oftmal ein großes Problem für mich, weil dort einfach diverse Tastaturbelegungen nicht so sind, wie ich sie kenne. Das geht sicherlich vielen Leuten so. [1] Immer mal wieder begegnet man einer Funktion, die man schnell verfügbar haben möchte. Und weil es nur begrenzt viele F-Tasten gibt, habe ich persönlich zu den bereits vorhandenen Präfixen (C-x) und (C-c) (für globale beziehungsweise lokale (modespezifische) Funktionen) noch einmal zwei solche Präfixe spendiert: die gleich daneben liegenden (C-v) und (C-b)[2]. Diese habe ich ebenfalls mit globalen respektive lokalen Funktionen belegt.

Aber genug der Motivation und Geschichtchen. Zum Thema gibt es gar nicht so viel zu sagen.

[1] Es gibt auch echte Extremfälle: ich habe es erlebt, dass jemand seine Tastaturbelegungen so umgebaut hat, dass sie wie eine uralte, kommerzielle C-Entwicklungsumgebung agieren.

[2] Das ist natürlich völlig gegen den Standard und alle Empfehlungen und ich kenne auch niemanden sonst, der solches von sich aus freiwillig getan hätte. Ich hingegen bin sehr glücklich damit.

Einerseits gibt es viele Varianten, wie man Tastenbelegungen setzen kann, aber nur eine, die als empfehlenswert gilt. Davon zeigt Abschnitt A.3 eine ganze Menge. Hier nur ein Beispiel, an dem wir auch gleich noch die zweite erwähnenswerte Sache an eigenen Tastenbelegungen sehen können:

```
(global-set-key '[(control up)] '(lambda ()
                                   (interactive)
                                   (scroll-down 1)))
```

Mal abgesehen davon, dass ich es als enorm praktisch empfinde, dass ich mit gehaltener Control-Taste und den Hochundruntertasten scrollen kann, sehen wir hier die angesagte Syntax, wie man beim Tastendruck einer Funktion ein Argument übergibt: man verwende `(lambda)`. Folgendes ginge halt *nicht*:

```
(global-set-key '[(control up)] (scroll-down 1))  ;; FALSCH!
```

Was hier passieren würde, ist dass, beim *Setzen* der Tastenbelegung einmal gescrollt werden würde. Wir brauchen aber eine Funktion, die beim Tastendruck aufgerufen werden soll. Eine solche Funktion könnte man nun mit einem Namen versehen und dann an die entsprechende Taste binden:

```
(defun my-scroll-one-line-down ()
 "Scroll just one line down."
 (interactive)
 (scroll-down 1))

(global-set-key '[(control up)] 'my-scroll-one-line-down)
```

Aber das ist ja noch länger als die Version mit `lambda` und erzeugt zudem noch eine neue Funktion, die wir aber sonst gar nicht weiter brauchen. Ergo: man nehme `lambda`.

4.3. X-Ressources

Muss ich noch was zu schreiben. Momentan nur Code:

```
Emacs*background: #cecece
Emacs*EmacsFrame.geometry: 100x61+0+0
! from http://www.emacswiki.org/cgi-bin/wiki.pl?EmacsNiftyTricks
Emacs*XlwMenu*fontSet: -*-helvetica-medium-r-normal-*-*-120-*-*-*-*-*-*
Emacs*XlwMenu*foreground: black
Emacs*XlwMenu*shadowThickness: 1
```

4.4. Gängige Lispfunktionen

Im Laufe der Zeit werden die meisten Anwender feststellen, dass man zum Konfigurieren immer wieder ähnliche Lispfunktionen benötigt.

`require` Um erst einmal eine Bibliothek nachzuladen verwendet man diesen Befehl:

```
(require 'tex-site)
```

Sofern es im `load-path` eine Datei gibt, deren Inhalt das verlangte Feature liefert, wird diese geladen. In den meisten Fällen wird der Name des Features der gleiche sein wie der der Datei.

`autoload` Mit `require` holt man sich sofort die benötigten Funktionen und Variablen in den aktuellen XEmacsprozess hinein. Das macht natürlich den Start unnötig langsam und wenn ich jedes Mal den das JDE starte, auch wenn ich gerade nur ein Perlprojekt bearbeite ist das nicht unbedingt sinnvoll. Dieser Problematik widmet sich `autoload`. Es bindet eine Funktion an eine Datei. Sobald diese Funktion aufgerufen wird, wird die angegebene Datei nachgeladen:

```
(autoload 'turn-on-eldoc-mode "eldoc" nil t)
```

Sobald also jemand versucht, die Funktion `turn-on-eldoc-mode` aufzurufen, wird XEmacs deren Definition (und damit alles, was weiterhin dafür notwendig ist) aus der Datei `eldoc.el` laden.

`setq` Dieser Aufruf setzt die Werte von Variablen:

```
(setq auto-save-timeout 120)
```

Eine sehr häufige Anwendung ist das Setzen der Variablen `auto-mode-alist`, anhand derer XEmacs bestimmt, in welchen Mode eine neu geöffnete Datei geht:

```
(setq auto-mode-alist
      (append '(("\\.rb$" . ruby-mode)) auto-mode-alist))
```

Die etwas merkwürdige Verwendung der `append`-Funktion[3] soll uns hier nicht weiter stören. Es sei nur erwähnt, dass das so seinen Sinn hat.

`add-hook` Jeder Majormode kommt mit einem gleichnamigen *Hook*. Das ist quasi der Haken, an den man seine eigenen Sachen mit dranhängen kann. Für den `matlab-mode` gibt es den `matlab-mode-hoo`, für den `cperl-mode` den `cperl-mode-hook` usw.

Das Anhängen des eigenen Codes geschieht durch den Aufruf der Funktion `add-hook`. Diese nimmt zwei Argumente entgegen: den Namen des Hooks[4] und die Funktion, die man dazu hängen möchte. Ein Beispiel:

```
(add-hook 'text-mode-hook
          'turn-on-filladapt-mode)
```

Nun wird man in den meisten Fällen mehr als nur eine Funktion anhängen wollen oder aber sogar eine Funktion mit weiteren Argumenten aufrufen wollen. Dazu kann man sich entweder erst eine eigene Funktion definieren, die alles erledigt und die an den Hook anhängen:

```
(defun my-text-mode-hook ()
  "Meine speziellen Einstellungen für den Textmodus."
  (auto-fill-mode 1)
  (turn-on-filladapt-mode))

(add-hook 'text-mode-hook
          'my-text-mode-hook)
```

oder aber man verwendet eine ad-hoc definierte Funktion, die noch nicht einmal einen eigenen Namen hat, weil sie keinen braucht. Dazu verwendet man die Funktion `lambda`, die genau das macht: sie definiert eine Funktion ohne Namen.

```
(add-hook 'text-mode-hook
          '(lambda ()
             (auto-fill-mode 1)
             (turn-on-filladapt-mode)))
```

Die Variante mit der eigenen Funktion hat den Vorteil, dass es leicht ist, die eigenen Erweiterungen wieder zurückzunehmen (siehe dazu die Onlinehilfe zu der Funktion `remove-hook`), aber da man das eh sehr selten benötigt (wenn überhaupt), trifft man die Lambdavariante eigentlich am häufigsten an.

[3] Man könnte ja meinen, dass es reichen würde, wenn man `append` mit dem neuen Element und der Liste, an die es drangehängt wrden soll, aufruft.
[4] Genauer das entsprechende Symbol, aber das soll erstmal nicht weiter interessieren.

`set-key` Nachdem nun die Modes definiert sind, müssen nur noch die Tastaturkürzel an den eigenen Geschmack angepasst werden. Dazu gibt es eine ganze Klasse von Funktionen, die alle irgendwie `set-key` oder `defin-key` heißen. Schauen wir sie uns gleich einige mal anhand von Beispielen an:

```
;; Globale Definiton:
(global-set-key '[(f4)] 'kill-this-buffer)

;; Lokale Definition fuer den aktuellen mode in einer Funktion, die
auf dem passenden Hook liegt:
(defun my-cperl-mode-keys ()
  "Setting local keybindings for major mode: perl."
  (local-set-key '[(meta tab)] 'hippie-expand))

;; Keydefinition mit Angabe der Keymap
(defun ska-latex-mode-keys ()
  "Set keys for latex-mode (AUCTeX)."
  (define-key LaTeX-math-keymap
    (concat LaTeX-math-abbrev-prefix "/")  'LaTeX-math-frac))
```

Es gibt verschiedene Arten, wie man die zu verwendende Tastatursequenz angeben kann, aber man sollte sich auf eine einigen, und diese hier lässt sich auch im GNU Emacs verwenden.

4.5. Eigene Dateien

Für den langjährigen Anwender ist das Customizemenu jedoch mehr so eine Art Quelle für neue Ideen (wenn man von Zeit zu Zeit einfach mal ein wenig stöbert) oder aber ein Platz, um Dinge auszuprobieren. Am Ende werden die allermeisten XEmacser ihre eigenen Dateien haben, die sie nach eigenem Gutdünken organisieren. Denn eines ist mal klar: eine reine `.emacs` oder `init.el` reicht irgendwann einfach nicht mehr aus.

Ein sehr detailliertes und engagiertes Projekt in dieser Richtung ist EMacro ([8]). Dort wird versucht, eine Konfiguration für beide Emacsen (GNU Emacs und XEmacs) zu erstellen, die zudem noch portabel bezüglich verschiedener Betriebssysteme ist und so ziemlich jedes Paket, was den Leuten jemals untergekommen ist, konfiguriert.

An dieser Stelle wird noch eine etwas übersichtlichere Organisation vorgeschlagen, die jedoch nur als Hinweis verstanden werden soll.

Ausgehen tut diese Konfiguration von der Annahme, dass alle Konfigurationsdateien im einem Verzeichnis `/.xemacs` gesammelt werden. [5] Dort gibt es drei Dinge:

- Zunächst gibt es ein Unterverzeichnis `site-lisp`, in dem alle Pakete aus dem Internet, die nicht Bestandteil der eigentlichen XEmacsdistribution sind, gelagert werden.
- Desweiteren findet sich ein zweites Unterverzeichnis, einfallsreich `my` benannt, in dem sich alle eigenen Erweiterungen finden werden.
- Schließlich liegt dort auch die `init.el`, die XEmacs grundlegend konfiguriert.

4.5.1. Die `init.el`

Die Aufgabe dieser Datei ist es, die Umgebung des XEmacs soweit vorzubereiten, dass im Weiteren alles gefunden werden kann und darauf die einzelnen Dateien nachzuladen.

Zunächst werden der `load-path` und einige Verzeichnisse definiert:

```
;; This directory contains all the xemacs relevant stuff:
(defvar my-xemacs-dir
  (expand-file-name "~/.xemacs/my")
```

[5] Tatsächlich habe ich das bereits getan, als es noch keine `init.el` im selbigen Verzeichnis gab. Aber dieser Ansatz ist so naheliegend, dass ich mir darauf wohl nix einbilden darf.

```
"The directory where all the XEmacs configuration (and more) goes.
This contains all my files, not those of others."

(defvar my-site-lisp-dir
  (expand-file-name "~/.xemacs/site-lisp")
"The directory where all the XEmacs files from the Web goe.
This contains all the cool utilities and maybe newer versions of
libraries than on the system."
  )
(setq load-path
      (append
       (list
        my-site-lisp-dir
        (concat my-site-lisp-dir "/docbookide")
        (concat my-site-lisp-dir "/xslide")
        (concat my-site-lisp-dir "/gnuplot-mode"))
        load-path
       ))
```

Hier kann man schon erkennen, dass z. B. das DocbookIDEpaket aus dem Internet nachinstalliert wurde.

Im zweiten Schritt werden die vom XEmacs zu verwendene Customizedatei sowie die zentrale Konfigurationsdatei für persönliche Erweiterungen und einstellungen gesetzt und geladen:

```
(setq custom-file (concat my-xemacs-dir "/config/custom.el"))
(load custom-file t t)))
(load (concat my-xemacs-dir "/config/personal.el") nil nil 1)))
```

Abschließend kann hier noch Code ausgeführt werden, der garantiert am Ende der Initialisierung laufen soll, wie es beispielsweise für das desktop-Paket sinnvoll ist:

```
(load "desktop")
(add-hook 'kill-emacs-hook
      '(lambda ()
         (desktop-truncate search-ring 3)
         (desktop-truncate regexp-search-ring 3)))
(desktop-load-default)
(desktop-read)
```

4.5.2. Das Verzeichnis `my`

Dieses Verzeichnis gliedert sich in weitere Unterverzeichnisse auf:

`config` Die Konfigurationsdateien, die so benötigt werden. Das sind nicht nur `personal.el` und `customize.el` sondern auch die Initfiles von Gnus oder VM und ähnliches sowie die eigenen Tastaturkürzeldefinitionen.

`data` Alle Datenfiles, wie z. B. die BBDB, die recent-files aber auch Autosavedateien und eigene Pixmaps für den VM.

`lisp` Selbstgeschriebene ELisp-Bibliotheken.

`templates` Vorlagen für auto-insert und Skeletons für alle Sprachen.

Für viele Dateien wird in der unten beschriebenen `personal.el` bestimmt, wo sie landen werden, für einige wenige geschieht das bereits in der `init.el`.

Dateien in `config`

Zunächst sind hier die beiden wichtigsten Konfigurationsdateien in dem hier vorgestellten Setup: `custom.el`, das alle Einstellungen aus dem Customizesystem aufnimmt und in der `init.el` auf diesen Dateinamen gesetzt wurde, sowie `personal.el` die alle Hooks und Einstellungen vornimmt, die explizit in ELisp vorliegen. Oftmals werden hier auch Standardeinstellungen explizit noch einmal gesetzt. Der meiste Code hier ist aus den Kommentaren der ELispdateien

selber, die meist mit einer Kurzbeschreibung zur Installation und Anwendung daher kommen, gemopst und im Lauf der Jahre weiter getrieben worden. Für eine beispielhafte `personal.el` siehe Anhang A.1.

Desweiteren finden sich hier sehr eigenwillige Tastatureinstellungen. Ausgehend von den Präfixen (C-x) für global vorhandene und (C-c) für modespezifische Tastenbelegungen habe ich die gleich daneben liegenden (C-v) und (C-b) entsprechend verwendet. Damit gebe ich zwei Tasten (eben (C-v) und (C-b)) auf, die ich so oder so niemals so verwendet habe[6], gewinne aber zwei ganze neue Maps. In den allermeisten Fällen funktioniert das auch reibungslos, nur manchmal wird ein Mode selber versuchen, diese Tasten zu belegen. Die Fälle müssen wirklich sehr selten sein, da ich mich an keinen einzigen gerade erinnern kann. Die beiden Dateien `ska-local-keys.el` sowie `ska-global-keys.el` (siehe Anhang A.2 und A.3) setzen alle diese Tasten.

Auf die hier ebenfalls gelagerten Dateien `vm-init.el` und `gnus-init.el` wird an gesondertet Stelle eingegangen (werden) (Abschnitte 3.13 und 3.14).

Beilbt noch eine grundlegene und fast leere Datei, die lediglich die Beschreibung des Benutzers enthält: `user.el`. Ihr Inhalt ist schnell wiedergegeben:

```
(setq user-mail-address "meine@emailadresse.de")
(setq my-copyright-holder "Mein Name oder MeineFirma")
(provide 'user)
```

(natürlich fehlt in dieser Wiedergabe der klassische ELispkopf mit Copyright und so). Erwähnenswert ist hier eigentlich nur der `my-copyright-holder`, da der Inhalt dieser Variablen in den hier vorgestellten AutoInsertVarianten (s. Abschnitt 3.7) verwendet wird.

Dateien in `lisp`

Neben einer kleinen Datei mit einigen mehr oder weniger nützlichen Kleinigkeiten (`ska-utils.el`) finden sich hier die Dateien von den ELispbibliotheken, die ich selber geschrieben habe. Diesen habe ich (ganz dreist) sogar ein eigenes Kapitel in dieser Einleitung gewidmet (Kapitel 9). Die Funktionen aus `ska-utils.el` werden im Laufe dieses Dokuments meist nebenbei irgendwo erwähnt.

[6] So, so, so ...

5. Schatzkistchen

Spendieren wir doch den kleinen Paketen und Befehlen, die uns das tägliche Leben einfach einfacher und angenehmer gestalten, ein eigenes Kapitel. Hier sollte sich so manches kleines Wunder finden, das vielleicht nur ein kleiner Code-Schnipsel ist, ohne Webseite und Mailingliste, das aber so dermaßen hilfreich sein kann, dass es zumindest hier seinen Ruhm ernten soll.

5.1. Pakete

Zeilennummern Ja, es ist hilfreich, am Rand die Zeilennummern eingeblendet zu bekommen. Der (M-x) `line-number-mode`, der die aktuelle Zeilennummer in der Modeline anzeigt, reicht nicht. Zum Glück haben wir dafür setnu.

Zunächst ein wenig Konfiguration:

```
(setq setnu-line-number-format "%4d ")
(setq setnu-line-number-face (copy-face 'default 'setnu-my-face))
(set-face-background 'setnu-my-face "#bfbfbf")
(set-face-foreground 'setnu-my-face "#777777")
```

Und dann in den Hooks der gewünschten Modes einschalten:

```
(setq cperl-mode-hook
      '(lambda ()
         (turn-on-setnu-mode)))
```

Allerdings hat der Autor dieser Zeilen damit einige Probleme gehabt. Ab und an crasht dieses Paket den gesamten XEmacs!![1] Als Tipp kann hier nur gegeben werden, sofort, wenn beim Löschen oder Einfügen einer Zeile ein extentp-Fehler gemeldet wird, die Zeilennummern vermöge (M-x) `setnu-mode` auszuschalten und alle Dateien zu speichern. Die Erfahrung zeigt, dass `folding-mode` und PCL-CVS Probleme bereiten.

Screen Real Estate Bildschirmplatz ist kostbar. Warum also sollte man eine weitere Zeile an ein Menu verschwenden, dass man eh nur hier und da mal benötigt? Auftritt active-menu. Vom Autor dieser Zeilen erdacht[2], rudimentär implementiert und von erfahrenen Programmierern fertig gestellt, bietet es die Möglichkeit, die Menuleiste nur dann einzublenden, wenn die Maus in die Nähe des oberen Fensterrandes kommt. Damit dabei die Framegröße konstant bleibt, muss active-menu noch an die verwendete Schriftart angepasst werden. Der aktive Bereich kann ebenfalls eingestellt werden.

```
(require 'active-menu)
(setq active-menu-sensitive-height 10)
(setq active-menu-frame-compensation 1)
(turn-on-active-menu)
```

Locate Ein Interface zum Programm `locate`, das einen Buffer öffnet, der die gefunden Dateien zum direkten Anspringen enthält. Unter Umständen muss man sich `locate.el` erst noch aus der GNU Emacs Distribution kopieren.

```
(require 'locate)
(global-set-key '[(control v) (control l)] 'locate)
```

[1] Ja, zwei Ausrufezeichen sind sicherlich schlechter Stil, aber ein Crash im XEmacs ist so selten, dass sie es mir hier wert sind.
[2] Man verzeihe die schamlose Selbstbeweihräucherung, bitte

5.2. Befehle

Einfache Befehle, die man in sein tägliches Repertoire aufnehmen sollte.

`transpose-...` Auf den Tasten C-t, C-x t und M-t liegen die Transpose-Funktionen. Mit ihnen lassen sich Zeichen, Zeilen und Worte respektive vertauschen. Das Verhalten von C-t mag dem einen oder der anderen nicht genehm genug sein. Der folgende Code ändert das Verhalten leicht.

```
(defun ska-electric-transpose-chars ()
  "Replacement for the default transpose-chars command.
This is usually bound to C-t and it behaves somewhat unintelligent
because I always have to move back one char when I mistyped
something. This function checks whether the user is typing and then
goes back one char itself."
  (interactive)
  (if (eq last-command 'self-insert-command)
      (progn (transpose-chars -1)
             (forward-char))
    (transpose-chars -1)))
```

Groß-Kleinschreibung Oftmals sieht man Leute, die das erste Zeichen eines Wortes löschen und danach das gleiche Zeichen, nur groß geschrieben, wieder eingeben. Das ist Geschichte: M-c für *capitalize*, M-u und M-l für *upper* und *lower*.

Filename-Completion Ja, ja, Filename-Completion[3] im Minibuffer und in der Shell kennt jeder. Aber mitten im Buffer, einfach da, wo der Cursor gerade steht? Ungewöhnlich. Praktisch. Wie konnte ich nur jemals ohne ...? Das folgende setzt Shift-TAB entsprechend.

```
(global-set-key '[(iso-left-tab)] 'comint-dynamic-complete)
```

Home und End Auf deutschen Tastaturen auch Pos1 und End genannt. Der im Folgenden vorgestellte Code sorgt dafür, das der Druck auf die Home-Taste

1. an den Anfang der Zeile springt,
2. wenn man sich dort bereits befindet an den Anfang des Fensters (also in die höchste, sichtbare Zeile),
3. und im letzten Schritt an den Anfang des Buffers.

```
(global-set-key '[(home)] 'chb-home)
(global-set-key '[(end)]  'chb-end)
;; ...
(defun chb-home ()
  (interactive)
  (setq zmacs-region-stays t)
  (if (not (bolp))
      (beginning-of-line)
    (if (eq this-command last-command)
    (cond
     ((not (= (point) (window-start)))
      (move-to-window-line 0)
      (beginning-of-line))
     (t
      (goto-char (point-min)))))))

(defun chb-end ()
  (interactive)
  (setq zmacs-region-stays t)
```

[3] Deutsch für Filename-Completion? Dateinamenkomplettierung? Nein, danke.

```
  (if (not (eolp))
      (end-of-line)
    (if (eq this-command last-command)
    (cond
     ((not (= (point) (save-excursion
                (move-to-window-line -1)
                (end-of-line)
                (point))))
      (move-to-window-line -1)
      (end-of-line))
     (t
      (goto-char (point-max)))))))
```

Schnelle Markierung Diesen Code verwende ich seit Jahren immer wieder, mehrmals täglich. Gefunden habe ich das mal in Michael Koflers *Linux*-Buch ([13]) und es ist einfach nur praktisch! Einfach schnell eine Markierung setzen „Hier war ich“ irgendwo anders hin gehen, dort was nachschauen, ändern oder kopieren und zack! wieder zurück.

```
;; suggested key-bindings:
(global-set-key '(control \.) ska-point-to-register)
(global-set-key '(control \,) ska-jump-to-register)

(defun ska-point-to-register()
  "Store cursorposition _fast_ in a register. Use ska-jump-to-register
to jump back to the stored position."
  (interactive)
  (setq zmacs-region-stays t)
  (point-to-register 8))

(defun ska-jump-to-register()
  "Switches between current cursorposition and position
that was stored with ska-point-to-register."
  (interactive)
  (setq zmacs-region-stays t)
  (let ((tmp (point-marker)))
    (jump-to-register 8)
    (set-register 8 tmp)))
```

Ausführbare Skripte OK, jetzt mal ehrlich: wie oft schon wurde in einer Skriptsprache ein neues Programm angelegt, geschrieben, gestartet und das folgende „Permission denied“ von der Shell mit einem `chmod +x programm` beantwortet. Der in dieser Doku beschriebene auto-insert-Mechanismus kann da schon weiterhelfen, aber diese Lösung ist hier ist noch generischer: wann immer eine Datei gespeichert wurde, wird geschaut, ob sie einen Shebang besitzt und wenn ja, wird sie gleich ausführbar gemacht, so sie es denn nicht schon ist.

```
(add-hook 'after-save-hook
        #'(lambda ()
        (and (save-excursion
               (save-restriction
                 (widen)
                 (goto-char (point-min))
                 (save-match-data
                   (looking-at "^#!"))))
             (not (file-executable-p buffer-file-name))
             (shell-command (concat "chmod u+x " buffer-file-name))
             (message
              (concat "Saved as script: " buffer-file-name)))))
```

Keine TABS Ja, es ist ein alter Streit unter Programmierern, ob es denn nun gut ist, TABs zu Einrücken zu verwenden oder aber lieber die richtige Anzahl Leerzeichen. Ich hänge der letzteren Fraktion an und da ich eh für jede Sprache einen eigenen Hook pflege, setze ich überall dort, wo ich keine TABS sehen möchte folgenden Code mit ein (hier am Beispiel für den cperl-mode)

```
(setq cperl-mode-hook
      '(lambda ()
         (make-local-hook 'write-contents-hooks)
          (add-hook 'write-contents-hooks #'ska-untabify
                    nil t)))
```

Dazu braucht es noch folgende Funktion

```
(defun ska-untabify ()
  "My untabify function as discussed and described at
http://www.jwz.org/doc/tabs-vs-spaces.html
and improved by Claus Brunzema:
 - return nil to get 'write-contents-hooks' to work correctly
   (see documentation there)
 - 'make-local-hook' instead of 'make-local-variable'
 - when instead of if
Use some lines along the following for getting this to work in the
modes you want it to:

\(add-hook 'some-mode-hook
           '(lambda ()
               (make-local-hook 'write-contents-hooks)
                (add-hook 'write-contents-hooks 'ska-untabify nil t)))"
  (save-excursion
    (goto-char (point-min))
    (when (search-forward "\t" nil t)
      (untabify (1- (point)) (point-max)))
    nil))
```

Wiederholen eines Kommandos Nicht nur zum Wiederholen und ggf. Ändern eines gerade gegebenen Kommandos, auch wenn man mal bei einer Suche- und Ersetzaktion nicht weiß, wie man einen Zeilenumbruch eingeben soll, hift `repeat-complex-command`:

```
(global-set-key '[(shift f9)] 'repeat-complex-command)
```

Es zeigt im Minibuffer den letzten Befehl in Lisp-Form an. Somit kann man Kleinigkeiten leicht ändern.

Rechtecke Das Auschneiden von rechteckigen Bereichen kann man auf zwei Arten erreichen.

- Mit der Maus durch Binden der Funktion `mouse-track-do-rectangle` an eine Mausaktion (standardmäßig liegt das auf `M-Mausziehen`). Dann wir sogar nur ein Rechteck als Region markiert.
- Mit den Tasten durch ganz normales Markieren der Region und dann aber nur verwenden des Bereiches, der den Beginn der Region und die aktuelle Position als Eckpunkte hat. Dann helfen die Tasten `C-x` `r` `k` (`kill-rectangle`) und `C-x` `r` `y` (`yank-rectangle`) weiter. Ein `C-h` `a` `rectangle` hilft weiter.

6. Todo

So, hier nur ein kleiner Platzhalter, eine kleine Merkhilfe für mich, was (noch) alles geplant ist.

- auto-fill-mode
- filladapt
- ispell
- ECB?
- mehr Elispintro.

7. Übriggebliebenes

7.1. Filevariablen

Es ist möglich, in einer beliebigen Textdatei diverse Einstellungen zu speichern, die XEmacs dann beim nächsten Öffnen der Datei verwendet. AUCTeX macht davon großen Gebrauch, indem es bei einem Projekt aus vielen Einzeldateien das Masterfile definiert (ähnliches kann auch PSGML), aber auch C-Coder können so ihre Standards festlegen und definieren, wie groß ein Tabulatorsprung sein soll oder ähnliches.

Das Format ist einfach: am Ende der Datei erscheint eine spezielle Zeile mit dem Inhalt „Local Variables: “. Diese Zeile kann durch die für die jeweilige Datei geltenden Kommentarregeln auskommentiert sein, z. B.:

```
%%% Local Variables:
```

für eine LaTeX-Datei oder auch

```
/* Local Variables: */
```

für eine C-Datei. XEmacs erkennt die Teile vor „Local“ und nach dem Doppelpunkt und merkt sie sich. Kommen danach Zeilen, die das gleiche Präfix und Postfix haben, werden sie interpretiert. So kann ein Eintrag

```
;;; Local Variables: ***
;;; mode:lisp ***
```

sicherstellen, dass die Datei in den lisp-mode geht. In AUCTeX kann man so das Zentraldokument einstellen:

```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: "main"
%%% End:
```

Wir erkennen hier bereits, dass das Schlüsselwort „End:“ (mit passenden Präfix und Postfix) die Filevariablen beendet.

Als ein kleines Kuriosum betrachte ich folgende Datei:

```
-*- normal -*-

;; Local variables:
;; enable-local-eval: t
;; hack-local-variables-hook: save-buffers-kill-emacs
;; end:
```

Sie schaltet in den `normal-mode`, in dem die Filevariablen unabhängig vom Wert der Variablen `enable-local-variables` verarbeitet werden. Tja, und dann wird XEmacs einfach mal beendet. Diese Datei lässt sich nicht öffnen.

7.2. Speedbar

Abbildung 7.1 zeigt eine Speedbar. Dieses Paket stellt funktionale Einheiten einer Datei da und lässt sie direkt anspringen. Speedbar erkennt Dateien an ihren Dateinamen[1] und kann mit einer ganzen Menge von Formaten sicher umgehen.

[1] Was dazu führt, dass man seine Perlprojekte doch wieder auf `.pl` enden lässt und ggf. für die Installation in den Produktivbetrieb diese unschöne Endung wegfallen lässt.

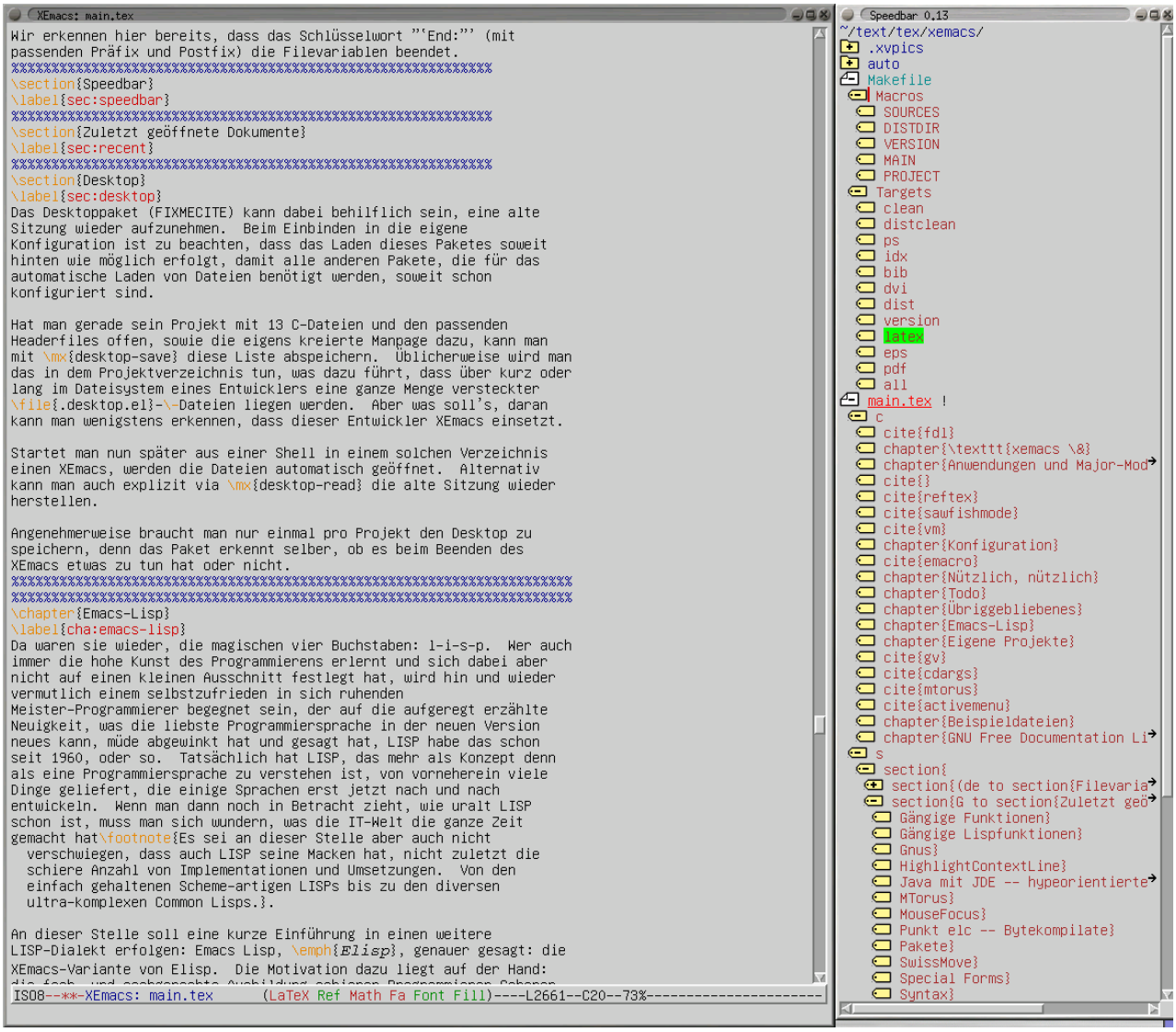


Abbildung 7.1.: Rechts neben dem Hauptframe findet sich eine Speedbar, mit vielen aufgeklappten Bereichen. Vom Makefile werden die Makros sowie die verschieden Targets angezeigt, von den LaTeX-Sourcen dieses Dokuments sehen wir einige Kapitel und Abschnitte. Der grün hinterlegte Text stammt vom (nicht sichtbaren) Mauszeiger, der beim Anlegen des Bildschirmschnappschusses dort verweilte.

- C-Programmdateien zeigen ihre Funktionendefinitionen,
- C++-Dateien nach Klassen sortiert ihre Memberfunktionen und -variablen,
- LaTeX-Dateien bieten ihre Kapitelüberschriften nach Hierarchie sortiert an und
- Makefiles ihre Makros und Targets.

Was für C++ gilt, gilt so oder ähnlich auch für die meisten anderen objektorientierten Sprachen und was für C gilt, gilt für andere prozedurale Sprachen. Die Speedbar kann dabei sogar auf zwei verschiedene Mechanismen zurückgreifen, die diese Analyse übernehmen, entweder imenu oder das ältere etags. Sofern imenu vorhanden ist (und bei neueren XEmacsen ist das der Fall), sollte es bevorzugt werden.

Für die mausfreie Verwendung empfiehlt es sich, die Funktion `speedbar-get-focus` auf eine Tastenkombination zu legen. Dann wird bei Bedarf ein Speedbarframe erst geöffnet und ansonsten die Speedbar angesprungen [2], z. B.:

```
(global-set-key '[(control v) (control s)] 'speedbar-get-focus)
```

Neuere Speedbarversionen werden mit einem eigenen Kommando zum Umschalten des Einausklappzustandes geliefert, sollte dieses Kommando nicht vorhanden sein, hilft folgender Code weiter:

```
(defun ska-speedbar-toggle-expand ()
  (interactive)
  (beginning-of-line)
  ;; if we're on a [+] line we can simply expand
  (if (re-search-forward ":\\s-*.\\+."
                         (save-excursion (end-of-line) (point))
                         t)
      (speedbar-expand-line)
    ;; if starts with "\\s>", we're in a expanded list
    ;; else
    ;; go back to the last line with [-] at the beginning
    (progn
      (end-of-line)
      ;; correction by CHB
      (re-search-backward ":\\s-*.-." (point-min) t)
      (speedbar-contract-line))
    ))

;; um im entsprechenden Hook die Keybindings zu setzen:
(defun ska-speedbar-keys ()
  "Set keybindings in major navigation tool Speedbar."
  (define-key speedbar-key-map '[(space)]    'ska-speedbar-toggle-expand)
  )
```

Wer jetzt noch seinen Windowmanager mitteilen kann, wo sich ein Speedbarframe automatisch öffnen soll [3], ist klar im Vorteil.

Achja, die Bedienung fehlt noch. Die wichtigsten Tasten werden in Tabelle 7.1 beschrieben.

7.3. Zuletzt geöffnete Dokumente

Das Standardpaket für ein Menu mit den zuletzt bearbeiteten Dokumenten ist recent-files. Mit wenigen Zeilen initialisiert, kann es nicht nur die letzten Dateien anzeigen sondern auch eine eigene Liste von permanenten Dateien verwalten. Das ist für solche Dauerbrenner wie `.bashrc` oder `.xemacs/init.el` sicherlich eine interessante Methode. Es lässt sich konfigurieren, ob die einzelnen Abschnitte eigene Untermenus sein sollen und wie viele Dateien angezeigt werden. Entsprechender Initialisierungscode könnte folgendermaßen aussehen:

[2] so der Windowmanager denn mitspielt, aber das sollte er eigentlich.
[3] nämlich direkt rechts neben der immer gleichen Größe des XEmacsframes

SPACE	Aufklappstatus hin- und herschalten (zumindest mit dem Code aus dem Text. Mehr braucht man selten.
RETURN	Wählt einen Eintrag an und wir werden zur entsprechenden Stelle im Text gewarpt.
Mittlere Maus	wie RETURN.
+	Klappt einen Baum auf.
-	Klappt einen Baum wieder zu.
g	Bringt die Speedbar auf den neuesten Stand
Rechte Maus	Wichtiges Kontextmenu
q	Beendet die Speedbar.

Tabelle 7.1.: Tastenbelegungen in der Speedbar.

```
(load "recent-files")
(setq recent-files-dont-include
      '("~$" "tmp/." "INBOX" ".bbdb" ".newsrc." ))

(setq recent-files-non-permanent-submenu t)
(setq recent-files-commands-submenu t)
(setq recent-files-number-of-entries 40)
(recent-files-initialize)
```

Eine Alternative hierzu ist picklist ([3]), das ein Fork einer früheren Version von recent-files ist, aber noch weitere Funktionalitäten hinzufügt. So liefern neuere Versionen eine Art *apropos* für die Liste mit und die Datendatei hält deutlich mehr Informationen über die Dokumente vor, so dass andere Sortierungen möglich sind. Entstanden ist picklist ursprünglich, weil recent-files keine Cursorpositionen abgespeichert hat (wozu es mittlerweile natürlich noch andere Pakete gibt).

7.4. Desktop

Das Desktoppaket ([18]) kann dabei behilflich sein, eine alte Sitzung wieder aufzunehmen. Beim Einbinden in die eigene Konfiguration ist zu beachten, dass das Laden dieses Paketes soweit hinten wie möglich erfolgt, damit alle anderen Pakete, die für das automatische Laden von Dateien benötigt werden, soweit schon konfiguriert sind.

Hat man gerade sein Projekt mit 13 C-Dateien und den passenden Headerfiles offen, sowie die eigens kreierte Manpage dazu, kann man mit M-x `desktop-save` diese Liste abspeichern. Üblicherweise wird man das in dem Projektverzeichnis tun, was dazu führt, dass über kurz oder lang im Dateisystem eines Entwicklers eine ganze Menge versteckter `.desktop.el`-Dateien liegen werden. Aber was soll's, daran kann man wenigstens erkennen, dass dieser Entwickler XEmacs einsetzt.

Startet man nun später aus einer Shell in einem solchen Verzeichnis einen XEmacs, werden die Dateien automatisch geöffnet. Alternativ kann man auch explizit via M-x `desktop-read` die alte Sitzung wieder herstellen.

Angenehmerweise braucht man nur einmal pro Projekt den Desktop zu speichern, denn das Paket erkennt selber, ob es beim Beenden des XEmacs etwas zu tun hat oder nicht.

8. Emacs-Lisp

Da waren sie wieder, die magischen vier Buchstaben: l-i-s-p. Wer auch immer die hohe Kunst des Programmierens erlernt und sich dabei aber nicht auf einen kleinen Ausschnitt festlegt hat, wird hin und wieder vermutlich einem selbstzufrieden in sich ruhenden Meister-Programmierer begegnet sein, der auf die aufgeregt erzählte Neuigkeit, was die liebste Programmiersprache in der neuen Version neues kann, müde abgewinkt hat und gesagt hat, LISP habe das schon seit 1960, oder so. Tatsächlich hat LISP, das mehr als Konzept denn als eine Programmiersprache zu verstehen ist, von vorneherein viele Dinge geliefert, die einige Sprachen erst jetzt nach und nach entwickeln. Wenn man dann noch in Betracht zieht, wie uralt LISP schon ist, muss man sich wundern, was die IT-Welt die ganze Zeit gemacht hat[1].

An dieser Stelle soll eine kurze Einführung in einen weitere LISP-Dialekt erfolgen: Emacs Lisp, *Elisp*, genauer gesagt: die XEmacs-Variante von Elisp. Die Motivation dazu liegt auf der Hand: die fach- und sachgerechte Ausbildung schierer Programmiererscharen, die erstens sich selbst und zweitens der Welt durch Entwicklung von viel, viel ELisp-Code weiterhelfen.

Wie üblich zunächst die Hinweise zu bereits bestehender, sehr guter Dokumentation. Es gibt eine englischsprachige Einleitung zu EmacsLisp, die im Info-System zu finden ist.

C-h I `Emacs-lisp-intro` führt dorthin. Zudem existiert noch eine Referenz, ebenfalls im Info-System: C-h I `Lispref` sollte jederzeit dorthin verzweigen. Sollte auf diesem Wege eines der beiden Schmuckstücke nicht zu finden sein, kann es sich um ein Problem im Info-System handeln. Oft hilft es dann, mal in einer Shell `info` aufzurufen und dort mal zu suchen und zu blättern. Unangenehmerweise haben die internen Info-Seiten des XEmacs meist einen anderen Einstiegspunkt als dasjenige von der Shell.

Diese Einleitung soll pragmatisch nur dahin führen, bestehenden Lisp-Code zu verstehen und soll inspirieren, eigene kleine Code-Fragmente zu schreiben. Gegen die ausführliche bestehende und genannte Dokumentation tritt diese Einleitung nicht an.

8.1. Syntax

Wie bereits beschrieben, lässt sich die Syntax von LISP mit zunächst ganz wenigen, einfachen Regeln beschreiben:

- Jeder etwas komplexere Ausdruck ist durch runde Klammern eingefasst; so ein Ausdruck heißt *Liste*.
- Variablennamen oder Literale sind ebenfalls Ausdrücke.
- Das erste Element in jeder Liste ist ein Funktions-Aufruf, alle weiteren Elemente sind Argumente zu dieser Funktion. Eine solche Liste heißt auch *Form*.
- Jedes Argument kann wieder eine Liste sein.
- Steht vor einer Liste ein Hochkomma (`'`), so ist das erste Element keine Funktion und die weiteren Elemente werden ebenso wenig evaluiert.
- Es gibt special Forms, wie `if`, `while` oder auch `defun`, die eine speziellere Syntax besitzen.

Mit diesen wenigen Regeln lässt sich ein sehr, sehr großer Teil des bestehenden Lisp-Codes bereits ganz gut lesen.

[1] Es sei an dieser Stelle aber auch nicht verschwiegen, dass auch LISP seine Macken hat, nicht zuletzt die schiere Anzahl von Implementationen und Umsetzungen. Von den einfach gehaltenen Scheme-artigen LISPs bis zu den diversen ultra-komplexen Common Lisps.

8.2. Special Forms

Kommen wir nun zu den Special-Forms.

`if`
```
(if CONDITION
     THEN
   ELSE)
```

Die Special-Form `if` erwartet drei Argumente, die ihrerseits beliebige Listen sein dürfen, führt entsprechend des Ergebnis des ersten Arguments dann das zweite oder dritte aus und liefert das Ergebnis der ausgeführten Liste zurück. Ein Beispiel gefällig?

```
(insert
 (if (eobp)
     "Ende des Buffers"
   "Mittendrin"))
```

Dabei sehen wir auch gleich, wie man den Rückgabe-Wert von `if` verwendet: Wenn wir uns aktuell am Ende des Buffers befinden, liefert `if` „Ende des Buffers“ zurück, ansonsten „Mittendrin“, das Ergebnis wird dann der Funktion `insert` übergeben, die den String direkt einfügt. Nicht sehr sinnvoll, zugegeben, aber erklärend.

Eine Besonderheit gilt es noch zu beachten, die für das `if` von ELisp gilt: der THEN-Teil darf nur ein Ausdruck sein. Es gilt also, den Code so zu formulieren, dass der THEN-Teil der unwesentliche ist und der ELSE-Abschnitt der entscheidende. Dort dürfen beliebig viele Anweisungen stehen:

```
(if (= (point) (point-max))
    (message "am Ende ist die Datei %d Zeichen lang" (point-max))
  (goto-char (point-min))
  (insert "Hier ist der Anfang"))
```

Die Bedingung ist die gleiche wie im vorigen Beispiel, nur anders formuliert, der THEN-Teil, wie die Einrückung anzeigt, besteht aus einer `message`[2] und der ELSE-Abschnitt geht an den Anfang der Datei[3] und schreibt ein wenig sinnloses Zeug dort hinein. Sinnvoller könnte es sein, zu schauen, ob dort der String

```
#!/usr/bin/perl -w
```

bereits steht (ein weiteres `if` vermutlich mit einem `looking-at` als Bedingung) und es ggf. einzusetzen.

`while` Gleich nach der bedingten Verarbeitung mittels `if`, vielleicht sogar noch kurz davor, kommen die Schleifenkonstrukte. Soundsooft immer den gleichen Code ausführen, dafür braucht es eine `while`-Schleife. Auch in Lisp. Neben vielen anderen Möglichkeiten, die hier aber nicht interessieren sollen. Bei `while` läuft das so, dass das erste Argument als Bedingung verstanden wird. Solange dieser Ausdruck „wahr“ zurück liefert, werden das zweite und alle folgenden Argumente immer wieder einmal ausgeführt.

```
(while (not (eobp))
  (forward-line)
  (beginning-of-line)
  (insert ";;"))
```

Der gezeigte Code kommentiert ab der aktuellen Cursorposition jede Zeile einer Lispdatei aus, indem jeder Zeile zwei Semikolon vorangestellt werden. Dabei erfolgt der Test mit einer Verneinung auf das Ende des Buffers und die auszuführenden Schritte sind:

[2] mit diesem Befehl kann man Nachrichten in der Echo-Area anzeigen, zudem versteht er noch gängige Format-Anweisungen, wie sie auch `printf` in C beispielsweise schluckt, hier ein `%d` für eine Integer-Zahl.

[3] Das hier ist der richtige Weg in Programmen, `beginning-of-buffer` soll dazu nicht verwendet werden, das ist für die interaktive Verwendung gedacht

1. Gehe eine Zeile vor,
2. gehe an den Anfang der Zeile und
3. füge zwei Semikolon ein.

Das wars auch schon zu `while`.

`defun` Oh, ein ganz wichtiger Befehl, nämlich der, um Befehle zu definieren.

```
(defun neue-funktion (arg1 arg2)
  "Die erste Zeile erklaert in einem Satz was die Funktion tut.
Alle weiteren Saetze koennen sich dann ruhig auch ueber mehrere
Zeilen erstrecken.  Lediglich der erste sollte dieser Konvention
folgen, da er fuer Kurzhilfen verwendet wird.  Diese Doku wird
angezeigt, wenn man mit C-h f neue-funktion nach Hilfe ruft.  Cool,
oder?  Andere Sprachen muessen sich da mit spezieller Syntax in den
Kommentaren vor der Definition behelfen."
  (interactive)
  (message "ich mach nicht viel."))
```

So, der Dokumentationsstring, der zu `defun` als drittes Argument hat sich bereits selber in großen Teilen erklärt. Das erste Argument ist offensichtlich der Name der neuen Funktion und das zweite Argument ist eine Liste von Argumenten. Alle weiteren Argumente werden als auszuführende Befehle verstanden.

`let` Dieser Befehl dient zum Anlegen von lokalen Variablen. Es ist sicherlich nicht sehr hilfreich, wenn jede Variable `i` oder so allgemein benamste wie `start` gleich global wären. Was wäre, wenn man sich in einer Funktion befindet, die eine ebensolche Variable verwendet und eine andere Funktion aufruft, die sie ebenfalls verwendet? Chaos wäre vorprogrammiert. Den Geltungsbereich einer Variablen nennt man *Scope* und man sollte zusehen, dass man den globalen Scope nicht mit Allgemeinplätzen vergriesgnaddelt. Also: für globale Variablennamen schön ein eigenes Kürzel davor setzen, meist den eigenen Paktenamen, und sonst eben `let` einsetzen.

```
(let ((start (point))
      (end (+ (point) 9))
      current)
  (setq current start)
  (while (< current end)
    (forward-char 1)
    (setq current (point))
    (insert (format "%d" current))))
```

OK, der Beispielcode ist Blödsinn, aber er zeigt, worum es geht: Im ersten Argument (man beachte die Klammerung, es geht bis hinter `current`) werden die Variablen `start` und `end` definiert und jeweils mit einem Startwert versehen, während `current` nur definiert wird. In derauf folgenden Argumenten, die erneut den auszuführenden Block ausmachen, werden diese Variablen dann verwendet. Mit `setq` wird der Wert von `current` bei jedem Schleifendurchlauf neu gesetzt.

Es gibt natürlich noch durchaus einige weitere Spezialformen, und von Makros[4] wollen wir gar nicht erst anfangen. Die hier Vorgestellten werden für den Anfang ausreichen, eine ganze Menge bestehenden Codes zu verstehen. Wer auf eine unbekannte Funktion trifft, stellt einfach den Cursor darauf und tippt (C-h) (f). Daraufhin wird XEmacs die Funktion unter dem Cursor als Voreinstellung beim Prompt nach dem Funktionsnamen anbieten, und die Dokumentation, die ganz besonders bei den wichtigen Funktionen sehr gut ist, wird angezeigt.

[4] Hier sind *Echte Lisp Macros*, englisch: *Lisp True Macros* gemeint, die nicht mit den von C oder C++ bekannten Makros verwechselt werden sollten.

8.3. Datentypen

Offensichtlich sollte es in einer Programmiersprache, die *List Processing* heißt, einen Datentyp *Liste* geben. Aber auch andere gängige Datentypen, wie *number* (mit den Spezialisierungen *integer* und *float*), *string*, *vector* (so was ähnliches wie eine Liste), *hash-table* und *symbol* sind neben einigen anderen vorhanden. In den allermeisten Fällen wird es dem angehenden ELisp-Programmierer nicht so wichtig sein, mit welcher Art Datentyp er es zu tun hat, da dieses Konzept in Lisp nicht die Wichtigkeit genießt wie es in anderen Sprachen, wie C, der Fall ist.

8.3.1. Prädikate

Gut zu wissen in diesem Zusammenhang ist es aber, dass es (meistens) eine Funktion gibt, die feststellen kann, ob ein Ding einen bestimmten Typ hat oder nicht. Solche Funktionen, die eine Ja/Nein-Entscheidung treffen, heißen *Prädikate* und sind üblicherweise daran zu erkennen, dass ihr Name relativ unmotiviert auf ein „p“ endet: `numberp`, `hash-table-p` oder auch `listp`. Leider ist die Nomenklatur, wie hier bereits zu erkennen, nicht ganz eindeutig, ob das p nun noch einen Strich davor bekommt oder nicht.

Eigentlich gehört auch ein Abschnitt über Prädikate gar nicht in den Abschnitt über Datentypen, aber es bot sich gerade so an. Natürlich gibt es noch viel mehr Prädikate. So könnte ein Kommando, das die `grep`-Funktion in der Funktionalität von Perl nachbaut, ein Prädikat zum Filtern einer Liste verwenden:

```
(defun perl-grep (l predicate)
  (defun helper (ret-list rest)
    (if (null rest)
    (reverse ret-list)
        (progn
          ;; hier rufen wir das praedikat auf
          (if (funcall predicate (car rest))
          (setq ret-list (cons (car rest) ret-list)))
          (helper ret-list (cdr rest)))))
  (helper '() l))
```

8.3.2. Spezialitäten in ELisp

Die bisher genannten Typen sind einerseits nur eine Untermenge von andererseits recht generellen Lisp-Datentypen. ELisp hat einige Typen, die auch nur in diesem Umfeld Sinn machen. Beispiele dafür sind: *Buffer*, *Marker*, *Extent*, *Window*, *Frame* und sogar *Window Configuration*, also die Einstellung der aktuellen Frameaufteilung.

All diese Datentypen erleichtern die Arbeit mit dem XEmacs und die Programmierung von nützlichen Päckchen. Die genaue Dokumentation findet sich in den Info-Seiten (`info lispref`, auf den Unterseiten *Lisp Data Types*)

8.4. Gängige Funktionen

Sammeln wir doch mal einige der wichtigen Funktionen zusammen, die man für einfache Erweiterungen benötigen wird.

8.4.1. Bewegung

Wir verordnen nun dem Point ein wenig Sport und Bewegung, zwar nicht an der frischen Luft aber immerhin.

`forward-char` Geht eine anzugebende Anzahl von Zeichen vor- oder rückwärts (falls die Zahl negativ mitgegeben wurde).

`forward-line` Entsprechendes zeilenweise.

`forward-word` Naja, ...

`backward-char` Wie `forward-char` mit negativem Argument.

backward-word Kann man sich vielleicht auch schon denken.

backward-line So, und jetzt kommt die Überraschung: diese Funktion gibt es (zum Zeitpunkt dieses Schreibens) nicht. Komisch eigentlich.

search-forward Sucht vorwärts nach einem anzugebenden String. Dabei kann die Suche bis zu einer bestimmten Position begrenzt werden. Neben dem gibt es noch weitere optionale Argumente.

```
(search-forward "hallo")
```

Wenn die Funktion ihr Ziel findet, wird der Point hinterher *hinter* dem gefunden Wort stehen.

search-backward Diese Funktion gibt es natürlich auch.

re-search-forward und re-search-backward Suchen nicht nur nach einem ganz normalen String vor- oder rückwärts, sondern verwenden einen regulären Ausdruck zum Suchen. Diese Ausdrücke gehören, speziell in ELisp, zu den eher fortgeschrittenen Techniken und werden an dieser Stelle nicht erläutert. Erfahrene Perlprogrammierer werden sich allerdings nicht sehr wohl mit den Regexps aus ELisp fühlen. Wer fortgeschrittene Textverarbeitung betreiben will, wird aber nicht um sie herum kommen.

goto-char

beginning-of-line

beginning-of-buffer Nein, nein, nein, diese Funktion ist für den interaktiven Gebraucht gedacht. In eigenem Code sollte man lieber `(goto-char (point-min))` verwenden.

8.4.2. Einfügen und Löschen

Frühe und einfache eigene Funktionen werden oft nur Textbausteine halbwegs parametrisiert einfügen. Etwas weiter entwickelte werden auch schonmal Text löschen. Was die Superfunktionen machen, sei dahin gestellt, hier interessieren mal nur Einfügen und Löschen.

insert Insert übernimmt einen String und fügt ihn an aktueller Position ein:

```
(insert "hier bin ich")
(let ((text "hier auch"))
  (insert text))
```

format Wer etwas mehr als nur feste Strings einfügen möchte bedient sich dieser Formatierungsfunktion, die ähnlich arbeitet wie `printf` aus der Sprache C. Es gibt spezielle Formatierungsanweisungen, die mit einem Prozentzeichen beginnen und sich auf weitere Argumente zu der Funktion beziehen. Die genauen Anweisungen entnehme die interessierte Leserin bitte der Dokumentation.

```
(let ((objekt '(eine liste)))
  (insert
   (format "Eine Zahl: %d, eine magische Objektrepräsentation: %s"
           23 objekt)))
```

concat Diese Funktion gehört ebenfalls zum Handwerkszeug des Texteinfügens, da sie einfach mehrere Zeichenketten aneinander fügt.

```
(concat "text1" "text 2"
       (format "%d" 42))
```

kill-region Befehle zum Löschen (nicht alle aber ausreichend viele) beginnen mit `kill-`. Dabei wird allerdings der ausgeschnittene Text auf den `kill-ring` geschoben, wo er später wieder verwendet werden kann, was bei großen Textmengen vielleicht nicht so wünschenswert ist. Jedenfalls gibt es neben `kill-region`, das eine Region definiert durch einen Anfangs- und einen Endpunkt entgegen nimmt, noch `kill-line`, `kill-word` sowie `kill-paragraph` und einige mehr.

8.4.3. Positionsbestimmung

point

point-at-eol

point-at-eob

eobp

bolp

8.4.4. Benutzereingaben

interactive

completing-read

read-number

read-string

8.4.5. Listenverarbeitung

list

append

reverse

cons

mapc

mapconcat

8.4.6. Einfach wichtige Funktionen

progn

shell-command

interactive

save-excursion

message

car, **cdr**

8.5. Punkt elc – Bytekompilate

Lispprogramme können kompiliert werden, so auch im Falle des ELispdialektes. Diese Dateien sind daran zu erkennen, dass sie nicht auf `.el` sondern auf `.elc` enden. Durch die Kompilation ist es für XEmacs möglich, die Dateien schneller zu laden und den darin befindlichen Code auszuführen. Bei eigenen Dateien der Halbschwergewichtsklasse und neueren Rechnern macht das vermutlich eher mehr Ärger als Sinn. Zum einen werden die Kompilate beim Laden vorgezogen, so vorhanden, was zu Verwirrungen führen kann, wenn man die .el-Datei bearbeitet und nicht an das Kompilat denkt, zum anderen sind neuere Maschinen doch schnell genug, um auch die reinen Dateien zu laden. Bei großen Paketen, die ohnehin nur aus dem Internet geladen und installiert werden, ohne große Veränderungen, macht das schon eher Sinn.

Oftmals kommen Pakete mit einem eigenen `Makefile`, in dem man meist noch eine Variable

```
EMACS=emacs
```

ersetzen muss:

```
EMACS=xemacs
```

Denn die kompilierten Dateien sind nicht zwischen den beiden Emacsgeschmacksrichtungen austauschbar.

Manuell geht das einfach mit (M-x) `byte-compile-buffer` oder aber (M-x) `byte-compile-file`.

8.6. Beispiele

Für den Moment muss hier auf Hunderttausende von bestehenden ELispzeilen in der XEmacsinstallation verwiesen werden. Dazu muss man ggf. die Quellen (also die .el-Dateien) aus einem Extrapaket der eigenen Distribution nach installieren. Irgendwann werden hier aber bestimmt ein paar Schritt-für-Schritt Beispiele auftauchen.

9. Eigene Projekte

Dieses mag vielleicht eine möglichst allgemein gehaltene Einleitung zu einem weit verbreiteten Programm sein. Dennoch wird sie ganz wesentlich von einer Person geschrieben: *mir*. Daher kann ich mir ein paar Dinge leisten. Beispielsweise finde ich es völlig in Ordnung, hier in Icherzählerperspektive zu schreiben, was für eine anspruchsvolle und gute technische Dokumentation –na sagen wir mal– unüblich ist. Neben dem kann ich aber auch einfach mal meine eigenen Projekte vorstellen und ihnen somit ein wenig Prominenz verschaffen[1]. Immerhin will ich mich auf diejenigen beschränken, die ich selber für sinnvoll und halbwegs gut gelungen erachte.

9.1. HighlightContextLine

Zum Betrachten von Postscriptdateien gibt es das Programm `gv` ([15]). Dieses kommt mit einem bemerkenswerten Feature daher: wenn die Seite größer ist als der angezeigte Bereich, wird beim Scrollen eine schwarze Linie am ehemaligen Bildrand eingeblendet. Ich frage mich manchmal, warum das nicht viel mehr Programme können, da es die Orientierung doch sehr erleichtert. Das gilt besonders, weil verschiedene Programme verschiedene Auffassungen vom richtigen Scrollverhalten haben. So scroll `less` in halben Bildschirmen, wenn man auf die Taste (PgDn) drückt, andere Programme zeigen gerade noch die zuletzt sichtbare Zeile an, `acroread` springt an das nächste Ende einer Seite und XEmacs hat gar eine konfigurierbare Anzahl von *Überlappzeilen*.

Gegen dieses Dilemma tritt `highlight-context-line.el` an, in dem es beim Scrollen die zuletzt sichtbare Zeile farbig hinterlegt. Wenn die Zeile natürlich gerade eine Leerzeile war, sieht man nix, aber ansonsten funktioniert das sehr gut.

Die Einrichtung ist denkbar einfach:

```
(load "highlight-context-line")
(setq highlight-context-line-background "#c5c500")
```

Jetzt nur noch eine Farbe aussuchen und das war's dann auch schon.

9.2. MouseFocus

Bei Windowmanagern, die im X-Windowsystem die Fenster verwalten und verschieben, vergrößern und verkleinern, verzieren und verschwinden lassen, ist es schon immer üblich gewesen, zwischen verschiedenen Arten des Fokusverhaltens wählen zu können. Das (anderswo weitverbreitete und nach meiner Meinung unbenutzbare) Klicktofocusverhalten bedeutet, dass man ein Fenster anklicken muss, damit es den Fokus bekommt. Das Verhalten „Fokus folgt der Maus", das auch unter anderen Namen noch bekannt ist, hingegen gibt jedem Fenster den Fokus, in das die Maus hineingeht. Für die XEmacs-Windows gilt eigenlich ja das erstgenannte. Das empfinde auch ich als richtig. Andere Leute aber nicht. So ein Bekannter von mir, auf dessen ausdrücklichen Wunsch ist dieses Modul geschrieben habe, das ein „Fokus folgt der Maus" Verhalten für XEmacs-Fenster liefert. Einfach nur nachladen und einschalten:

```
(require 'mouse-focus)
(turn-on-mouse-focus)
```

[1] Diese Prominenz haben sie vielleicht gar nicht verdient, aber auch diesen Fall verbuche ich unter den gegebenen Freiheiten.

9.3. SwissMove

Tja, *SwissMove* habe ich die Navigation durch große Dateien genannt, bei der man in immer kleiner werdenden Schritten voran- und zurückschreitet. Wenn man weiß, wohin man will, kann man hiermit mit wenigen Schritten ans Ziel finden. Bei jedem Schritt wird die Schrittweite halbiert. Die erste Schrittweite geht genau die Hälfte der absoluten Länge in die gewünschte Richtung. Stünde ich also zu Beginn auf einer Position bei etwa 30% der Datei, blieben noch 60% bis zum Ende. Ein SwissMove vorwärts brächte mich an eine Stelle bei etwa 60%. Ein sofortiger Schritt zurück von dort aus würde die erste Schrittweite von 30% halbieren und mich also zu etwa 45% bringen. Man muss es einfach mal versuchen. Ich habe viel Spass gehabt, das zu programmieren und fand die Idee auch gut, aber verwende diese Funktion dennoch nicht. Aber wer weiß, vielleicht gibt es ja experimentierfreudige Menschen, die diese Art der Navigation verwenden.

```
(require 'swiss-move)
(global-set-key '[(shift prior)] 'swiss-move-line-up)
(global-set-key '[(shift next)]  'swiss-move-line-down)
```

9.4. CDargs

CDargs ([10]) ist eigentlich ein Programm für die Shell, das Bookmarks für das Dateisytem ermöglicht. Diese Bookmarks lassen sich jedoch auch vom XEmacs aus auslesen. Dazu liefet das CDargs-Paket ein entpsrechendes .el-File mit. Die XEmacsvariante kann sogar noch einen entsprechenden Hook ausführen und so beispielsweise einen evtl. vorhandenen Desktop automatisch laden, sobald das Verzeichnis mit (M-x) `cv` angesprungen wurde.

```
(require 'cdargs)
(add-hook 'cdargs-warped-hook
          '(lambda ()
             (when (file-exists-p
                    (expand-file-name desktop-basefilename "./"))
               (desktop-read))))
```

9.5. MTorus

Eine bewegte Geschichte, tolle Features und viel zu erzählen gibt es hier. Momentan sei dazu auf [12] verwiesen. Auch wenn dieser Abschnitt in diesem Kapitel der kleinste ist, ist doch dieses Projekt wohl mein größtes (und mittlerweile in kompetentere Hände gewandert).

9.6. TrackScroll

Mich persönlich überzeugen ja viele der tollen Errungenschaften des Graphical Userinterface Designs nicht wirklich. Ich habe mich im XEmacs der Toolbar sehr früh entledigt, habe dann einigen Aufwand betrieben, um die Menuleiste soweit wie möglich loszuwerden ([1]) und auch die Scrollbar finde ich unsäglich in ihrer Verwendung. Warum muss ich da auf so kleine Bereiche mit der Maus zielen? Dass sich die Scrollbar noch auf meinem Bildschirm befindet verdankt sie zweierlei:

1. Sie ist klein und der Platz, den sie braucht geht vom horizontalen Raum ab, nicht vom (wertvollen) vertikalen.
2. Sie gibt eine gute Übersicht über die aktuelle Position innerhalb einer Datei und die Größe der Datei.

Zum Scrollen selber verwende ich sie nur noch selten, denn das hier vorgestellte Paket liefert zwei Funktionalitäten:

1. Anfassen und Verschieben des Fensterinhaltes. Wenn ich einfach nur mit der mittleren Maustaste irgendwo im Fenster ziehe, wandert der Inhalt mit. Das wird allerdings von manchen Modes überschrieben.
2. Scrollbar überall. Bei gleichzeitig gedrückter Control-Taste wird der gesamte Fensterbereich zu einer Scrollbar.

Die Verwendung ist einfach:

```
(require 'track-scroll)
```

und ob es dem persönlichen Geschmack entspricht, muss jede Anwenderin selber ausprobieren.

A. Beispieldateien

A.1. `personal.el`

```
;;; personal.el --- Setup modes, hooks and other personal stuff
;; Copyright (C) 2000-2004 Stefan Kamphausen

;; Author: Stefan Kamphausen <mail@skamphausen.de>
;;         http://www.skamphausen.de/
;; Time-stamp: <13-Feb-2005 01:13:40 ska>

;; Keywords:
;; This file is not part of XEmacs.

;; This code is free software; you can redistribute it and/or modify it
;; under the terms of the GNU General Public License as published by
;; the Free Software Foundation; either version 2, or (at your option)
;; any later version.

;; This code is distributed in the hope that it will be useful, but
;; WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
;; General Public License for more details.

;; You should have received a copy of the GNU General Public License
;; along with this code; see the file COPYING.  If not, write to the Free
;; Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
;; 02111-1307, USA.

;;; not Synched up with: (X)Emacs

;;; Commentary:
;; This is The Next Generation of my XEmacs customization.
;; What formerly had been ~/.xemacs-personal now should be in this
;; file. It sets up all the major modes and their hooks, loads several
;; files (keys and utils and other non-standard stuff), sets the paths
;; and more. Just have a look.
;; Please note that this is a folded file, thus it hides almost
;; everything when you first open it. Simply right-click on the points
;; where you see three braces ('{') and that tree should appear. Or
;; use M-x fold-open-buffer (C-c @ C-o) to show everything.


;; To use:
;; put in your .emacs (or ~/.xemacs/init.el for recent XEmacsen):
;;
;;(defvar my-xemacs-dir
;;  (expand-file-name "~/.xemacs/my")
;;"The directory where all the XEmacs configuration (and more) goes.
;;This contains all my files, not those of others.")
;;
;;(cond
;; ((string-match "XEmacs" emacs-version)
;;  (load (concat my-xemacs-dir "/config/personal.el") nil nil t)))
```

```
;;
;; Actually this file is part of a large-scale setup I wrote and I
;; tried to create it as generically as possible so that you can use
;; it yourself with only a few changes (which probably will be in
;; user.el).

;;; Code:


;;============================================================================
;;{{{ PATHs, Filenames and Modules
;;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

;;{{{ Define some directories containing our own stuff
(defvar my-lisp-dir
  (concat my-xemacs-dir "/lisp")
  "The personal lisp directory. All self-written files and files from
the web goe here. Larger packages like e.g. VM or PSGML are installed
in subdirs."
  )

(defvar my-config-dir
  (concat my-xemacs-dir "/config")
  "This directory contains the configuration files. Keybindings as well
as local/personal settings plus init files of other packages
e.g. vm-init-file"
  )

(defvar my-templates-dir
  (concat my-xemacs-dir "/templates")
  "This directory contains all templates, inserting features and
skeletons I use together with XEmacs."
  )
(defvar my-data-dir
  (concat my-xemacs-dir "/data")
  "This directory contains all data XEmacs likes to store."
  )
;;}}}
;;{{{ Setting paths and list
(setq data-directory-list
      (append
       (list
        (concat my-xemacs-dir "/etc"))
       data-directory-list
       ))

(setq load-path
      (append
       (list
        my-config-dir
        my-templates-dir
        my-lisp-dir)
       load-path
       ))
;;}}}
;;{{{ Loading the site files
;; (hopefully) the only file with user specific informations
(load "user" nil nil)
(setq my-host-file
      (concat my-config-dir "/"
              (replace-in-string (exec-to-string "hostname")
                                 "\n" "")
              ".el"))
(if (file-readable-p my-host-file)
```

```
    (load my-host-file nil nil))
(setq my-domain-file
      (concat my-config-dir "/"
              (replace-in-string (exec-to-string "domainname")
                                 "\n" "")
              ".el"))
(if (file-readable-p my-domain-file)
    (load my-domain-file nil nil))
;;}}}
;;{{{ The settings of the new filenames for some packages
(setq vm-init-file            (concat my-config-dir "/vm-init.el"))
(setq gnus-init-file          (concat my-config-dir "/gnus.el"))
(setq save-place-file         (concat my-data-dir   "/places.el"))
(setq recent-files-save-file  (concat my-data-dir   "/recent-files.el"))
(setq bbdb-file               (concat my-data-dir   "/bbdb"))
(setq w3-hotlist-file         (concat my-data-dir   "/w3-hotlist"))
(setq w3-configuration-directory (concat my-data-dir "/w3"))
(setq eshell-directory-name   (concat my-data-dir   "/eshell/"))
(setq shadow-info-file        (concat my-data-dir   "/shadows"))
(setq shadow-todo-file        (concat my-data-dir   "/shadow_todo"))
(setq ctypes-file-name        (concat my-data-dir   "/ctypes.el"))
(setq bookmark-default-file   (concat my-data-dir   "/bookmarks.el"))
(setq auto-save-directory     (concat my-data-dir   "/autosave/"))
(setq auto-save-list-file-prefix (concat my-data-dir   "/autosave/saves-"))
;;}}}
;;{{{ Our own save require defun
;; inspired by Init-safe-require from the new sample.init.el file
(defun ska-safe-require (feat)
"Try to REQUIRE the specified feature.  Errors occurring are silenced.
\(Perhaps in the future there will be a way to get at the error.)
Returns t if the feature was successfully required."
  (condition-case nil
      (require feat)
    (error (display-warning 'personal-load (format "Unable to load %s " feat)))))
;;}}}
;;{{{ Now load the keybindings (they require the utils then)
(ska-safe-require 'ska-global-keys)
(ska-safe-require 'ska-local-keys)
;;}}}

;;}}}
;;============================================================================

;;============================================================================
;;{{{ MAJOR MODES
;;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;;{{{ Early Stuff
;; This section contains things that I want to enable globally or can't find a
;; way to start later and work correctly (like setnu).
(load "setnu")
(turn-on-font-lock)
;;}}}
;;============================================================================
;;{{{ Text Mode
(ska-safe-require 'filladapt)
(setq auto-mode-alist
      (append '(("README" . text-mode)) auto-mode-alist))

(add-hook 'text-mode-hook
          '(lambda ()
             (auto-fill-mode 1)
             (turn-on-filladapt-mode)
             ;;(abbrev-mode 1)
             ;;(if (file-readable-p "~/.abbrev_defs")
```

```
            ;;  (read-abbrev-file "~/.abbrev_defs"))
            ))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Perl Mode: cperl
(setq cperl-mode-hook
      '(lambda ()
         ;; my keybindings
         (ska-coding-keys cperl-mode-map)
         (ska-cperl-mode-keys)
         (auto-fill-mode 1)
         (turn-on-setnu-mode)
         (turn-on-font-lock)
         ;; if you don't want tabs
         (setq indent-tabs-mode nil)
         ;; full featured mode:
         ;;(setq cperl-hairy t)
         ;; alternatively:
         ;;(setq cperl-auto-newline-after-colon t)
         ;;(setq cperl-electric-parens "({[")
         (setq cperl-auto-newline nil)
         (setq cperl-electric-linefeed nil)
         (setq cperl-electric-parens nil)
         (setq cperl-electric-lbrace-space t)
         (setq cperl-electric-keywords nil)
         (setq cperl-mode-abbrev-table nil)
         (setq cperl-lazy-help-time 1)
         (setq cperl-help t)
         (setq cperl-pod-here-fontify nil)
         (setq cperl-highlight-variables-indiscriminately t)
         (make-local-hook 'write-contents-hooks)
         (add-hook 'write-contents-hooks #'ska-untabify
                   nil t)
         ))

;; from newsgroups (with title "trailing spaces in Cperl Mode")
;; Here's my solution:
;;
;; M-X customize RET cperl-faces RET
;; Find "Invalid Face".  Set its value to: '(quote default)
;; Set, Save, Done
;;
;; I don't know if this is the "correct" way to accomplish this, or even
;; the best way, but it seems to work.  It seems odd there's not a
;; easier way to accomplish this, since everything else about cperl-mode
;; seems customizable.
(defun strip-end-of-line-whitespace ()
  (interactive)

  (let ((count 0))
    (save-excursion
      (goto-char (point-min))
      (let ((case-replace nil)
            (case-fold-search nil))
        (while (re-search-forward "[ \t]+$" nil t)
          (replace-match "" t t)
          (incf count))))
    (when (interactive-p)
      (message "Stripped %d line%s."
               count
               (if (= count 1) "" "s")))))


;;}}}
```

```
;;-----------------------------------------------------------------------------
;;{{{ Ruby
;; this code is from inf-ruby.el
(autoload 'ruby-mode "ruby-mode"
  "Mode for editing ruby source files")
(setq auto-mode-alist
      (append '(("\\.rb$" . ruby-mode)) auto-mode-alist))
(setq interpreter-mode-alist (append '(("ruby" . ruby-mode))
                                     interpreter-mode-alist))
(autoload 'run-ruby "inf-ruby"
  "Run an inferior Ruby process")
(autoload 'inf-ruby-keys "inf-ruby"
  "Set local key defs for inf-ruby in ruby-mode")
(autoload 'rubydb "rubydb3x"
  "Run ruby debugger")
(add-hook 'ruby-mode-hook
          '(lambda ()
             (ska-coding-keys ruby-mode-map)
             (ska-ruby-mode-keys)
             (auto-fill-mode 1)
             (turn-on-filladapt-mode)
             (turn-on-setnu-mode)
             (turn-on-font-lock)
             (inf-ruby-keys)
             ))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ HTML PSGML Mode
;; not working in hook?
(setq html-auto-sgml-entity-conversion t)
;;(setq sgml-data-directory (concat my-data-dir "/psgml"))
(add-hook 'html-mode-hook
          '(lambda ()
             (auto-fill-mode 1)
             (setq sgml-insert-missing-element-comment nil)
             (setq sgml-set-face nil)
             (setq sgml-markup-faces nil)
             (setq fill-column 100)
             (setq html-helper-use-expert-menu t)
             (setq html-helper-do-write-file-hooks t)
             (add-to-list 'sgml-catalog-files
                          (concat sgml-data-directory "/CATALOG"))
             ))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ CSS - Cascading Style Sheets
;; needs css-mode.el from
;; http://www.garshol.priv.no/download/software/css-mode/
(autoload 'css-mode "css-mode")
(setq auto-mode-alist
      (cons '("\\.css\\'" . css-mode) auto-mode-alist))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Sawfish - Editing librep-lisp
;; needs sawfish.el from
;; <URL:http://www.davep.org/emacs/#sawfish.el>
(autoload 'sawfish-mode "sawfish" "sawfish-mode" t)
(setq auto-mode-alist (cons '("\\.sawfishrc$"  . sawfish-mode) auto-mode-alist)
      auto-mode-alist (cons '("\\.jl$"         . sawfish-mode) auto-mode-alist)
      auto-mode-alist (cons '("\\.sawfish/rc$" . sawfish-mode) auto-mode-alist))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ SGML, XML and XSL Mode
(autoload 'sgml-mode "psgml" "Major mode to edit SGML files." t)
```

```
(autoload 'xml-mode "psgml" "Major mode to edit XML files." t)
(setq auto-mode-alist
      (append
       '(("\\.xml$" .xml-mode))
       auto-mode-alist))

;; docbook still isn't working??
(add-hook 'sgml-mode-hook
          '(lambda ()
;            ;;(setq sgml-catalog-files '("/usr/share/xemacs/21.1.8/etc/psgml/CATALOG"))
             ;; correct display of ISO8859-1 chars (umlauts etc)
             (load "iso-sgml")
             (setq isosgml-modes-list
                   (append
                    '(xml-mode)
                    isosgml-modes-list))

             (ska-sgml-keys)
             (setq sgml-auto-insert-required-elements t)
             (setq sgml-insert-end-tag-on-new-line t)
             (setq sgml-insert-missing-element-comment nil)
             ;; (setq sgml-custom-markup '(("menustring" "insert\rstring")))
             (setq sgml-indent-data t)
             ;;display element name in modeline.
             ;; this makes problems when the DTD isn't parsed yet:
             ;(setq sgml-live-element-indicator t)

             ;;;;; Faces.
             (setq sgml-set-face t)   ;; whether PSGML colorizes buffer
             (setq sgml-auto-activate-dtd t) ;; autom parse DTD
             (setq sgml-use-text-properties t)
             (setq sgml-markup-faces
                   '((start-tag     . font-lock-keyword-face)
                     (end-tag   . font-lock-keyword-face)
                     (comment   . font-lock-comment-face)
                     (pi        . font-lock-string-face)
                     (sgml      . font-lock-reference-face)
                     (doctype   . font-lock-variable-name-face)
                     (entity    . font-lock-function-name-face)
                     (shortref  . font-lock-type-face)))
             ;; if you don't like the default font-lock faces make
             ;; your own:
;;             (make-face 'sgml-comment-face)
;;             (make-face 'sgml-doctype-face)
;;             (make-face 'sgml-end-tag-face)
;;             (make-face 'sgml-entity-face)
;;             (make-face 'sgml-ignored-face)
;;             (make-face 'sgml-ms-end-face)
;;             (make-face 'sgml-ms-start-face)
;;             (make-face 'sgml-pi-face)
;;             (make-face 'sgml-sgml-face)
;;             (make-face 'sgml-short-ref-face)
;;             (make-face 'sgml-start-tag-face)

;;             (set-face-foreground 'sgml-comment-face "dimgrey")
;;             (set-face-foreground 'sgml-doctype-face "maroon")
;;             (set-face-foreground 'sgml-end-tag-face "blue2")
;;             (set-face-foreground 'sgml-entity-face "seagreen")
;;             (set-face-foreground 'sgml-ignored-face "maroon")
;;             (set-face-background 'sgml-ignored-face "gray90")
;;             (set-face-foreground 'sgml-ms-end-face "maroon")
;;             (set-face-foreground 'sgml-ms-start-face "maroon")
;;             (set-face-foreground 'sgml-pi-face "maroon")
;;             (set-face-foreground 'sgml-sgml-face "maroon")
```

```
;;             (set-face-foreground 'sgml-short-ref-face "goldenrod")
;;             (set-face-foreground 'sgml-start-tag-face "blue2")

;;           (setq-default sgml-markup-faces
;;                         '((comment . sgml-comment-face)
;;                           (doctype . sgml-doctype-face)
;;                           (end-tag . sgml-end-tag-face)
;;                           (entity . sgml-entity-face)
;;                           (ignored . sgml-ignored-face)
;;                           (ms-end . sgml-ms-end-face)
;;                           (ms-start . sgml-ms-start-face)
;;                           (pi . sgml-pi-face)
;;                           (sgml . sgml-sgml-face)
;;                           (short-ref . sgml-short-ref-face)
;;                           (start-tag . sgml-start-tag-face)))
             ))


;; DocBook IDE mode
;; from http://www.nwalsh.com/emacs/docbookide/
;;(autoload 'docbook-mode "docbookide" "Major mode for DocBook documents." t)
;(add-hook 'docbook-mode-hook
;          'turn-on-font-lock)
;;(setq auto-mode-alist
;;      (append
;;       (list
;;        '("\\.docbook" . docbook-mode))
;;       auto-mode-alist))


;; XSL mode
(autoload 'xsl-mode "xslide" "Major mode for XSL stylesheets." t)
(add-hook 'xsl-mode-hook
          '(lambda ()
             ;; xslide uses font-lock
             (setq-default sgml-set-face nil)
             (setq sgml-markup-faces nil)
             (turn-on-font-lock)
             ))

(setq auto-mode-alist
      (append
       (list
        '("\\.fo$" . xsl-mode)
        '("\\.xsl$" . xsl-mode))
       auto-mode-alist))

;; sxml-mode -- better?
;(autoload 'sxml-mode "sxml-mode" "Major mode for editing XML documents." t)
;(setq auto-mode-alist
;      (append
;       '((".*\.xml$" . sxml-mode )
;        (".*\.xsl$" . sxml-mode))
;       auto-mode-alist))
;; and there is a package xxml.el which fontifies XML docs, but only
;; in gnu emacs?
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ DTD Mode
(autoload 'dtd-mode "tdtd" "Major mode for SGML and XML DTDs." t)
(autoload 'dtd-etags "tdtd"
  "Execute etags on FILESPEC and match on DTD-specific regular expressions."
  t)
(autoload 'dtd-grep "tdtd" "Grep for PATTERN in files matching FILESPEC." t)
```

```
;; Turn on font lock when in DTD mode
(add-hook 'dtd-mode-hooks
          'turn-on-font-lock)

(setq auto-mode-alist
      (append
       (list
        '("\\.dcl$" . dtd-mode)
        '("\\.dec$" . dtd-mode)
        '("\\.dtd$" . dtd-mode)
        '("\\.ele$" . dtd-mode)
        '("\\.ent$" . dtd-mode)
        '("\\.mod$" . dtd-mode))
       auto-mode-alist))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ DQS Stuff
(setq auto-mode-alist
      (append
       '(("\\.dqs$" .ksh-mode))
       auto-mode-alist))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Postscript
;; must require it, when autoloading, my ps-postscript-command is overwritten
;;(autoload 'postscript-mode "postscript.el" "" t)
(ska-safe-require 'postscript)
(setq auto-mode-alist
      (cons '("\\.e?ps$".postscript-mode) auto-mode-alist))
(defconst ps-postscript-command '("gv" "-"))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ BraveGNUWorldMode
(autoload 'bravegnuworld-mode "bravegnuworld")
(setq auto-mode-alist
      (append
       '(("BraveGNUWorld.*\.html" . bravegnuworld-mode))
       auto-mode-alist))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ eLisp Mode

(add-hook 'emacs-lisp-mode-hook
          '(lambda ()
             ;; meine Keybindings
             (ska-coding-keys emacs-lisp-mode-map)
             (ska-elisp-mode-keys)
             (auto-fill-mode 1)
             (turn-on-font-lock)
             (turn-on-setnu-mode)
             (turn-on-filladapt-mode)
             (turn-on-eldoc-mode)
             (make-local-hook 'write-contents-hooks)
             (add-hook 'write-contents-hooks #'ska-untabify
                       nil t)
             ))

(add-hook 'lisp-interaction-mode-hook
          '(lambda ()
             (ska-coding-keys emacs-lisp-mode-map)
             (ska-elisp-mode-keys)
             (setq mode-name "LispI")
             (turn-on-eldoc-mode)
```

```
            (turn-on-font-lock)
            (turn-on-setnu-mode)
            (make-local-hook 'write-contents-hooks)
            (add-hook 'write-contents-hooks 'ska-untabify)
            ))
;;}}}
;;-----------------------------------------------------------------------------
;;-----------------------------------------------------------------------------
;;{{{ Common Lisp Mode
;; Taken from CHBs chb-modes.el (ilisp with cmucl)
(setq cmulisp-program "/tools/common/cmucl2/bin/lisp")
;(setq cmulisp-local-source-directory "/tools/common/cmucl/src")
(setq common-lisp-hyperspec-root "http://localhost/externalDocu/Common_Lisp_HyperSpec/")

(autoload 'run-ilisp   "ilisp" "Select a new inferior Lisp." t)
(autoload 'cmulisp     "ilisp" "Inferior CMU Common Lisp." t)
(defun cmu ()
  (interactive)
  (cd command-line-default-directory)
  (cmulisp))

(add-hook 'lisp-mode-hook #'(lambda ()
                              (require 'ilisp)
                              (local-set-key '[(return)]
                                             #'(lambda ()
                                                 (interactive "*")
                                                 (newline-and-indent-lisp)
                                                 (move-to-column (current-indentation))))))

(add-hook 'ilisp-load-hook
          #'(lambda ()
              (setq ilisp-*prefix* "\C-c")
              (setq ilisp-*use-fsf-compliant-keybindings* t)
              (setq ilisp-*enable-cl-easy-menu-p* t)
              (setq ilisp-*enable-scheme-easy-menu-p* t)
              (setq lisp-no-popper t)
              (setq ilisp-motd nil)
              (if (eq window-system 'x)
                  (setq lisp-font-lock-keywords lisp-font-lock-keywords-2))
              (defkey-ilisp '[(control c) (control h)] 'hyperspec-lookup)
              (defkey-ilisp '[(control b) (control b)] #'comint-interrupt-subjob)
              (add-hook 'ilisp-init-hook
                        #'(lambda ()
                            (default-directory-lisp ilisp-last-buffer)
                            ;; this is sooo ugly, but
                            ;; comint-output-filter-functions and
                            ;; similar stuff just DID NOT WORK.
                            (require 'advice)
                            (require 'comint)
                            (defadvice comint-insert (after ilisp-comint-insert activate)
                              (save-selected-window
                                (select-window
                                 (get-buffer-window (ilisp-buffer) t)
                                 t)
                                (goto-char (point-max))
                                (recenter -1)))

                            (when (file-readable-p "./.ilisp-inferior-lisp-startup.lisp")
                              (ilisp-send "(load \"./.ilisp-inferior-lisp-startup.lisp\")")
                              (message ".ilisp-inferior-lisp-startup.lisp loaded"))))
              (add-hook 'ilisp-mode-hook
                        #'(lambda ()
                            (defkey-ilisp '[(control c) (control c)]
                              #'compile-defun-and-go-lisp)
```

```
;                           (defkey-ilisp
;                             '[(f11)]
;                             #'(lambda ()
;                                 (interactive)
;                                 (with-current-buffer (ilisp-buffer)
;                                   (comint-send-eof)
;                                   (while (comint-check-proc (current-buffer))
;                                     (sleep-for 0.1))
;                                   (cmulisp))))
                            (setq truncate-lines nil)))
             ))

(add-hook 'cmulisp-hook
          #'(lambda ()
              (setq ilisp-init-binary-extension "x86f")
              (setq ilisp-init-binary-command "(progn \"x86f\")")
              (setq ilisp-binary-extension "x86f")
              (setq comint-prompt-regexp
                    "^\\([0-9]+\\]+\\|\\*\\|[-a-zA-Z0-9]*\\[[0-9]+\\]:\\|[-A-Z0-9]*>\\) ")
              ))

;;}}}
;;-----------------------------------------------------------------------------
;;{{{ SQL Mode
(autoload 'sql-mode "sql" "SQL Editing Mode" t)
(setq auto-mode-alist
      (append
       '(("\\.sql$" . sql-mode))
       auto-mode-alist))
(add-hook 'sql-mode-hook
          '(lambda ()
             (ska-coding-keys sql-mode-map)
             (ska-sql-mode-keys)
             (sql-highlight-oracle-keywords)
             ))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ C-MODE

(add-hook 'c++-mode-hook
          '(lambda ()
             ;; my keybindings
             (ska-coding-keys c++-mode-map)
             (ska-c-common-mode-keys c++-mode-map)
             (ska-c++-mode-keys)
             ))
(add-hook 'c-mode-hook
          '(lambda ()
             ;; my keybindings
             (ska-coding-keys c-mode-map)
             (ska-c-common-mode-keys c-mode-map)
             (ska-c-mode-keys)
             ))

(add-hook 'c-mode-common-hook
          '(lambda ()
             (auto-fill-mode 1)
             ;; with c-ignore-auto-fill we have indention only in comments :)
             (setq fill-column 70)
             ;; determines which context to show when matching paren off screen
             (setq paren-backwards-message t)
             ;; line numbers
             (turn-on-setnu-mode)
             ;; syntax highlighting
```

```
            (turn-on-font-lock)
            ;; mark the current line -- somehow it doesn't really
            ;; work together with setnu.el
            ;;(marky-mode)
            ;; ctype dynamically scans files for new type definitions
            (ska-safe-require 'ctypes)
            ;; hereby we collect all we've seen
            (setq ctypes-write-types-at-exit t)
            ;; .. and read it
            (ctypes-read-file nil nil t t)
            ;; needed for ever collecting, too:
            (ctypes-auto-parse-mode 1)
            ;; testing the oo-browser from beopen.com
            ;;(load "br-start")
            (setq c-indent-comments-syntactically-p t)
            ;; testing the electric and hungry features...
            ;; (c-toggle-auto-hungry-state t)
            ;; ... no, I don't like em...
            (make-local-hook 'write-contents-hooks)
            (add-hook 'write-contents-hooks #'ska-untabify
                      nil t)
            (setq indent-tabs-mode nil)
            ))

;;}}}
;;-----------------------------------------------------------------------------
;;{{{ JDE Java Development
;; until I read about a way to load JDE when I need it I'll comment it
;; out:
;;(ska-safe-require 'jde)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Gnuplot-Mode

;; gnuplot-mode.el is no standard
(autoload 'gnuplot-mode "gnuplot" "gnuplot major mode" t)
(autoload 'gnuplot-make-buffer "gnuplot" "open a buffer in gnuplot mode" t)
(setq auto-mode-alist
      (append
       '(("\\.gp$" . gnuplot-mode)
         ("\\.gnu$" . gnuplot-mode))
       auto-mode-alist))
(add-hook 'gnuplot-mode-hook
          '(lambda ()
             ;; meine Keybindings
             (ska-coding-keys gnuplot-mode-map)
             ))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ LaTeX Handling
;; I only do LaTeX, no plain-TeX, no ConTeXt or such, be warned.
(ska-safe-require 'tex-site)

(setq TeX-parse-self t) ; Enable parse on load.
(setq TeX-auto-save t) ; Enable parse on save.
(add-hook 'TeX-mode-hook
          '(lambda ()
             (setq ispell-parser 'tex)
             ))

(setq LaTeX-label-function 'reftex-label)
(add-hook 'LaTeX-mode-hook
          '(lambda ()
             ;; meine keybindings
```

```
            (ska-coding-keys TeX-mode-map)
            (ska-latex-mode-keys)
            (fume-add-menubar-entry)
            (setq fume-display-in-modeline-p nil)
            (LaTeX-math-mode)
            (turn-on-reftex)
            (fume-add-menubar-entry)
            (setq fume-display-in-modeline-p nil)
            (when (string-match "diary" buffer-file-name)
              (turn-on-tex-diary))
            ))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ BiTeX-Mode-Hook
 (add-hook 'BibTeX-mode-hook
          '(lambda ()
            (ska-bibtex-mode-keys)
            ))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ (Unix) Manual
;;(setq Manual-switches '("-a"))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Visiting files

(add-hook 'find-file-hooks
          '(lambda ()
             ;; this makes xemacs WAAY slow (at least in CC files)
             ;;(fume-add-menubar-entry)
             ;;(marky-mode) doesnt work correctly (yet)
             ;;(ska-fit-fill-column-to-frame)
             ;(ska-add-fixme-highlighting)
             (auto-insert)
             ))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Writing files
(add-hook 'write-file-hooks
          '(lambda ()
             (time-stamp)
             ))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Info Mode
;; I'd like to read info in Helvetica, but I can't get it to work :(
;(add-hook 'info-mode-hook
;         '(lambda ()
;            (set-frame-property (buffer-dedicated-frame) [default
;                                                          font]
;                                "-*-Helvetica-Medium-R-*-*-*-120-*-*-*-*-*-*")
;            ))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ R - Gnu's S
;;(ska-safe-require 'ess-site)
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Povray-Mode
(setq auto-mode-alist
      (append
       '(("\\.pov$" . pov-mode))
       auto-mode-alist))
(autoload 'pov-mode "pov-mode"
```

```
  "Major mode for edition povray scene files" t nil)
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Speedbar-Mode
(add-hook 'speedbar-mode-hook
          '(lambda ()
             (ska-speedbar-keys)))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ MatLab Mode
(autoload 'matlab-mode "matlab" "Matlab Editing Mode" t)
(add-to-list
 'auto-mode-alist
 '("\\.m$" . matlab-mode))
(setq matlab-indent-function t)
(setq matlab-shell-command "matlab")
;; the original in matlab.el does not work for me :(
(copy-face 'paren-match 'matlab-region-face)
(add-hook 'matlab-mode-hook
          '(lambda ()
             (ska-matlab-mode-keys)
             ;; line numbers
             (turn-on-setnu-mode)
             ;; syntax highlighting
             (turn-on-font-lock)
             (make-local-hook 'write-contents-hooks)
             (add-hook 'write-contents-hooks #'ska-untabify
                       nil t)
             (setq indent-tabs-mode nil)
             ))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Mail Mode
(add-hook 'mail-setup-hook
          '(lambda ()
             (ska-mail-keys)
             ))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Makefiles
(add-hook 'makefile-mode-hook
          '(lambda ()
             (ska-makefile-keys)
             ))
;;}}}
;;-------------------------------------------------------------------------
;;{{{ PCL-CVS
;(setq cvs-inhibit-copyright-message t)
(add-hook 'cvs-mode-hook
      #'(lambda ()
;         (ska-cvs-keys)
          ;;(setq cvs-program "/lan/cvs/cvs")
          ))
;;}}}
;;-------------------------------------------------------------------------
;;}}}
;;=========================================================================

;;=========================================================================
;;{{{ LISP-PACKAGES

;;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;;-------------------------------------------------------------------------
;;{{{ TeX-Diary
```

```
(ska-safe-require 'tex-diary)

;;{{{ MTorus
(defun mtorus-my-buffer-skip-p (buffer)
  "The my predicate used for 'mtorus-buffer-skip-p'
is to skip only the special buffers whose name begins with a space."
  (or
   (string-match "[ ]+"         (buffer-name buffer))
   (string-match "[Cc]ompletio" (buffer-name buffer))
   (string-match "[Cc]ompil" (buffer-name buffer))))
(setq mtorus-buffer-skip-p 'mtorus-my-buffer-skip-p)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ TAB free coding
(defun ska-untabify ()
  "My untabify function as discussed and described at
http://www.jwz.org/doc/tabs-vs-spaces.html
and improved by Claus Brunzema:
 - return nil to get 'write-contents-hooks' to work correctly
   (see documentation there)
 - 'make-local-hook' instead of 'make-local-variable'
 - when instead of if
Use some lines along the following for getting this to work in the
modes you want it to:

\(add-hook 'some-mode-hook
          '(lambda ()
              (make-local-hook 'write-contents-hooks)
               (add-hook 'write-contents-hooks 'ska-untabify nil t)))"
  (save-excursion
    (goto-char (point-min))
    (when (search-forward "\t" nil t)
      (untabify (1- (point)) (point-max)))
    nil))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Interface to CDargs
(ska-safe-require 'cdargs)
(add-hook 'cdargs-warped-hook
          '(lambda ()
             (when (file-exists-p
                    (expand-file-name desktop-basefilename "./"))
               (desktop-read))))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Scrolling Using Mouse Dragging
(ska-safe-require 'track-scroll)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Emacs Lisp Documentation
(autoload 'turn-on-eldoc-mode "eldoc" nil t)
;; this works not with standard eldoc.el from SuSE 7.3:
(setq eldoc-minor-mode-string " ed")
;;
(setq eldoc-idle-delay 1.5)
;; is put into the hooks above...
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ MailCrypt
(load-library "mailcrypt") ; provides "mc-setversion"
(mc-setversion "gpg")
(autoload 'mc-install-write-mode "mailcrypt" nil t)
(autoload 'mc-install-read-mode "mailcrypt" nil t)
(add-hook 'mail-mode-hook 'mc-install-write-mode)
```

```
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Auto Save
;; for auto-save-directory see top of file where all the special files
;; and dirs are defined
(setq auto-save-timeout 120)
(setq auto-save-interval 100)
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Low Level Debug
;;this is required by ska-global-keys.el
;;(require 'll-debug)
;; need this for printf output in C++ mode
(setcdr (assq 'c++-mode ll-debug-insert-variable-output-alist)
        (cdr
         (assq 'c-mode ll-debug-insert-variable-output-alist)))

(setcdr (assq 'c++-mode ll-debug-insert-debug-output-alist)
        (cdr
         (assq 'c-mode ll-debug-insert-debug-output-alist)))

;;}}}
;;-------------------------------------------------------------------------
;;{{{ Marky Mode -- doesn't work correctly yet...
;; (require 'marky)
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Highlight Scrolling Context
(load "highlight-context-line")
;; for old version: (setq highlight-context-line-expire 3)
(setq highlight-context-line-background "#c5c500")
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Setnu: Line Numbers
;; please note that I have experienced some problems with this which
;; are not trackable for me. When you get some kind of extent-error
;; when killing lines or creating new lines or something else, try
;; turning setnu off by calling M-x setnu-mode and see if that helps
;; or reopen the file you are just editing. Otherwise XEmacs might
;; crash soon :(
(setq setnu-line-number-format "%4d ")
(setq setnu-line-number-face (copy-face 'default 'setnu-my-face))
(set-face-background 'setnu-my-face "#bfbfbf")
(set-face-foreground 'setnu-my-face "#777777")
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Active Menu
;; The menubar is only shown when the mouse comes to the top of the window
;; my idea, claus' implementation :)
;; see http://www.cbrunzema.de/software.html#active-menu
(ska-safe-require 'active-menu)
(setq active-menu-sensitive-height 10)
(turn-on-active-menu)
;;}}}
;;-------------------------------------------------------------------------
;;{{{ Locate
;; An Emacs interface to the unix locate command
;; taken from the GNU Emacs distribution from Debian (CHB)
(autoload 'locate "locate")
(autoload 'locate-with-filter "locate")
;;}}}
;;-------------------------------------------------------------------------
;;{{{ folding
;; hide and show blocks
```

```
;(load "folding" 'nomessage 'noerror)
;(folding-mode-add-find-file-hook)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Function Menu
;;parse functions for easy navigation
(ska-safe-require 'func-menu)
;; Addition to func-menu: the ruby stuff
;; NOTE: I tried to patch the original but got confused which version
;; loaded at starttime and such....
(defvar fume-function-name-regexp-ruby
  "^\\s-*\\(class\\|def\\)+\\s-*\\([^(\n ]+\\)")
(setq fume-function-name-regexp-alist
      (append
       '((ruby-mode . fume-function-name-regexp-ruby))
       fume-function-name-regexp-alist))

(defun fume-find-next-ruby-function-name (buffer)
  "Searches for the next ruby function in BUFFER."
  (set-buffer buffer)
  (if (re-search-forward fume-function-name-regexp nil t)
      (save-excursion
        (let* ((retpnt (match-beginning 2))
               (retname (buffer-substring retpnt (match-end 2))))
          (goto-char (match-beginning 0))
          (cond ((looking-at "^\\s-*def")
                 (if (re-search-backward
                  "\n?\\s-*class\\s-*\\([A-Z][A-Za-z0-9_]*\\)\\s-*" nil t)
                 (let* ((classname (buffer-substring
                                    (match-beginning 1) (match-end 1))))
                   (if (not (string-match (concat "^" classname "\\.") retname))
                       (setq retname (concat classname "." retname)))))))
          (cons retname retpnt)))))


(setq fume-find-function-name-method-alist
      (append
       '((ruby-mode . fume-find-next-ruby-function-name))
       fume-find-function-name-method-alist))

;;}}}
;;-----------------------------------------------------------------------------
;;{{{ recent files
;;Show last visited files in menubar

(load "recent-files")
(setq recent-files-dont-include
      '("~$" "tmp/." "contact/." "INBOX" ".bbdb" ".newsrc." "places.el"))

;; moved to customize:
;;(setq recent-files-non-permanent-submenu t)
;;(setq recent-files-commands-submenu t)
;; guess you need this one
(setq recent-files-number-of-entries 40)
(recent-files-initialize)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ filladapt
(setq filladapt-mode-line-string " Fa")
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ vertical scrolling in place
;; unfortunately this messes up my VM, so until I found out how to
;; clean that, it stays commented out
```

```
;;(ska-safe-require 'scroll-in-place)
;;(turn-on-scroll-in-place)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ save cursorpositions
(ska-safe-require 'saveplace)
(setq-default save-place t)
(setq save-place-limit 100)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ hippie-expand
;;expand text trying various ways to find its expansion.

(ska-safe-require 'hippie-exp)
(setq hippie-expand-try-functions-list
      '(try-expand-dabbrev
        try-expand-dabbrev-all-buffers
        try-complete-file-name-partially
        try-expand-list
        try-complete-lisp-symbol-partially
        try-expand-line))
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ bbdb
;; the insidous big brother data base -> put into vm-init-file
;;(ska-safe-require 'bbdb)
;;(bbdb-initialize 'vm 'message 'sendmail)
(setq bbdb/news-auto-create-p 'bbdb-ignore-most-messages-hook
      bbdb/mail-auto-create-p 'bbdb-ignore-most-messages-hook
      bbdb-ignore-most-messages-alist
      '(("To" . "home@skamphausen\\.de")
        ("Subject" . "\\(emacs|cdargs|\\.el\\)"))
      bbdb-ignore-some-messages-alist
      '(("From" . "daemon")
        ("From" . "abuse@")
        ("From" . "news@")
        ("From" . "cron")
        ("From" . "root@")
        ("From" . "noreply@")
        ("From" . "prontomail")
        ("From" . "majordomo")
        ("From" . "postmaster")
        ("From" . "quota")
        ("Subject" . "MAILER DAEMON")))
(setq bbdb-notice-hook 'bbdb-auto-notes-hook)
(setq bbdb-auto-notes-alist
      '(("Organization" (".*" company 0))
        ("Newsgroup" ("[^,]+" newsgroups 0))
        ("Subject" (".*" last-subj 0 t))
        ("User-Agent" (".*" mailer 0))
        ("X-Newsreader" (".*" mailer 0))
        ("X-gpg-key-ID" (".*" key 0))
        ("X-Mailer" (".*" mailer 0))
        ("X-URL" (".*" url 0))
        ;;("X-Face" (".+" face 0 'replace))
        ;;("Face" (".+" cface 0 'replace))
        ))
(setq bbdb-default-area-code "0551"
            bbdb-north-american-phone-numbers-p nil
            bbdb-complete-name-allow-cycling t)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ letter-template
(setq texletter-template-directory "~/text/tex/LetterTemplates")
```

```
(setq texletter-letter-directory   "~/text/tex")
(autoload 'texletter-create-letter
  "texletter"
  "Create LaTeX letters from templates."
  t)
(autoload 'texletter-install-templates-as-menu
  "texletter"
  "Adding menu items for all templates."
  t)
(texletter-install-templates-as-menu)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ resize-minibuffer-mode
;; makes the minibuffer automatically resize when _typing_ text. This
;; is not for displaying larger information
(autoload 'resize-minibuffer-mode "rsz-minibuf" nil t)
(resize-minibuffer-mode 1)
(setq resize-minibuffer-window-exactly nil)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ minibuffer-complete: Cycle completions in minibuffer
(load "minibuffer-complete-cycle")
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Speedbar
(autoload 'speedbar-frame-mode "speedbar" "Popup a speedbar frame" t)
(autoload 'speedbar-get-focus "speedbar" "Jump to speedbar frame" t)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ AutoInsert
;; automatic insertion of text into new files.
;; Find appropriate skeletons for various modes.
(load "ska-auto-insert" nil nil nil)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ Jiggle The Cursor
;; http://www.eskimo.com/~seldon/jiggle.el
;; I changed the modeline minor mode marker in the .el file
;; Problem is that this plugs into the wrong hook. It is called
;; whenever a buffer is selected but I want it to only jiggle when the
;; buffer is shown (compare chb-next-buffer defun)
;(require 'jiggle)
;(setq jiggle-sit-for-how-long .05)
;(setq jiggle-how-many-times 2)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ counter
;; http://www.eskimo.com/~seldon/counter.el
;;(autoload 'counter "counter" nil t)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ ispell

;; Es gibt in Version >20 ein Problem mit ispell, dass eine Funktion
;; nicht definiert ist :-(
;; Dieses soll wohl ersma laufen:
;(if (not (fboundp 'set-process-coding-system))
;    (defun set-process-coding-system (foo bar)
;      (print "Still an ispell error")
;     ))
;; which dictionary
(setq-default ispell-dictionary "american")
;;}}}
;;-----------------------------------------------------------------------------
```

```
;;{{{ gnuserv
;; Xemacs acts as a server for editing sessions
;; (unless (gnuserv-running-p) (gnuserv-start))
;;}}}
;;-----------------------------------------------------------------------
;;{{{ efs
;; remote file editing
;(ska-safe-require 'efs)
;(setq efs-default-user "username")
;(efs-set-user "ftp.yourdomain.org" "yourusername")
;(efs-display-ftp-activity)
;;}}}
;;-----------------------------------------------------------------------
;;{{{ vkill
(autoload 'vkill "vkill" nil t)
(autoload 'list-unix-processes "vkill" nil t)
;;}}}
;;-----------------------------------------------------------------------
;;{{{ fff
(autoload 'fff-find-file-in-envvar-path                  "fff" nil t)
(autoload 'fff-insert-file-in-envvar-path                "fff" nil t)
(autoload 'fff-find-file-in-exec-path                    "fff" nil t)
(autoload 'fff-insert-file-in-exec-path                  "fff" nil t)
(autoload 'fff-find-file-in-path                         "fff" nil t)
(autoload 'fff-insert-file-in-path                       "fff" nil t)
(eval-after-load "fff" '(fff-install-map))
;;}}}
;;-----------------------------------------------------------------------
;;}}}
;;=======================================================================

;;=======================================================================
;;{{{ MISCELLEANOUS SETTINGS
;;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;;{{{ Important Hooks
(add-hook 'after-save-hook
        #'(lambda ()
        (and (save-excursion
               (save-restriction
                 (widen)
                 (goto-char (point-min))
                 (save-match-data
                   (looking-at "^#!"))))
             (not (file-executable-p buffer-file-name))
             (shell-command (concat "chmod u+x " buffer-file-name))
             (message
              (concat "Saved as script: " buffer-file-name)))))
;;}}}
;;-----------------------------------------------------------------------
;;{{{ tiny stuff
;; no splash screen
(setq-default inhibit-startup-message t)
;; y instead of  y e s  is enough
(fset 'yes-or-no-p 'y-or-n-p)
;; delete key != backspace
(setq delete-key-deletes-forward t)
;; file name in icon
(setq frame-title-format (list mode-line-buffer-identification))
;; Timestamp within the first `time-stamp-line-limit' lines
;; Format: Time-stamp: <>
(setq time-stamp-active t)
(setq time-stamp-format "%02d-%3b-%:y %02H:%02M:%02S %u")
(setq time-stamp-line-limit 16)
;; für Menueintrag Edit->Text Properties
```

```
(ska-safe-require 'facemenu)
;; browser
;;(setq-default browse-url-browser-function 'browse-url-netscape)
;; The default mode should not just be fundamental
(setq default-major-mode 'text-mode)
;; flash only top and bottom line
(setq visible-bell 'top-bottom)
;;}}}
;;-----------------------------------------------------------------------------
;;{{{ other variables
;; ignore files when completing with TAB
(setq completion-ignored-extensions
      (append '(".aux" ".dvi" ".ps" ".bbl" ".blg" ".o" ".class")
              completion-ignored-extensions)
      )

;; Tune Modeline To My Taste:
;; ========================
;;
;; The Problem is that modes that change the modeline rely on it to be
;; in the original format (e.g. fume-toggle-modeline-display)
;;
;; line and column number before file name (so that they don't get
;; lost for long fume-displays)
;; and remove the "Xemacs:"-string...
;;(defvar my-modeline-buffer-identification
;;  (list (cons modeline-buffer-id-left-extent ":")
;;        (cons modeline-buffer-id-right-extent " %17b"))
;;  "My modification of `modeline-buffer-identification': removed the
;;  \"XEmacs\" from it.")
;; if I remove XEmacs, fume doesn't work anymore :(
;(setq default-modeline-format
;      (list
;       ""
;       (cons modeline-modified-extent 'modeline-modified)
;       (list 'line-number-mode "(%l,")
;       (list 'column-number-mode "%c) ")
;       (cons modeline-buffer-id-extent 'modeline-buffer-identification)
;       "  "
;       'global-mode-string
;       "   %[("
;       (cons modeline-minor-mode-extent
;             (list "" 'mode-name 'minor-mode-alist))
;       (cons modeline-narrowed-extent "%n")
;       'modeline-process
;       ")%]---"
;       (cons -3 "%p")
;       "-%-"))
;;}}}
;;-----------------------------------------------------------------------------
(require 'mwheel)
(mwheel-install)


;;}}}
;;=============================================================================


;; Local variables:
;; mode: emacs-lisp
;; folded-file: nil
;; end:
```

A.2. ska-local-keys.el

```
;;; ska-local-keys.el --- my own major mode dependend key-bindings
;; Copyright (C) 2000-2002 Stefan Kamphausen

;; Author: Stefan Kamphausen <mail@skamphausen.de>
;; Time-stamp: <17-Feb-2004 11:57:59 ska>

;; Keywords:
;; This file is not part of XEmacs.

;; This program is free software; you can redistribute it and/or modify it
;; under the terms of the GNU General Public License as published by
;; the Free Software Foundation; either version 2, or (at your option)
;; any later version.

;; This program is distributed in the hope that it will be useful, but
;; WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
;; General Public License for more details.

;; You should have received a copy of the GNU General Public License
;; along with this program; see the file COPYING. If not, write to the Free
;; Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
;; 02111-1307, USA.


;;; Commentary:
;; In this file we define defuns for loading in each mode specific
;; hook. Each defun follows the naming convention
;; ska-<major-mode-name>-keys
;; Note that skeleton template key-bindings will be loaded from this
;; file, too.

;;; To use:
;;  1. (load "ska-local-keys" t nil)
;;           reads this file
;;  2. For each mode you need to call the appropriate defun in the
;;     hook, e.g:
;; (add-hook 'c-mode-hook
;;        '(lambda ()
;;           ;; my keybindings
;;           (ska-c-mode-keys c-mode-map)
;;           ))

;;; Code:
;; My own prefix for globally available commands: C-b
;; (just like C-x and C-c for xemacs' defaults)
(global-unset-key '[(control b)])

;; Newsgroup says it would be best to use a different modifier (Hyper,
;; Super) as personal prefix.

;; Maybe this is better:?
;(define-prefix-command 'user-mode-specific-prefix t)
;(defvar user-mode-specific-map (symbol-function
;'user-mode-specific-prefix) "\
;The default keymap for all mode specific actions defined by the
;user.")
;(define-key global-map '[(control b)] 'user-mode-specific-prefix)

;;{{{ Programming Keys For All Modes
;;---------------------------------
(defun ska-coding-keys (map)
```

```
  "Sets the key bindings which shall be available in all programming
languages. Argument MAP is the local keymap (e.g. cperl-mode-map)."
  (define-key map '[(return)]                'newline-and-indent)
  (define-key map '[(control b) (d)]         'chb-insert-debug-output)
  (define-key map '[(control b) (\;)]        'chb-comment-region-or-line)
  (define-key map '[(control b) (\:)]        'chb-uncomment-region-or-line)
  (define-key map '[(control b) (c)]         'chb-copy-and-comment-region-or-line)
  )
;;}}}

;;{{{ C and C++ Mode
;;------------------
(ska-safe-require 'ska-skel-c)
;; This defun is slightly different from the others since it works for
;; two major modes. Therefore the map to use is passed
(defun ska-c-common-mode-keys (map)
  "Set my personal keys for C and C++.
Argument MAP is c-mode-map or c++-mode-map."
  (define-key map '[(meta tab)]                      'hippie-expand)
  (define-key map '[(control b) (control a)]         'ska-edit-automake-file)
  (define-key map '[(control b) (control b)]         'compile)
  (define-key map '[(control b) (control f)]         'fume-rescan-buffer)
  (define-key map '[(control b) (control m)]         'manual-entry)
  ;; macros, templates, skeletons:
  (define-key map '[(control b) (m) (f) (f)]         'ska-skel-c-fflush)
  (define-key map '[(control b) (m) (f) (p)]         'ska-skel-c-fprintf)
  (define-key map '[(control b) (m) (i)]             'ska-skel-c-include)
  (define-key map '[(control b) (m) (m)]             'ska-skel-c-main)
  (define-key map '[(control b) (m) (c)]             'ska-skel-c-comment)
  (define-key map '[(control b) (m) (n)]             'ska-skel-c-printf-newline)
  (define-key map '[(control b) (m) (p)]             'ska-skel-c-printf)
  (define-key map '[(control b) (m) (P)]             'ska-skel-c-printf-flush)
  (define-key map '[(control b) (m) (l) (w)]         'ska-skel-c-loop-while)
  )

(defun ska-c++-mode-keys ()
  "Set my personal keys for C++ mode."
  (local-set-key '[(control b) (m) (c)]             'ska-skel-cc-class)
  (local-set-key '[(control b) return]              'ska-skel-cc-endl)
  (local-set-key '[(control b) (m) (l) (f)]         'ska-skel-cc-loop-for)
  )
(defun ska-c-mode-keys ()
  "Set my personal keys for plain C mode."
  (local-set-key '[(control b) (m) (c)]             'ska-skel-c-comment)
  (local-set-key '[(control b) return]              'ska-skel-cc-endl)
  (local-set-key '[(control b) (m) (l) (f)]         'ska-skel-c-loop-for)
  )
;;}}}

;;{{{ Perl Mode
;;-------------
(require 'ska-skel-perl)
(defun ska-cperl-mode-keys ()
  "Setting local keybindings for major mode: perl."
  (local-set-key '[(meta tab)]               'hippie-expand)
  (local-set-key '[(control b) (control b)]  'executable-interpret)
  ;; skeletons and makros MIGHT use C-b C-s as prefix if they get too many
  (local-set-key '[(control b) (control f)]  'ska-skel-perl-file-loop)
  (local-set-key '[(control b) (control s)]  'ska-skel-perl-sub)
  (local-set-key '[(control b) (control i)]  'ska-skel-perl-prog-id)
  (local-set-key '[(control b) (control o)]  'ska-skel-perl-options)
  )
;;}}}
```

```
;;{{{ Ruby Mode
;;-------------
(require 'ska-skel-ruby)
(defun ska-ruby-mode-keys ()
  "Setting local keybindings for major mode: Ruby."
  (local-set-key '[(meta tab)]               'hippie-expand)
  (local-set-key '[(control b) (control b)]  'run-ruby)
  (local-set-key '[(control b) (control e)]  'ruby-insert-end)
  (local-set-key '[(control b) (control d)]  'ska-skel-ruby-def)
  (local-set-key '[(control b) (control c)]  'ska-skel-ruby-class)
  (local-set-key '[(control b) (control h)]  'ska-skel-ruby-header)
  (local-set-key '[(control b) (control s)]  'ska-skel-ruby-string-subst)
  (local-set-key '[(control b) (a)]          'ska-skel-ruby-array-new)
  (local-set-key '[(control b) (h)]          'ska-skel-ruby-hash-new)
  (local-set-key '[(control b) (d)]          'ska-skel-ruby-do-with-args)
  )
;;}}}

;;{{{ SQL Mode
;;------------
(require 'ska-skel-sql)
(defun ska-sql-mode-keys ()
  "Setting local keybindings for major mode: SQL."
  (local-set-key '[(control b) (control v)] 'ska-skel-sql-view)
  (local-set-key '[(control b) (control v)] 'ska-skel-sql-build)
  )
;;}}}

;;{{{ LaTeX Mode
;;--------------
(defun ska-latex-mode-keys ()
  "Set keys for latex-mode (AUCTeX)."
  (local-set-key '[(tab)]                    'LaTeX-indent-line)
  (define-key LaTeX-math-keymap
    (concat LaTeX-math-abbrev-prefix "/")    'LaTeX-math-frac)
)
(defun ska-bibtex-mode-keys ()
  "Set keys for bibtex-mode."
  (local-set-key '[(control return)]         'bibtex-next-field)
)
;;}}}

;;{{{ Matlab Mode
(ska-safe-require 'ska-skel-matlab)
(defun ska-matlab-mode-keys ()
  "Set keys for matlab-mode."
  (local-set-key '[(control b) (m) (f)]   'ska-skel-matlab-function)
  )
;;}}}

;;{{{ Emacs Lisp Mode
;;-------------------
(defun ska-elisp-mode-keys ()
  "Set keys for elisp-mode."
  (local-set-key '[(control b) (control b)]  'eval-buffer)
  (local-set-key '[(control b) (control d)]  'll-debug-insert-debug-output)
    )
;;}}}

;;{{{ VM and Mail
;;------
(require 'ska-skel-mail)
(defun ska-vm-keys ()
  "Set keybindings in VM Mailreader."
```

```
  (local-set-key '[(a)]                     'vm-toolbar-autofile-message)
)
(defun ska-mail-keys ()
  "Set keybindings for mail writing buffers."
  (local-set-key '[(control b) (control r)]  'ska-skel-mail-regards)
  (local-set-key '[(control b) (control k)]  'ska-skel-mail-coffee-or-tee)
  (local-set-key '[(control b) (control b)]  'ska-mail-insert-answer-gap)
  )
;;}}}

;;{{{ Speedbar
;;------------
(defun ska-speedbar-keys ()
  "Set keybindings in major navigation tool Speedbar."
  (define-key speedbar-key-map '[(space)]    'ska-speedbar-toggle-expand)
  )
;;}}}

;;{{{ Makefile
(defun ska-makefile-keys ()
  "Set keybindings for makefile-mode."
  (local-set-key '[(control b) (control b)]   'compile)
  )
;;}}}

;;{{{ SGML, XML and HTML
;;----------------------
(defun ska-sgml-keys ()
  "Set keybindings for all modes done by PSGML."
  (local-set-key '[(control c) (control c)]  'sgml-insert-element)
  (local-set-key '[(control c) (a)]          'sgml-insert-attribute)
  (local-set-key '[(return)] 'newline-and-indent)
  )
;;}}}

;;{{{ PCL-CVS
(defun ska-cvs-keys ()
  "Set keybindings for CVS mode."
  (local-set-key '[(G)] 'cvs-update)
  (local-set-key '[(S)] 'cvs-status)
  (local-set-key '[(Q)] 'cvs-examine)
  (local-set-key '[(c)] 'cvs-mode-commit))
;;}}}

(provide 'ska-local-keys)

;;; local-keys.el ends here
;; Local variables:
;; mode: emacs-lisp
;; folded-file: t
;; end:
```

A.3. ska-global-keys.el

```
;;; ska-global-keys.el --- global keybindings
;; Copyright (C) 2000-2002 Stefan Kamphausen <mail@skamphausen.de>

;; Author: Stefan Kamphausen <mail@skamphausen.de>
;; Time-stamp: <25-May-2004 22:27:22 ska>

;; Keywords:
;; This file is not part of XEmacs.
```

```
;; This program is free software; you can redistribute it and/or modify it
;; under the terms of the GNU General Public License as published by
;; the Free Software Foundation; either version 2, or (at your option)
;; any later version.

;; This program is distributed in the hope that it will be useful, but
;; WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
;; General Public License for more details.

;; You should have received a copy of the GNU General Public License
;; along with this program; see the file COPYING. If not, write to the Free
;; Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
;; 02111-1307, USA.


;;; Commentary:
;; This sets the keybindings I want to have available in all modes.

;;; To Use:
;; put
;;    (load "ska-global-keys" t nil)
;; in an init file

;;; Code:
(require 'ska-utils)
(require 'chb-util)
(require 'll-debug)
;; My own prefix for globally available commands: C-v
;; (just like C-x and C-c for xemacs' defaults)
(global-unset-key '[(control v)])



;;{{{ Navigation And Movement
;; ---------------------------
;; Scrolling with keys
(global-set-key '[(control up)]           '(lambda ()
                                             (interactive)
                                             (scroll-down 1)))
(global-set-key '[(control down)]         '(lambda ()
                                             (interactive)
                                             (scroll-up 1)))
(global-set-key '[(control \.)]           'ska-point-to-register)
(global-set-key '[(control \,)]           'ska-jump-to-register)
(global-set-key '[(control \')]           'point-to-register)
(global-set-key '[(meta \')]              'register-to-point)
(global-set-key '[(control backspace)]    'backward-kill-word)
(global-set-key '[(control delete)]       'kill-word)
(global-set-key '[(shift prior)]          'chb-next-buffer)
(global-set-key '[(shift next)]           'chb-previous-buffer)
(global-set-key '[(home)]                 'chb-home)
(global-set-key '[(end)]                  'chb-end)
(global-set-key '[(control v) (control s)] 'speedbar-get-focus)
(global-set-key '[(control v) (control y)] 'ska-insert-primary-selection)
;;}}}

;;{{{ Mouse
;; ---------
;; The Hyperbole information manager package uses (shift button2) and
;; (shift button3) to provide context-sensitive mouse keys.  If you
;; use this next binding, it will conflict with Hyperbole's setup.
;; Choose another mouse key if you use Hyperbole.
```

```
;; CONTROL:
;;(global-set-key '[(control button1)]       'fume-mouse-function-goto)
;; still undefined ?!?
;;(global-set-key '[(control button3)]       'fume-mouse-function-goto)
;; SHIFT: menus
(global-set-key '[(shift button2)]         'modeline-buffers-menu)
(global-set-key '[(shift button3)]         'mouse-function-menu)
;; META: rectangle operations. meta button1 is default
(global-set-key '[(meta button2)]          'yank-rectangle)
(global-set-key '[(meta button3)]          'kill-rectangle)
;;}}}

;;{{{ Function Keys
;; -----------------

;; keyboard macros
(global-set-key '[(f1)]                    'call-last-kbd-macro)
(global-set-key '[(shift f1)]              'chb-start-kbd-macro)
;; bookmarks
(global-set-key '[(f2)]                    'bookmark-set)
(global-set-key '[(shift f2)]              'list-bookmarks)
(global-set-key '[(f3)]                    'bookmark-jump)
;; just to have something on shift F3 *gg*
(global-set-key '[(shift f3)]              'list-buffers)

;; Buffer and Window operations
(global-set-key '[(f4)]                    'kill-this-buffer)
(global-set-key '(shift f4)                'ska-kill-this-window)
;; remove the newly opened (help) buffer but leave windows in state
(global-set-key '(control f4)              '(lambda ()
                                              (interactive)
                                              (kill-this-buffer)
                                              (other-window -1)))
(global-set-key '[(f5)]                    'delete-other-windows)
(global-set-key '[(shift f5)]              'delete-window)
(global-set-key '[f6]                      'split-window-vertically)
(global-set-key '[(shift f6)]              'split-window-horizontally)
(global-set-key '[f7]                      'shrink-window)
(global-set-key '[(shift f7)]              'shrink-window-horizontally)
(global-set-key '[f8]                      'enlarge-window)
(global-set-key '[(shift f8)]              'enlarge-window-horizontally)
(global-set-key '[(control f7)]            '(lambda ()
                                              (interactive)
                                              (set-frame-height
                                               (selected-frame)
                                               (- (frame-height) 1))))
(global-set-key '[(control f8)]            '(lambda ()
                                              (interactive)
                                              (set-frame-height
                                               (selected-frame)
                                               (+ (frame-height) 1))))
;; The function keys with higher numbers seem to be used by some
;; packages so I won't rely on them
(global-set-key '[(shift f9)]              'repeat-complex-command)
;;}}}

;;{{{ All Other C-v (and more) settings
;; ---------------------------------------
(global-set-key '[(control v) (control a)] 'chb-align-to-char-in-previous-line)
(global-set-key '[(control v) (control c)] 'set-justification-center)
(global-set-key '[(control v) (control f)] 'fill-region)
(global-set-key '[(control v) (control l)] 'locate)
(global-set-key '[(control v) (control m)] 'vm)
(global-set-key '[(control v) (control n)] 'gnus)
```

```
(global-set-key '[(control v) (control o)] 'occur)
(global-set-key '[(control v) (control p)] 'ska-insert-path)
(global-set-key '[(control v) (control r)] 'chb-query-replace-by-example)
(global-set-key '[(control v) (control t)] 'ska-insert-current-time-string)
(global-set-key '[(control v) (control v)] 'll-debug-toggle-comment-region-or-line)
(global-set-key '[(control v) (control x)] 'ska-insert-exec-text)
(global-set-key '[(control v) (f)]         'ska-expand-file-name)
(global-set-key '[(control v) (g)]         'fume-prompt-function-goto)
(global-set-key '[(control v) (l)]         'add-change-log-entry)
(global-set-key '[(shift space)]           'dabbrev-expand)
(global-set-key '[(control z)]             'yank)
(global-set-key '[(control \`)]            'accelerate-menu)
(global-set-key '[(meta k)]                'kill-entire-line)
(global-set-key '[(iso-left-tab)]          'comint-dynamic-complete)
(global-set-key '[(control t)]             'ska-electric-transpose-chars)

;;}}}


;;{{{ Heavy Default Overriding
(define-key read-file-name-must-match-map "~" 'ska-minibuffer-electric-tilde)
(define-key read-file-name-map "~"            'ska-minibuffer-electric-tilde)
;;}}}

;;{{{ A New Keyboard Layer
;; ------------------------
(global-set-key '[(super q)] '(lambda () (interactive) (insert "7")))
(global-set-key '[(super w)] '(lambda () (interactive) (insert "8")))
(global-set-key '[(super e)] '(lambda () (interactive) (insert "9")))
(global-set-key '[(super a)] '(lambda () (interactive) (insert "4")))
(global-set-key '[(super s)] '(lambda () (interactive) (insert "5")))
(global-set-key '[(super d)] '(lambda () (interactive) (insert "6")))
(global-set-key '[(super z)] '(lambda () (interactive) (insert "1")))
(global-set-key '[(super x)] '(lambda () (interactive) (insert "2")))
(global-set-key '[(super c)] '(lambda () (interactive) (insert "3")))
;; brackets and co.
(global-set-key '[(super f)] '(lambda () (interactive) (insert "(")))
(global-set-key '[(super g)] '(lambda () (interactive) (insert "[")))
(global-set-key '[(super h)] '(lambda () (interactive) (insert "]")))
(global-set-key '[(super j)] '(lambda () (interactive) (insert ")")))
(global-set-key '[(super v)] '(lambda () (interactive) (insert "{")))
(global-set-key '[(super n)] '(lambda () (interactive) (insert "}")))
;; misc
(global-set-key '[(super r)] '(lambda () (interactive) (insert "+")))
(global-set-key '[(super t)] '(lambda () (interactive) (insert "*")))
(global-set-key '[(super y)] '(lambda () (interactive) (insert "%")))
(global-set-key '[(super u)] '(lambda () (interactive) (insert "@")))
(global-set-key '[(super b)] '(lambda () (interactive) (insert "$")))
(global-set-key '[(super space)] '(lambda () (interactive) (insert "_")))
;; sequences
(global-set-key '[(super i)] '(lambda () (interactive) (insert "->")))
(global-set-key '[(super o)] '(lambda () (interactive) (insert "first")))
(global-set-key '[(super p)] '(lambda () (interactive) (insert "second")))
(global-set-key '[(super k)] '(lambda () (interactive) (insert "unsigned")))
(global-set-key '[(super l)] '(lambda () (interactive) (insert "|")))
(global-set-key '[(super m)] '(lambda () (interactive) (insert "frei")))
;; umlauts
(global-set-key '[(super \[)] '(lambda () (interactive) (insert "ä")))
(global-set-key '[(super \])] '(lambda () (interactive) (insert "ö")))
(global-set-key '[(super \\)] '(lambda () (interactive) (insert "ü")))
(global-set-key '[(super \{)] '(lambda () (interactive) (insert "Ä")))
(global-set-key '[(super \})] '(lambda () (interactive) (insert "Ö")))
(global-set-key '[(super \|)] '(lambda () (interactive) (insert "Ü")))
;;}}}
```

```
(provide 'ska-global-keys)

;;; ska-global-keys.el ends here
```

B. Lizenz

Ich stelle diese Dokumentation unter der GFDL bereit. Im Folgenden wird der (englische) Text der GFDL wiedergegeben. Ich habe dazu die auf [7] angebotene LaTeX-Version verwendet, die sich nach meinem Dafürhalten etwas seltsam verhält, indem sie selber das Inhaltsverzeichnis erzeugt und ihre Überschriften formatiert. Da ich aber nicht geneigt bin, genau zu prüfen, ob ich die *Formatierung* ändern darf, gebe ich das hier so wie es ist wieder.

B.1. GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been

arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

C. Dank

- Dank allen Entwicklern sowohl des XEmacs als auch des Gnu Emacs für den besten Editor der Welt.
- Dank an Claus Brunzema ([2]) für stetiges Korrekturlesen und Hilfe in Lisp-Geschichten.
- Dank an Ralf Angeli für Korrekturen und Hinweise zu meinem LaTeX-Code sowie zu AUCTeX.
- Dank meiner Familie für die Zeit, die ich hierfür hatte.

Bibliography

[1] Claus Brunzema: ActiveMenu – Eine Zeile mehr Screen Real Estate. http://www.cbrunzema.de/software.html\#active-menu, Wozu sollte man die eine Zeile Code da oben all die Zeit vergeben, wenn man das Menu doch so selten braucht? 9.6

[2] Claus Brunzema: Persönliche Webseite von Claus Brunzema. http://www.cbrunzema.de. C

[3] Claus Brunzema: Picklist – ein ersatz für recent-files. http://www.cbrunzema.de/software.html\#picklist, Enstanden aus einem Mangel in recent-files mit vielen netten Features. 7.3

[4] Carsten Dominik: RefTeX – Referenzen mit LaTeX. http://strw.leidenuniv.nl/~dominik/Tools/, RefTeX ist ein ELisppaket zur Verwaltung von Referenzen und Zitaten im XEmacs. 3.1.1

[5] FSF Free Software Foundation: FSF - Free Software Foundation Website. http://www.gnu.org/.

[6] FSF Free Software Foundation: GPL - GNU General Public License. http://www.gnu.org/copyleft/gpl.html, 1991.

[7] FSF Free Software Foundation: FDL - GNU Free Documentation License. http://www.gnu.org/copyleft/, 2000. (document), B

[8] Bruce Ingalls: EMacro – Ein Setup für beide Emacsen. http://emacro.sourceforge.net, Hier wird ein Setup vorgestellt, das nicht nur beide Emacsen bedient sondern auch so ziemlich alle Pakete konfiguriert. 4.5

[9] Kyle Jones: VM – View Mail. http://www.wonderworks.com/vm, Der Mailreader VM: ein ausgewachsenes Mailprogramm, wenn man noch nicht zu Gnus will. 3.13.2

[10] Stefan Kamphausen: CDargs – Bookmarks für die Shell. http://www.skamphausen.de/software/cargs. 9.4

[11] Stefan Kamphausen: Persönliche Webseite von Stefan Kamphausen. http://www.skamphausen.de, Hier finden Sie meine Beiträge zur Welt der Freien Software, meine Grafiken, wissenschaftlichen Publikationen sowie diese XEmacs-Einleitung., 2000-2004.

[12] Stefan Kamphausen und Sebastian Freundt: MTorus – Mehr als nur Cursorpositionen. http://mtorus.berlios.de, MTorus kann viel mehr als nur Cursorpositionen speichern und zielsicher anspringen. Die alte Version findet sich noch auf http://www.skamphausen.de/software/skamacs/mtorus. 9.5

[13] Michael Kofler: *Linux*. Addison Wesley, 2004. 32

[14] Dave Pearson: Sawfish-Mode. http://www.davep.org/emacs/\#sawfish.el, Die direkte Manipulation des Sawfish Windowmanagers vom XEmacs aus. 3.11

[15] Johannes Plass und Tim Theisen: GV – Ein Postscriptbetrachter. http://wwwthep.physik.uni-mainz.de/~plass/gv/, Ein ansprechender Postscriptbetrachter auf Basis von Ghostview. 9.1

[16] Stefan Reichoer: PSVN. http://www.xsteve.at/prg/emacs/psvn.el, XEmacsfrontend zur Subversion Versionsverwaltung. 17

[17] Viele Freiwillige: Debian Linux Distribution. http://www.debian.org.

[18] Morten Welinder und andere: Das Desktop-Paket für den XEmacs. Ist Bestandteil der XEmacsinstallation., Stellt alte Sitzungen wieder her. 7.4

Index