

Vim

Swaroop C H
(Übersetzung von Christoph Koydl)

17. Februar 2010

Einführung

„Ein Byte Vim“ ist ein Buch, welches darauf abzielt, Ihnen dabei zu helfen, wie man den Vim-Editor (Version 7) benutzt, selbst wenn Sie lediglich wissen, wie man eine Tastatur bedient.



Der erste Teil dieses Buches richtet sich an neue Benutzer, die verstehen wollen was Vim ist, und die lernen wollen, wie man ihn benutzt.

Der zweite Teil des Buches ist für Leute, die schon wissen, wie man den Vim bedient, und die etwas über die Funktionen wissen möchten, die den Vim so leistungsstark machen, z.B. Fenster und [Tabs](#), persönliche Informationsverwaltung, wie man einen 'Editor für Programmierer' daraus macht, wie Sie den Vim mit Ihren eigenen [Plugins](#) erweitern, und vieles mehr.

Lesen Sie das Buch jetzt

Jetzt können Sie das ganze Buch direkt übers Internet lesen.

Wenn Sie irgendwelche 'Vertipper' oder Rechtschreibfehler finden, fühlen Sie sich ermutigt die Dinge selbst zu bereinigen, mit einem Klick auf den 'Editieren'-Link an der linken Seitenleiste!

Das Buch kaufen

Eine, auf Papier gedruckte Fassung des Buches, können Sie, für die Zeit, in der Sie mal nicht vor dem Bildschirm sitzen, auch käuflich erwerben¹, und gleichzeitig unterstützen Sie damit die fortlaufende Weiterentwicklung und Verbesserung dieses Buches.

¹<http://www.swaroopch.com/buybook>

Oder Sie ziehen eine Spende, als Alternative hierzu, in Erwägung².

Herunterladen des Buches

PDF (1,5MB)³

Mediawiki XML dump (197K)⁴ (nur für fortgeschrittene Benutzer)

Was die Leser sagen

„Gut gemacht!!! Ich benutze den Vim erst seit zwei bis drei Wochen, und ich muss sagen, für einen Anfänger wie mir, ist das perfekt!!!“ –Jay⁵

„Das Buch ist sehr gut, und macht auch Spaß zu lesen. Danke fürs kostenlose Verteilen.“ –Yosi Izaq⁶

„Ihre Bücher sollten sich wie warme Semmeln verkaufen, so wie Sie sie präsentieren.“ –Deepak⁷

„Hochachtung! Danke - da steckt viel harte Arbeit drin. Besonders nett finde ich, dass der Anfang den Leser langsam an den Stoff heranzuführt. Nachdem ich den Vim nun ein paar Jahre benutze, habe ich inzwischen vergessen, wie seltsam mir das zuerst schien, und ich könnte das einem anderen wahrscheinlich nicht so gut erklären. Ich werde Ihr Buch sicherlich weitergeben, um die VimPropaganda voranzutreiben. ;-)“ –Joseph Sullivan⁸

Was ich versuche zu sagen ist, wenn Sie grundlegende Computerkompetenzen besitzen, sollten Sie den Vim sofort auf Ihrem Rechner installieren und Ihr Leben verbessern. Am besten lernen Sie den Vim kennen, wenn Sie Swaroop C H's phantastisches eBuch Ein Byte Vim lesen. Es wird Ihre Einstellung, wie Sie in Zukunft über Editoren denken, revolutionieren. –wooden nickels⁹

Habe mal 'Ein Byte Vim' durchgeblättert, und einen ganzen Haufen dabei gelernt, obwohl ich den Vim schon seit Jahren benutze. –Josh Nichols¹⁰

Starkes Buch !! Obwohl ich den Vim jeden Tag sowohl als Editor, wie auch als integrierte Entwicklungsumgebung benutze, dieses Buch macht dir klar, wie viel mehr er kann. –Raseel Bhagat¹¹

Wunderbar! Auf dieses Buch hat man nun wirklich gewartet. Ich war ein Vim-Anwender in den vergangenen Jahren, aber so viel Hilfestellung habe

²<http://www.swaroopch.com/byteofdonate>

³http://www.swaroopch.com/files/byteofvim/byte_of_vim_v050.pdf

⁴http://www.swaroopch.com/files/byteofvim/byte_of_vim_v050.xml

⁵http://groups.google.com/group/vim_use/msg/e1625069d4ea0ef9

⁶http://groups.google.com/group/vim_use/msg/09ca306a67b9d2cd

⁷<http://twitter.com/peerlessdeepak/status/1024279089>

⁸http://groups.google.com/group/vim_use/msg/362a82a4af132317

⁹<http://woodennickels.posterous.com/text-editing-your-way-to-heave>

¹⁰<http://twitter.com/techpickles/status/1025775542>

¹¹<http://twitter.com/raseel/status/1024291090>

ich noch nie in einem Buch gesehen! Danke für das Buch Swaroop! –Hiran Venugopalan¹²

Was für ein schönes Buch. Ich bin ein Langzeit-Vim-Anwender, aber das Thema 'Vim-Scripting' hab ich nie so richtig in die Birne gekriegt (abgesehen vom Beseitigen einiger Programmierfehler in Skripten anderer). Das ist die beste Einführung zum Thema 'Vim-Scripting' (Plugins schreiben, Syntax-Dateien,), die ich bis jetzt gesehen habe. Danke für die Bereitstellung im Internet! –namenlos (132.230.122.35)¹³

Danke Swaroop! Ich habe damit angefangen das Buch zu lesen, und ich muß sagen, es ist sehr gut geschrieben. Und ich habe keinerlei Zweifel daran, daß die großartige Gemeinschaft von uns 'VimAnwendern' das Programm durch Fehlerbeseitigungen, Zusätze oder kleine Korrekturen, verbessern werden. Das 'Wiki-Format' ist eine riesige Idee. –Eduard Fabra¹⁴

Und:

Das Buch wurde als Top-Tipp im Dezember 2008 im 'Offiziellen-Vim-Tipps-Wiki'¹⁵ verzeichnet.

Lizenz und Bestimmungen

1. Dieses Buch steht unter der creative-commons attribution-share alike 3.0 unported-Lizenz¹⁶
 - Das bedeutet:
 - Sie können das Buch mit anderen teilen, d.h. kopieren, verteilen und übertragen
 - Sie können es neu zusammenstellen, d.h. Angleichen
 - Unter den folgenden Bedingungen:
 - Zuordnung. Sie müssen dieses Werk, in der vom Urheber oder Lizenzgeber bezeichneten Weise kenntlich machen (jedoch nicht dergestalt, dass man den Eindruck gewinnt, dass man Sie oder die Tatsache, dass Sie das Buch gebrauchen, unterstützt.)
 - Share Alike: Wenn Sie das Buch verändern, transformieren, oder das Werk als Grundlage nutzen, dürfen Sie das neu erschaffene Werk nur unter derselben oder einer ähnlichen Lizenz wie dieser hier weitergeben.
 - Bei jedem Wiederverwenden oder Verteilen, müssen Sie anderen die Lizenzbedingungen dieses Buches klar machen.

¹²<http://www.swaroopch.com/blog/a-free-book-on-vim/#comment-116472>

¹³<http://www.swaroopch.com/notes/Talk:Vim>

¹⁴http://groups.google.com/group/vim_use/msg/dac94f3332f733e4

¹⁵http://vim.wikia.com/wiki/Main_Page#Did_you_know.3F_view_archive

¹⁶<http://creativecommons.org/licenses/by-sa/3.0/>

- Auf jede der obigen Bedingungen kann verzichtet werden, wenn Sie vom Urheber die Erlaubnis dazu bekommen haben.
 - Nichts in dieser Lizenz schmälert oder beeinträchtigt die moralischen Rechte des Urhebers.
2. Die Zuordnung muss kenntlich gemacht werden durch einen Link nach <http://www.swaroop.com/notes/Vim> und klar darauf hinweisen, dass der Originaltext von dieser URL heruntergeladen werden kann.
 3. Jedweder Programmier-Code, alle Skripte in diesem Buch sind lizenziert unter der 3-clause BSD-Lizenz¹⁷, soweit nichts anderes vermerkt ist.
 4. Einige Textpassagen, die in diesem Buch verwendet werden stammen von <http://en.wikipedia.org> und <http://en.wikiquote.org> und stehen unter der GNU Free Documentation License¹⁸.
 5. Freiwillige Beiträge zum Originalbuch müssen unter derselben Lizenz stehen, und das Urheberrecht muss auf den Hauptautor dieses Buches verweisen.

Übersetzungen

Wenn Sie daran interessiert sind Übersetzungen zu lesen oder selbst Übersetzungen in andere Sprachen zu diesem Buch beisteuern möchten, gehen Sie bitte zum Punkt 'Übersetzungen'.

Inhaltsverzeichnis

1	Vorwort	8
1.1	Über 'Vim'	8
1.2	Warum 'Vim'?	9
1.3	Warum wurde dieses Buch geschrieben?	9
1.4	Aktueller Stand des Buches	10
1.5	Offizielle Website	10
1.6	Lizenz und Bestimmungen	10
1.7	Was zum Nachdenken	11
1.8	Vorwort des Übersetzers	11
2	Einleitung	12
2.1	Was ist Vim?	12
2.2	Wozu ist Vim in der Lage?	13

¹⁷<http://www.opensource.org/licenses/bsd-license.php>

¹⁸<http://en.wikipedia.org/wiki/>

3	Installation	16
3.1	Windows	16
3.2	Mac OS X	16
3.3	Linux/BSD	16
3.4	Zusammenfassung	17
4	Erste Schritte	17
4.1	Den Vim starten	17
4.1.1	Grafische Version	17
4.1.2	Terminal Version	17
4.2	Grafisch oder Kommandozeile?	18
4.3	Einführung in Modi	18
4.4	Eine Datei erstellen	20
4.5	Zusammenfassung	22
5	Modi	22
5.1	Einleitung	22
5.2	Arten von Modi	23
5.3	Normalmodus	23
5.4	Wie Sie das Hilfesystem verwenden	23
5.5	Einfügemodus	23
5.6	Visueller Modus	27
5.7	Zusammenfassung	27
6	Tippkompetenz	28
6.1	Einführung	28
6.2	Die Grundreihen-Technik	29
6.3	Der grafische Tastaturspickzettel des Vim	30
6.4	Zusammenfassung	30
7	Navigation	30
7.1	Einführung	30
7.2	Den Cursor bewegen nach Vim-Art	32
7.3	Worte, Sätze, Absätze	34
7.4	Setzen Sie eine Marke	35
7.5	Herumspringen	36
7.6	Textteile	36
7.7	Zusammenfassung	37
8	Hilfe	37
8.1	Einführung	37
8.2	Der :help-Befehl	37
8.3	Wie man das :help topic liest	38
8.4	Das :hilfeallgemeinerregulärererausdruckanzeige-Kommando	38

8.5	Schnelle Hilfe	39
8.6	Internetforum und IRC	39
8.7	Zusammenfassung	39
9	Bearbeiten	40
9.1	Einführung	40
9.2	Lesen und Schreiben von Dateien	40
9.2.1	Zwischenspeicher	40
9.2.2	Auslagern	40
9.2.3	Sichere meine Datei	41
9.2.4	Hinweis	41
9.2.5	Arbeiten in meinem Verzeichnis	41
9.2.6	Herausschneiden, Kopieren und Einfügen	42
9.3	Markieren Sie Ihr Gebiet	44
9.4	Zeitmaschine	45
9.5	Eine starke Suchmaschine aber kein .com	46
9.6	Zusammenfassung	48
10	Vielfältigkeit	48
10.1	Einleitung	48
10.2	Viele Abschnitte verwenden Falten	48
10.3	Mehrere Zwischenspeicher	50
10.4	Mehrere Fenster	51
10.5	Mehrere Reiter	52
10.6	Zusammenfassung	53
11	Persönliche Informationsverwaltung	53
11.1	Einführung	53
11.2	Viki installieren	54
11.3	Seitenbeschreibungssprache	55
11.4	KamelSchrift abschalten	55
11.5	Was erledigt kriegen	56
11.6	Zusammenfassung	57
12	Scripting	57
12.1	Einführung	58
12.2	Makros	58
12.3	Elementarskripting	60
12.4	Aktionen	60
12.5	Entscheidungen	61
12.6	Datenstrukturen	62
12.7	Ein Vim-Skript schreiben	63
12.8	Das Verwenden eigenständiger Programmiersprachen	67
12.9	Zusammenfassung	70

13 Plugins	70
13.1 Einführung	70
13.2 Individualisierung des Vim mit der vimrc	70
13.3 Globales Plugin	74
13.4 Dateityppplugin	74
13.5 Ein Filetype-Plugin benutzen	75
13.6 Ein Dateityp-Plugin schreiben	78
13.7 Syntaxschema	79
13.8 Ein Syntaxschema benutzen	79
13.9 Ein Syntaxschema schreiben	80
13.10 Compiler-Plugins	83
13.11 Hausaufgabe : Schreiben Sie ein globales Plugin	85
13.12 Plugins abschalten	85
13.13 Zusammenfassung	85
14 Ein Editor für Programmierer	85
14.1 Einführung	85
14.2 Einfaches Zeugs	86
14.2.1 Syntaxhervorhebung	86
14.2.2 Intelligentes Einrücken	86
14.2.3 Hüpfen	87
14.2.4 Shell-Befehle	87
14.2.5 Herumspringen	87
14.3 Teile des Codes durchstöbern	88
14.3.1 Dateisystem	88
14.3.2 ctags	88
14.3.3 cscope	89
14.4 Kompilierung (Programmübersetzung)	90
14.5 Schreiben leicht gemacht	90
14.5.1 Omnicompletion	90
14.5.2 Schnipsel benutzen	92
14.5.3 Schnipsel erstellen	93
14.6 Integrierte Entwicklungsumgebung	95
14.6.1 Projekt Plugin	95
14.6.2 Programmcode aus dem Text heraus starten	96
14.6.3 Integration von SCM	97
14.6.4 Mehr	97
14.7 Seine eigenen Plugins schreiben	97
14.8 Zugriff auf Datenbanken	99
14.9 Zusammenfassung	99

15 Mehr	100
15.1 Einführung	100
15.2 Moduszeile	100
15.3 Vim für unterwegs	100
15.4 Plugins aufrüsten	100
15.5 Dr. Chip's Plugins	101
15.6 Aus Vim heraus bloggen	101
15.7 Firefox soll sich wie Vim verhalten	101
15.8 Bram's Gesprächsrunde über die sieben Angewohnheiten	101
15.9 Was beisteuern zu Vim	102
15.10 Die Gemeinschaft	102
15.11 Zusammenfassung	102
16 Ausblick	102
16.1 Einführung	102
16.2 Zusammenfassung	103
17 Resonanz	103
18 Wohltätigkeits-Software	104
19 Schlußschrift	104
19.1 Über dieses Buch	104
19.2 Erschaffung dieses Buches	104
19.3 Inspiration	105
19.4 Über den Autor	105
20 Übersetzungen	105
20.1 Schwedisch	105
20.2 Traditionelles Chinesisch	106
20.3 Russisch	106
20.4 Deutsch	106
21 Glossar	106

1 Vorwort

1.1 Über 'Vim'

*Vim*¹⁹ ist ein Computerprogramm, das benutzt wird um zu schreiben, und es liefert Ihnen eine Reihe von Funktionen, die es Ihnen ermöglichen **besser zu** schreiben.

¹⁹<http://www.vim.org>

1.2 Warum 'Vim'?

Mal ehrlich: ganz selten gelingt Ihnen Ihre beste Arbeit beim ersten Versuch. Wahrscheinlicher ist es, dass Sie Ihren Text immer wieder so lange editieren, bis er 'gut' geworden ist. Wie Louis Brandeis einmal sagte:

There is no great writing, only great rewriting.

→etwa: „Man schreibt nicht große Literatur, sondern man erschreibt sie sich.“

Um diese zahlreichen, schnellen Änderungen zu erledigen, wäre es viel einfacher, wir hätten einen Editor, der diese Fähigkeiten beherrscht, und uns darin unterstützt, und das ist genau der Punkt, wo 'Vim' punktet, und bei weitem besser ist als die meisten Klartext- und rtf- Editoren (rtf = rich text format).

1.3 Warum wurde dieses Buch geschrieben?

Ich verwende den Vim-Editor seit ich den alten VI-Editor aus Unix-Unterrichtsstunden der Universität kennengelernt habe. Vim ist eines der wenigen Computerprogramme, die ich an beinahe 10 Stunden pro Tag einsetze. Ich wusste, daß das Programm so viele Funktionen bereithält, die ich noch nicht kannte, die aber alle das Zeug hatten, für mich eventuell nützlich zu sein, und so fing ich damit an den Vim Stück für Stück zu erforschen.

Um nun mein Verständnis klar herauszuarbeiten, und um anderen dabei zu helfen Vim zu verstehen, habe ich diese Sammlung von Notizen niedergeschrieben, und daraus ist jetzt ein Buch geworden.

Einige der Prinzipien, die ich versucht habe immer im Kopf zu behalten, während des Verfassens dieser Notizen, sind:

1. Eine klare, verständliche Sprache. Die Wichtigkeit dieses Punktes sollte immer wieder betont werden.
2. Schwerpunkt auf Beispielen und Anleitungen.
3. Die zentrale Anlaufstelle für Leser, die den Vim erlernen wollen von den Anfängen bis zu den wirklich anspruchsvollen Dingen.
4. Den Anwender dazu zubringen, Probleme 'vim-typisch' zu lösen von 'Modi' über Zwischenspeicherung bis hin zum Thema 'individuelles Einrichten und Anpassen'. Die meisten Leute erlernen lediglich die grundlegenden VI-Befehle, und versuchen nichts zu lernen, was darüber hinaus geht. Das Erlernen solcher Konzepte trennt die Spreu vom Weizen. Solche Leute gehören dann zum harten Kern, das heißt, sie werden 'Vimmer', was bedeutet, daß sie das meiste aus dem Vim herausholen, und das ist auch die Absicht dieses Buches.
5. Viele Dinge hier sind beschrieben und gespeichert zum nachschlagen für diejenigen, die den Vim als Entwicklungsumgebung nutzen. Probleme kann man immer

verschiedenartig lösen, und anstatt der Anwender sich nun verheddert herauszufinden, welches Plugin er jetzt ausprobieren soll, bietet das Buch dem Leser schon den passenden Hintergrund für seine Arbeit.

6. Gerade genug Information damit Sie es verstehen und anwenden können. Den Vim komplett verstehen zu wollen, macht hier wenig Sinn (Pareto-Prinzip).
7. In diesem Zusammenhang; das Buch sollte nicht versuchen das Nachschlagewerk zu kopieren. Da wo es angebracht ist, sollte es einfach die entsprechenden Teile herausstellen. In dieser Hinsicht entsteht nichts Überflüssiges, der Anwender weiß das hervorragende, eingebettete Nachschlage-Handbuch für sich zu nutzen, was wichtig ist und das Buch steht so auch für sich, aus eigener Stärke.

Zusammenfassend: Das Instrument des Denkens ist - Konzepte. Beispiele. Es *auf den Punkt* bringen.

1.4 Aktueller Stand des Buches

Dieses Buch befindet sich 'in Entwicklung' und kann noch nicht eine 1.0-Version genannt werden.

Hilfreiche Vorschläge sind höchst willkommen. Fügen Sie Ihre Ideen und Vorschläge mit Hilfe des 'Diskussion'-Link an der linken Seitenleiste jeder Seite der offiziellen Website, oder schreiben Sie mir eine Email²⁰.

1.5 Offizielle Website

Die offizielle Website des Buches ist <http://www.swaroopch.com/notes/Vim> . Über den Webauftritt können Sie das ganze Buch im Internet lesen, sich die neueste Version herunterladen, und mir auch Rückmeldungen zuschicken.

1.6 Lizenz und Bestimmungen

1. Dieses Buch steht unter der creative-commons attribution-share alike 3.0 unported-Lizenz²¹

Das bedeutet: Sie können das Buch mit anderen teilen, d.h. kopieren, verteilen und übertragen Sie können es neu zusammenstellen, d.h. Angleichen

Unter den folgenden Bedingungen: Zuordnung. Sie müssen dieses Werk, in der vom Urheber oder Lizenzgeber bezeichneten Weise kenntlich machen (jedoch nicht dergestalt, dass man den Eindruck gewinnt, dass man Sie oder die Tatsache, dass Sie das Buch gebrauchen, unterstützt.)

²⁰<http://www.swaroopch.com/contact/>

²¹<http://creativecommons.org/licenses/by-sa/3.0/>

Share-Alike: Wenn Sie das Buch verändern, transformieren, oder das Werk als Grundlage nutzen, dürfen Sie das neu erschaffene Werk nur unter derselben oder einer ähnlichen Lizenz wie dieser hier weitergeben. Bei jedem Wiederverwenden oder Verteilen, müssen Sie anderen die Lizenzbedingungen dieses Buches klar machen. Auf jede der obigen Bedingungen kann verzichtet werden, wenn Sie vom Urheber die Erlaubnis dazu bekommen haben. Nichts in dieser Lizenz schmälert oder beeinträchtigt die moralischen Rechte des Urhebers.

2. Die Zuordnung muss kenntlich gemacht werden durch einen Link nach <http://www.swaroop.com/notes/Vim> und klar darauf hinweisen, dass der Originaltext von dieser URL heruntergeladen werden kann.
3. Jedweder Programmier-Code, alle Skripte in diesem Buch sind lizenziert unter der 3-clause BSD-Lizenz²², soweit nichts anderes vermerkt ist.
4. Einige Textpassagen, die in diesem Buch verwendet werden stammen von <http://en.wikipedia.org> und <http://en.wikiquote.org> und stehen unter der GNU Free Documentation License²³.
5. Freiwillige Beiträge zum Originalbuch müssen unter derselben Lizenz stehen, und das Urheberrecht muss auf den Hauptautor dieses Buches verweisen.

1.7 Was zum Nachdenken

Bücher werden nicht geschrieben sie werden 'wieder'-geschrieben. Dein eigenes mit eingeschlossen Dies zu akzeptieren gehört zu den schwierigsten Dingen, besonders wenn es auch der siebte Versuch noch nicht geschafft hat.
– Michael Crichton

Perfekt ist etwas nicht, wenn es nichts mehr gibt, das man einer Sache hinzufügen kann, sondern wenn es nichts mehr gibt, das man weglassen könnte.
– Antoine de Saint-Exupery

1.8 Vorwort des Übersetzers

Swaroop C H's Buch Ein Byte Vim ist mein erstes Übersetzungsprojekt. Aus folgenden Gründen habe ich mich dazu bereit erklärt:

- Der Umfang des Buches ist '*überschaubar*' (89 Seiten).
- Das Buch ist leicht verständlich geschrieben.
- Ich bin ein großer Fan von *Vim*.
- Ich will der *Community* mit dieser Arbeit etwas zurückgeben, für all die wunderbare *Freie Software*, die uns allen zur Verfügung steht.

²²<http://www.opensource.org/licenses/bsd-license.php>

²³<http://en.wikipedia.org/wiki/>

2 Einleitung

Allerdings gibt es auch ein paar Probleme, wenn man so ein Buch übersetzt (vor allem beim ersten mal), und die sollen nicht verschwiegen werden. Zum einen wären da die vielen Beispiele aus dem Buch, die man aufgefördert wird nachzuvollziehen, weil man den Vim nun mal am besten dadurch kennenlernt, dass man ihn anwendet. Wann immer es also darum geht, selbst etwas einzutippen, habe ich dies mit **roter Farbe** deutlich gemacht.

Und letztlich ist da das Problem der vielen Anglizismen, die (gerade im Computerbereich) ihren Einzug in die deutsche Sprache gefunden haben. Zuerst wollte ich diese Begriffe schonungslos übersetzen, weil ich mir gesagt habe, wer eine Übersetzung macht, und die Anglizismen unberücksichtigt lässt, der kann es eigentlich auch bleiben lassen. Als Übersetzer habe ich einen Auftrag, den es ernst zu nehmen gilt. Andererseits hat man nun das Problem, dass den Text dann niemand mehr versteht. Übersetzen um jeden Preis, ist rohe Gewalt! Hier möchte ich nur mal zwei Beispiele geben:

- Browser
- Tab

Jeder weiß heutzutage was ein *Browser* ist. Es ist dieses '*Ding*' mit dem man ins Internet geht. Richtig? Und dann kann man auch schon *los-surfen*. Genau! Ein Browser ist ein Programm mit dem man sich Internetseiten (vorzugsweise HTML) anschauen kann. Demnach wäre es ein ..

- Internetseitenanzeigeprogramm

Tabs findet man vor allem auf grafischen Benutzeroberflächen, die an Registerkarten erinnern. Ein Tab ist also ein ..

- Registerkartenelement

Der Kompromiss liegt nun darin, die Anglizismen so zu belassen wie sie sind, und ein eigenes Glossar dafür zu schreiben. Ich habe diese Wörter also mit **blauer Farbe** geschrieben, und wer sich die Übersetzung dafür anschauen will, schlägt genau dort nach.

Der Übersetzer

2 Einleitung

2.1 Was ist Vim?

Vim ist ein Computerprogramm um Texte jeglicher Art, sei es nun Ihr Einkaufszettel, ein Buch, oder Computerprogramme zu schreiben.

Was Vim so besonders macht ist, daß es eines der wenigen Programme ist, die **einfach und vielfältig** sind.

2 Einleitung

Einfach bedeutet, es ist einfach einen Einstieg zu finden. Und einfach bedeutet, es gibt eine minimalistische Schnittstelle, die Ihnen dabei hilft sich auf Ihre Hauptaufgabe zu konzentrieren dem Schreiben. Einfach bedeutet, es ist um ein paar Kernkonzepte herum gebaut, mit denen Sie komplexere Funktionen leicht erlernen können.

Vielfältig bedeutet, Aufgaben schneller, besser und leichter zu erledigen. Vielfältig bedeutet 'nicht-so-einfache-Dinge' zu ermöglichen. Vielfältig bedeutet nicht, dass es kompliziert sein muß. Vielfältig folgt dem Paradigma „**minimale Anstrengung. Höchst-möglicher Effekt.**“

2.2 Wozu ist Vim in der Lage?

Ich hör Sie schon: „Na toll es ist ein Texteditor. Was soll jetzt das ganze Theater?“

Jede Menge 'Theater'.

Schauen wir uns mal ein paar zufällige Beispiele an, um Vim mit dem momentan genutzten Editor Ihrer Wahl zu vergleichen. Sinn dieser Übung für Sie ist es sich die Frage zu beantworten Wie würde ich das mit dem „*Editor, den ich gerade benutze, machen?*“, für jedes Beispiel.

Hinweis Zerbrechen Sie sich nicht den Kopf über Einzelheiten irgendwelcher Vim-Befehle.

Es geht jetzt lediglich darum Sie über die Möglichkeiten aufzuklären, und nicht darum zu erklären, wie diese Befehle funktionieren. Dies tut dann der Rest des Buches.

Aufgabe	Mit Vim
Wie bewegen Sie den Cursor 7 Zeilen nach unten?	Drücken Sie 7j
Wie löschen Sie ein Wort? Ja – ein „Wort“	Drücken Sie dw
Wie durchsuchen Sie die Datei nach genau dem Wort, auf dem der Cursor gerade steht?	Drücken Sie *
Wie machen Sie ein 'Suchen&Ersetzen' daß sich ausschließlich auf die Zeilen 50-100 bezieht?	Führen Sie aus :50,100s/old/new/g
Was ist, wenn Sie zwei verschiedene Teile derselben Datei gleichzeitig sehen möchten?	Führen Sie aus :sp (um die Ansicht zu teilen [to split])
Was, wenn Sie eine Datei öffnen wollen, deren Name im aktuellen Dokument ist, und der Cursor ist auf dem Namen positioniert?	Drücken Sie gf (bedeutet g ehe zu diesem f ile [Datei])

2 Einleitung

- | | |
|---|--|
| Was ist, wenn Sie jetzt mal ein besseres Farbschema auf Ihrem Bildschirm haben wollen? | Führen Sie aus <code>:colorscheme desert</code> →um das 'Wüsten'-Farbschema zu wählen (mein Lieblingsschema) |
| Was, wenn Sie mit dem Tastaturkürzel Ctrl-s den Befehl 'Datei speichern' festlegen möchten? | Führen Sie aus <code>:nmap <c-s> :w<CR></code> Hinweis:<CR> heißt hier: RETURN-Taste drücken! |
| Was, wenn Sie den Satz geöffneter Dateien mit sämtlichen Einstellungen, die Sie vorgenommen haben, speichern möchten, um später mit diesen Einstellungen weiterzuarbeiten? | Führen Sie aus <code>:mksession ~/letzte.sitzung.vim</code> , und öffnen Sie Vim das nächste mal mit <code>vim -S ~/letzte.sitzung.vim</code> |
| Was, wenn Sie verschiedene Stücke Ihres Programmcodes einfärben möchten? | Führen Sie aus <code>:syntax on</code> →Wenn die Sprache nicht richtig erkannt wird, nehmen Sie zum Beispiel <code>:set filetype=Wikipedia</code> |
| Was ist, wenn Sie bestimmte Teile Ihrer Datei ausblenden möchten, um sich besser auf einen Absatz zu einer gegebenen Zeit zu konzentrieren? | Führen Sie aus <code>:set foldmethod=indent</code> →unterstellt, Ihre Datei hat einen sauberen Einzug. Es gibt noch weitere Ausblendmethoden. |
| Was, wenn Sie mehrere Dateien öffnen wollen, mit Zugriff per Tab? | Benutzen Sie <code>:tabedit <Datei></code> um mehrere Dateien mit Zugriff per Tab zu öffnen (genau wie die Tabs Ihres Browsers) und benutzen Sie <code>Ctrl-Seite-Hoch</code> / <code>Ctrl-Seite-Runter</code> um zwischen den Tabs umzuschalten. |
| Einige Wörter benutzen Sie immer wieder in Ihrer Datei, und jetzt wünschen Sie sich eine Methode, um solche Wörter schnell einzufügen, wenn sie das nächste Mal dieses Wort brauchen. | Drücken Sie <code>Ctrl-n</code> um sich die Liste der Ergänzungen für das gegenwärtige Wort anzusehen, basierend auf allen Wörtern, die Sie im geladenen Dokument benutzt haben. Oder Sie machen es so <code>:ab mas</code> →Maslow's benötigte Hierarchie um das Kürzel automatisch zu erweitern, wenn Sie <code>mas <Leertaste></code> tippen. |

2 Einleitung

Sie haben Daten, wobei lediglich die ersten 10 Zeichen jeder Zeile für Sie zu gebrauchen sind, der Rest aber nicht. Wie extrahieren Sie diese Daten?

Drücken Sie **Ctrl-v** [VISUELL BLOCK], markieren Sie den Text und drücken Sie **y** um die markierten Zeilen und Spalten zu kopieren.

Und was ist, wenn Sie von jemanden ein Dokument ausschließlich in Großbuchstaben kriegen, Sie dies verwirrend finden, und es in Kleinbuchstaben umwandeln wollen?

Befehlsfolge in Vim:

```
:for i in range(0, line('$'))  
: call setline(i,tolower(getline(i))  
:endfor
```

Keine Angst. Solche Einzelheiten untersuchen wir in den folgenden Kapiteln. Wenn Sie das kurz & knackig haben wollen, könnten Sie auch das hier eingeben

```
:%s#\(\.\)\#\1\1#g
```

aber dann ist es vielleicht doch einfacher, nach obigem Beispiel zu verfahren. Es gibt sogar noch eine einfachere Methode, um den ganzen Text zu markieren (**1GVG**) und mit dem **u**-Befehlszeichen wandeln Sie alles in Kleinbuchstaben um. Aber das wäre dann wieder zu einfach, und ist bestimmt nicht so 'cool', als mit dem gezeigten Beispiel anzugeben, wo man den Vim in Einzelschritten in Aktion erleben kann.

Puh! Schon überzeugt?

An diesen Beispielen können Sie die vielfältigen Möglichkeiten des Vim im Einsatz sehen. Um eine vergleichbare Stufe der Funktionalität zu erreichen, würden Sie mit jedem anderen Editor schon längst durchdrehen. Und trotzdem das ist das Erstaunliche all diese Vielfalt bleibt, so gut wie nur möglich, verstehbar.

Machen Sie sich bitte klar, dass wir in all den Beispielen noch kein einziges Mal die Maus benutzt haben! Das ist gut. Zählen Sie mal nach, wie viel mal Sie an einem einzigen Tag Ihre Hand zwischen der Tastatur und der Maus hin- und herbewegen, dann wird Ihnen klar, daß Sie das vermeiden möchten, wo immer es geht.

Fühlen Sie sich jetzt durch all die Möglichkeiten hier nicht erschlagen. Das beste am Vim ist nämlich, daß Sie gar nicht sämtliche Funktionen kennen müssen, um ergebnisorientiert damit zu arbeiten. Sie müssen nur ein paar Grundkonzepte kennen. Wenn Sie das gelernt haben, erlernen Sie alle anderen Funktionen leicht, wenn Sie sie brauchen.

3 Installation

3.1 Windows

Wenn Sie Microsoft Windows verwenden, dann werden Ihnen die folgenden Schritte dabei helfen, die neueste Version von Vim 7 auf Ihrem Rechner zu installieren:

1. Besuchen Sie <http://www.vim.org/download.php#pc>
2. Laden Sie sich die selbst-installierende Binärdatei (`gvim72.exe`²⁴ gemäß diesen Textes) herunter
3. Installieren Sie den Vim mit einem Doppel-Klick, wie jedes andere Windowsprogramm auch

3.2 Mac OS X

Wenn Sie Mac OS X benutzen, dann haben Sie die Kommandozeilen-Version vom Vim bereits installiert. Starten Sie den Menübefehl Finder → Anwendungen → Zubehör → Terminal. Im Terminal starten Sie den Befehl `vim` und drücken die `Enter`-Taste. Jetzt sollten Sie den Begrüßungsbildschirm von Vim sehen.

Wenn Sie eine grafische Version des Vim laufen lassen wollen, laden Sie sich die neueste Version des Cocoabasierten MacVim-Projektes²⁵ herunter. Führen Sie einen Doppel-Klick auf die Datei (so was wie `MacVim-7_2-stable-1_2.tbz`) durch, die sich daraufhin entpackt. Damit wird auch ein Verzeichnis `'MacVim-7_2-stable-1_2'` erstellt. Öffnen Sie das Verzeichnis, und kopieren Sie die MacVim-Anwendung in Ihr Anwendungen-Verzeichnis.

Nähere Einzelheiten zu den MacVim Unterschieden, einschließlich wie Sie MacVim von der Kommandozeile starten, finden Sie in der MacVim-Referenz:

1. Klick auf Finder → Anwendungen → MacVim
2. Geben Sie ein `:help macvim` und drücken Sie die `Enter`-Taste.

3.3 Linux/BSD

Wenn Sie ein Linux oder *BSD-System verwenden, dann haben Sie zumindest schon mal eine minimale Konsolenversion des Vim installiert. Öffnen Sie ein Terminalprogramm wie z.B. `konsole` oder `gnome-terminal`, geben Sie `vim` ein, dann sollten Sie den Begrüßungsbildschirm sehen.

Wenn Sie eine Fehlermeldung wie `vim: command not found` erhalten, dann ist Vim nicht installiert. Sie installieren das Programm dann mit Ihren systemspezifischen Werkzeugen, wie z.B. `aptitude` auf Ubuntu/Debian GNU/Linux, `yum` für Fedora Linux, `pkg_add` oder `port` für FreeBSD, etc. Nehmen Sie bitte die zu Ihrem System gehörige Dokumentation zur Hand, und nutzen Sie die entsprechenden Internetforen um in Erfahrung zu bringen, wie Sie neue Programmpakete installieren.

²⁴<http://ftp.vim.org/pub/vim/pc/gvim72.exe>

²⁵<http://code.google.com/p/macvim/>

3.4 Zusammenfassung

Je nach dem, wie das Programm installiert ist, können Sie den **vim**-Befehl in der Kommandozeilenschnittstelle eingeben oder Sie benutzen die Menüs Ihres Betriebssystems um den Vim als grafische Anwendung zu öffnen.

So! Jetzt wo Vim auf Ihrem Computer eingerichtet ist, wollen wir ihn im nächsten Kapitel auch benutzen.

4 Erste Schritte

4.1 Den Vim starten

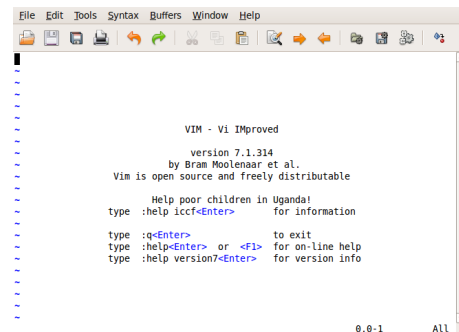
Der erste Schritt ist, na klar, wie man den Vim startet.

4.1.1 Grafische Version

Windows Klicken Sie auf Start → Programme → Vim 7 → gVim

Mac OS X Klicken Sie auf Finder → Anwendungen → MacVim

Linux/BSD Klicken Sie auf Anwendungen → Zubehör → Gvim Text Editor, oder drücken Sie Alt+F2, tippen dann gvim und drücken die 'Enter'-Taste

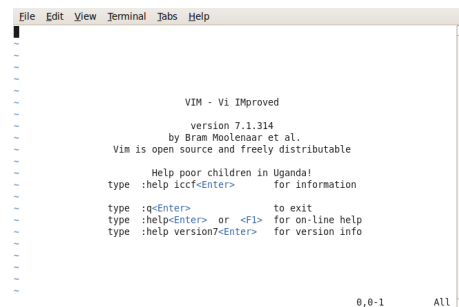


4.1.2 Terminal Version

Windows Klicken Sie auf Start → Öffnen tippen Sie **vim** und drücken Sie die 'Enter'-Taste

Mac OS X Klicken Sie auf Finder → Anwendungen → Zubehör → Terminal, tippen Sie **vim** und drücken Sie die 'Enter'-Taste

Linux/BSD Klicken Sie auf Anwendungen → Zubehör → Terminal, oder drücken Sie Alt+F2, tippen **konsole** bzw. **gnome-terminal** und drücken Sie die 'Enter'-Taste. Geben Sie dann **vim** ein, und drücken Sie 'Enter'.



Von jetzt an gilt: Wenn wir sagen 'Öffnen Sie Vim', benutzen Sie eine der beiden oben genannten Methoden.

Hinweis Wenn Sie den Vim gestartet haben, ist Ihnen vielleicht aufgefallen, daß Sie keinen Text unmittelbar eingeben können. Keine Panik, das erklären wir gleich.

4.2 Grafisch oder Kommandozeile?

Die grafische Version des Vim hat Menüs in der oberen Leiste des Programmfile, sowie verschiedene Optionen, die mit der Maus erreichbar sind. Merken Sie sich jedoch, dass Sie hier absolut freie Wahl haben. Auch mit der Tastatur alleine, haben Sie Zugriff auf sämtliche Funktionen.

Warum ist das wichtig? Weil, wenn jemand erstmal sehr gut tippen kann, und ausschließlich die Tastatur bedient, wird er insgesamt viel schneller und weniger fehleranfällig sein, als wenn er mit der Maus arbeitet. Der Grund hierfür liegt darin: Die nötige Handbewegung um von der Tastatur zur Maus zu wechseln braucht schon eine gewisse Zeit, und auch das Gehirn muß beim Wechsel von der Tastatur zur Maus (und umgekehrt) 'umschalten'. Wenn wir es uns zur Angewohnheit machen, die Tastatur sooft wie möglich zu benutzen, sparen wir uns wertvolle Handbewegungen.

Natürlich ist das subjektiv. Einige bevorzugen die Maus, und einige die Tastatur. Ich möchte Sie dazu ermutigen, sooft wie möglich die Tastatur zu nehmen, um die volle Leistungsfähigkeit des Vim zu erfahren.

4.3 Einführung in Modi

Stellen Sie sich vor, es ist Samstagabend, und die ganzen Fernsehshows langweilen Sie. Stattdessen möchten Sie einen Ihrer alten Lieblingsfilme sehen. Dann schalten Sie also Ihren Fernseher in den 'Videomodus', damit Sie sich eine DVD anstelle der Kabelkanäle anschauen können. Beachten Sie, daß der Fernseher noch immer das Fernsehsignal zeigt, aber Sie haben nun den Zusammenhang geändert, ob Sie sich eine DVD oder einen Fernsehkanal direkt anschauen wollen.

Damit vergleichbar: der Vim hat Modi. Vim hat z.B. einen Modus zur Texteingabe, einen Befehlsmodus, usw. Alle stehen in Beziehung mit dem Editieren von Text als Hauptzweck, aber Sie bestimmen, ob Sie irgendwelches Zeug eingeben wollen oder ob Sie Befehle ausführen möchten, die sich auf den Text beziehen.

Ist das nicht einfach?

Herkömmlicherweise ist das Konzept der Modi bei Anfängern der meistzitierte Grund, weshalb sie den Vim 'verwirrend' finden. Ich vergleiche das mit Fahrradfahren Sie legen sich zwar ein paar mal hin, aber wenn Sie den Dreh erstmal raus haben, wundern Sie sich was das ganze Brimborium darum sollte.

Warum also, hat der Vim Modi? Um die Dinge so einfach wie möglich zu machen, auch wenn ihr Gebrauch zu Anfang „befremdlich“ scheinen mag.

Was will ich damit sagen? Nehmen wir mal dieses Beispiel eines der Schlüsselziele des Vim ist die volle Zugriffsfähigkeit über die Tastatur, ohne daß Sie es je nötig hätten zur Maus zu greifen (Sie können natürlich die Maus nehmen, aber das ist völlig wahlfrei). Wie wollen Sie, vor diesem Hintergrund, unterscheiden zwischen dem Text, den Sie schreiben wollen, und den Befehlen, die Sie ausführen möchten? Vim's Lösung hierfür ist, daß Sie einen Normal-Modus, in dem Sie Befehle ausführen können, und einen Einfüge-Modus haben, um Texte zu schreiben. Zwischen diesen beiden Modi können Sie jederzeit hin- und her wechseln. Wenn Sie **i** drücken schaltet Vim in den Einfügemodus [**i**=insert],

während die `<ESC>`-Taste zurück in den Normalmodus wechselt.

Wie erreichen nun herkömmliche Editoren diese Unterscheidung zwischen Befehls- und Texteingabe? Indem sie grafische Menüs und Tastaturkürzel verwenden. Das Problem ist nur, daß dies als Maßstab nichts taugt. Zunächst mal, wenn Sie hunderte von Befehlen haben, wäre es blanker Wahnsinn, jeden Befehl in einem Menü unterbringen zu wollen. Des weiteren wäre das individuelle Einstellen, wie jeder dieser Befehle zu benutzen ist, noch viel schwieriger.

Nehmen wir mal ein konkretes Beispiel. Angenommen, Sie möchten das Wort „von“, jeweils durch das Wort „nach“ ersetzen, überall wo es in Ihrem Dokument auftaucht. Mit einem herkömmlichen Editor, könnten Sie über Menü einen Befehl wie 'Bearbeiten' → 'Ersetzen' anwenden (oder ein Tastaturkürzel wie Ctrl+R), und dann von eingeben, nach eingeben, und dann auf 'Ersetzen' klicken. Und dann kreuzen Sie noch die Option 'Alle ersetzen' an. Im Vim führen Sie (im Normalmodus) einfach dies aus `:%s/from/to/g`. Das `:s` steht hier für den „Ersetzen“-Befehl. Sehen Sie, wie viel einfacher das ist?

Was, wenn Sie jetzt diese Ersetzung ausschließlich auf die ersten 10 Zeilen des Textes anwenden wollen, und für jede Ersetzung eine 'ja/nein'-Bestätigung haben möchten? Die 'ja/nein'-Bestätigung lässt sich noch leicht mit einem herkömmlichen Texteditor erreichen, indem Sie die 'Alle ersetzen'-Option wieder deaktivieren. Machen Sie sich aber klar, daß Sie nach dieser Option auch erst suchen, und sie dann mit der Maus anklicken müssen (oder Sie nehmen eine lange Folge von Tastaturkürzel in kauf). Aber wie bewerkstelligen Sie das 'Ersetzen', bezogen auf die ersten 10 Zeilen? Im Vim geben Sie einfach ein `:0,10s/from/to/gc`. Die neue cOption, die wir hier verwenden, bedeutet, daß wir eine Bestätigungsmeldung [`c`onfirmation] für jede Ersetzung haben wollen.

Indem man den Schreib-(einfüge), vom Befehls-(normal)modus getrennt hat, ist es mit dem Vim ganz einfach, zwischen diesen beiden Kontexten umzuschalten.

Die ersten Schritte mit Vim scheinen schon ein wenig komisch und ungewohnt zu sein, aber wenn Sie erstmal gesehen haben, welche Ergebnisse man damit erzielen kann, ergibt es auf einmal einen Sinn. Und das beste daran ist, daß Sie mit diesen grundlegenden Konzepten alles wissenswerte für die Benutzung des Vim verstehen lernen.

Da Sie nun den Unterschied zwischen 'Normal'- und 'Einfüge'-Modus begriffen haben, können Sie sich nun die zahlreichen Befehle des Normalmodus anschauen, und unmittelbar anwenden. Vergleichen Sie dies mit dem Erlernen der Befehle eines herkömmlichen Editors, was im allgemeinen heißt, jede Menge Dokumentation zu lesen, nach vielen Menüs zu suchen, vieles mit der 'Versuch & Irrtum'-Methode auszuprobieren oder Sie müssen jemanden gerade heraus um Hilfe bitten.

Persönlich finde ich die Namensgebung der Modi für Anfänger nicht so passend. Ich würde den Einfügemodus lieber „*Schreibmodus*“ und den Normalmodus „*Wieder-Schreibmodus*“ nennen, aber um keine Verwirrung zu stiften, bleiben wir hier beim standardmäßigen Vim-Sprachgebrauch.

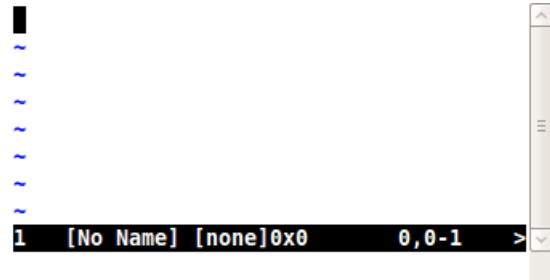
Hinweis Alle Befehle im Normalmodus sollten mit der 'Enter'-Taste abgeschlossen werden, um Vim zu signalisieren, daß wir den ganzen Befehl eingetippt haben. Wenn wir also sagen, 'Führen Sie aus' `:help vim-modes-intro`, dann sollten Sie dies hier

eingeben `:help vim-modes-intro` und anschließend auf die 'Enter'-Taste drücken

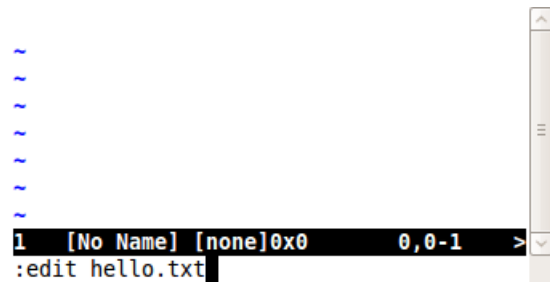
4.4 Eine Datei erstellen

Wir haben gesehen, wie man mit dem Vim Dateien öffnet, und schließt, jetzt wollen wir zwischenzeitlich mal was anderes machen, nämlich *schreiben*.

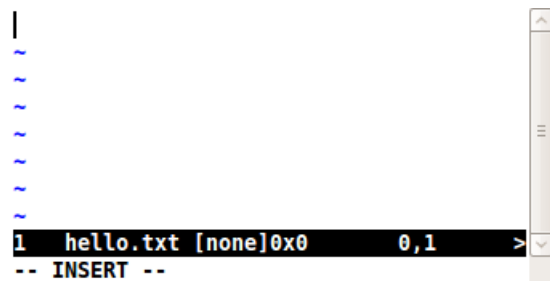
1. Öffnen Sie den Vim



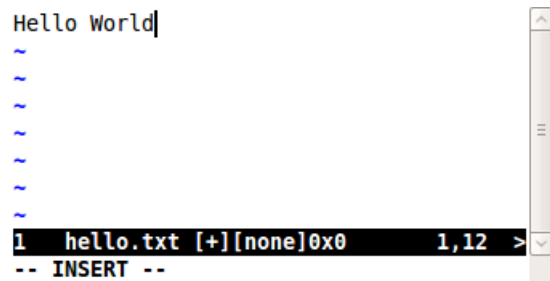
2. Tippen Sie `:edit hallo.txt` und drücken Sie die 'Enter'-Taste



3. `i` drücken



4. Tippen Sie den Text `Hallo Welt`.



5. `<ESC>`-Taste drücken

6. Tippen Sie `:write` und drücken Sie die 'Enter'-Taste

7. Schließen Sie den Vim mit `:q`.

Herzlichen Glückwunsch! Sie haben gerade Ihre erste Datei mit dem Vim erstellt :-).

Sieht das nach vielen Schritten aus? Ja. Zuerst tut es das. Der Grund hierfür ist, daß wir uns zunächst einmal daran gewöhnen müssen, den Vim zu starten, Texte zu schreiben, und das Programm wieder zu beenden. Denken Sie daran dies ist nur ein unwesentlicher Teil der Zeit, verglichen mit dem Zeitaufwand, um den Inhalt unserer Dokumente zu editieren.

Schauen wir uns mal an, was die obigen Befehle tun.

- `:edit hallo.txt` oder einfacher `:e hallo.txt`
 - Das öffnet eine Datei zum editieren. Falls es eine Datei dieses Namens noch nicht gibt, wird sie in dem Moment erstellt, wenn wir sie das erste mal „speichern“.
- `i` drücken

- Damit wechselt Vim in den Einfügemodus
- Tippen Sie den Text **Hallo Welt**.
 - Das ist die eigentliche Texteingabe
- <ESC>-Taste drücken
 - Bringt den Vim zurück in den Normalmodus
- **:write** oder einfacher **:w**
 - Dies weist Vim an, den Text (der zu diesem Zeitpunkt noch im Arbeitsspeicher des Computers ist) als Datei auf die Festplatte zu schreiben. Das bedeutet, was immer wir bisher geschrieben haben, ist nun dauerhaft gespeichert.
- **:quit** oder einfacher **:q** um die Datei in dem Fenster zu schließen, in dem wir gerade editieren. Wenn lediglich ein Fenster geöffnet war, wird damit auch das Programm (also Vim) beendet (Das Konzept der Fenster besprechen wir in einem späteren Kapitel).

Versuchen Sie diesen Ablauf mehrere Male, mit unterschiedlichen Dateinamen, anderem Text, usw., damit Sie mit den grundlegenden Schritten bei der Arbeit mit Vim vertraut werden.

Beachten Sie, wenn Sie im Einfügemodus sind, zeigt Ihnen Vim in der linken unteren Ecke - - EINFÜGEN- - . Wenn Sie sich aber im Normalmodus befinden, sehen Sie gar nichts. Das liegt daran, daß der Normalmodus, der voreingestellte Modus ist, in dem Vim läuft.

Nehmen Sie sich ruhig Zeit, um diese Informationen 'sacken' zu lassen, denn das ist wahrscheinlich die schwierigste Lektion, die Sie bei Vim lernen müssen. Der Rest ist einfach :)

Und keine Angst; von einem Hilfesystem sind Sie nicht allzu weit entfernt. Tatsächlich ist es nur ein **:help-Befehl** entfernt! Probieren Sie mal das hier **:help :edit** und Sie werden sehen, wie sich die dazugehörige Beschreibung öffnet. Na los – versuchen Sie's!

4.5 Zusammenfassung

Wir haben nun die grundlegenden Konzepte und die grundlegende Benutzung des Vim besprochen. Sehen Sie sich auch **:help notation** und **:help keycodes** an.

Verinnerlichen Sie diese Konzepte gut. Wenn Sie damit angefangen haben 'vim-typisch' zu denken, dann verstehen Sie die restlichen Funktionen des Vim leicht.

5 Modi

5.1 Einleitung

Die erste Bekanntschaft mit Modi, hatten wir im vorhergehenden Kapitel. Jetzt wollen wir dieses Konzept im Hinblick auf die verfügbaren Modi noch weiter untersuchen, und

sehen, was wir im jeweiligen Modus machen können.

5.2 Arten von Modi

Vim kennt drei Grundmodi Normal, Einfüge, und Visuell.

- Im Normalmodus können Sie Befehle ausführen. In diesem Modus startet Vim standardmäßig.
- Im Einfügemodus fügen Sie Text ein, d.h. damit schreiben Sie den Text.
- Mit dem Visuellen-Modus markieren Sie einen von Ihnen bestimmten Text, so daß Sie einen Befehl, eine Operation, auf genau diesen Text anwenden können.

5.3 Normalmodus

Voreingestellt, befinden Sie sich im Normalmodus. Lassen Sie uns mal sehen, was man in diesem Modus alles machen kann.

Tippen Sie `:echo "hallo welt"` und drücken Sie die 'Enter'-Taste. Nun sollten Ihnen die berühmten Worte *hallo welt* zurück schallen. Sie haben jetzt eben einen Vim-Befehl mit Namen `:echo` ausgeführt, ihm noch etwas Text mit auf den Weg gegeben, der Ihnen dann sofort ausgegeben wurde.

Tippen Sie `/hallo` und drücken Sie die 'Enter'-Taste. Vim wird nach diesem Ausdruck suchen, und zu diesem Ausdruck springen, sobald er das erste mal erscheint.

Das waren gerade zwei einfache Beispiele der Art, wie sie im Normalmodus zur Verfügung stehen. In späteren Kapiteln werden wir noch viel mehr von solchen Befehlen zu sehen bekommen.

5.4 Wie Sie das Hilfesystem verwenden

Fast genauso wichtig, wie den Normalmodus zu kennen, ist es zu wissen, wie man den `:help`-Befehl anwendet. Das ist der Ort, an dem Sie mehr über die Befehle erfahren, die im Vim zur Verfügung stehen.

Halten Sie fest, daß es nicht nötig ist jeden Vim-Befehl zu kennen. Es ist einfach besser zu wissen, wo man sie findet, wenn man sie braucht. Schauen Sie sich mal das hier an: `:help usr_toc` zeigt uns das Inhaltsverzeichnis des Referenzhandbuches (`toc`=table of contents). Oder sehen Sie mal hier nach `:help index`, um genau nach dem Thema zu suchen, das Sie gerade interessiert. Geben Sie dann z.B. ein `/insert mode`, um die zutreffenden Informationen, den Einfügemodus betreffend zu sehen.

Falls Sie sich diese beiden Hilfethemen anfangs nicht merken können, drücken Sie nur F1 oder geben Sie einfach `:help` ein.

5.5 Einfügemodus

Wenn Vim im Normalmodus hochfährt, wissen wir ja jetzt, daß man mit `i` in den Einfügemodus gelangt. Aber es gibt auch noch andere Möglichkeiten um vom Normalmodus in den Einfügemodus zu wechseln:

- Tippen Sie **:e** dapping.txt
- **i** drücken
- Geben Sie den folgenden Absatz ein (mit allen Rechtschreibfehlern, wir korrigieren das später):
`means being determined about being determined and being passionate
about being passionate`
- Drücken Sie die **<ESC>**-Taste, um in den Normalmodus zurückzugelangen.
- Geben Sie ein **:w**

Ach du Schreck! Sieht ganz so aus, als ob wir am Zeilenanfang ein Wort weggelassen haben, und unser **Cursor** steht nun aber am Zeilenende. Was machen wir denn nun?

Welche Methode wäre wohl die effektivste um an den Zeilenanfang zu kommen, und das fehlende Wort einzufügen? Nehmen wir jetzt etwa die Maus, um den **Cursor** an den Zeilenanfang zu setzen? Oder sollten wir vielleicht mit den Pfeiltasten eine lange Reise an den Zeilenanfang unternehmen? Drücken wir die **Pos1**-Taste, und dann **i**, um wieder in den Einfügemodus zu gelangen?

Es stellt sich heraus, als ob die beste Lösung darin besteht **I** zu drücken (den Großbuchstaben):

- Drücken Sie **I**
- Schreiben Sie **Dappin**
- Und mit **<ESC>** schalten wir wieder in den Normalmodus.

Beachten Sie, daß wir einen anderen Operator verwendet haben, um in den Einfügemodus zu gelangen, dessen besonderes Merkmal ist, daß er den **Cursor** an den Zeilenanfang setzt und dann in den Einfügemodus schaltet.

Ganz wichtig ist auch, daß Sie in den Normalmodus zurückgehen, sobald Sie Ihre Texteingabe beendet haben. Sie profitieren davon, wenn Sie sich dies zur Angewohnheit werden lassen, weil sich der Großteil Ihrer Arbeit (also nach der Texteingabe) im Normalmodus abspielt – da findet nämlich dann das '*Umschreiben*', Editieren, und Verbessern statt.

Jetzt sehen wir uns einmal eine weitere Variante des **i**-Befehls an. Die **i**-Taste setzt den **Cursor** vor die aktuelle Stelle und schaltet dann in den Einfügemodus. Um den **Cursor** hinter die aktuelle Position zu setzen, drücken Sie **a** ('a'fter).

- Drücken Sie **a**
- Tippen Sie **g** (um das Wort Dappin richtig zu ergänzen)
- **<ESC>** drücken, um in den Normalmodus zu gelangen

Analog zur Beziehung zwischen den Operatoren **i** und **I**, ist die Beziehung zwischen den Kommandos **a** und **A**. Wenn Sie Text am Zeilenende anfügen möchten, drücken Sie den **A**-Operator.

- Drücken Sie **A**
- Geben Sie ein **.** (einen Punkt, um den Satz korrekt abzuschließen)
- **<ESC>**- Taste drücken um in den Normalmodus zu gelangen

Um die vier Befehlsoperatoren, die wir bisher kennengelernt haben, zusammenzufassen:

Befehl	Bewirkt
i	fügt Text <i>vor</i> dem Cursor ein
I	fügt Text am Zeilenanfang ein
a	hängt Text <i>nach</i> dem Cursor an
A	hängt Text am Zeilenende an

Kleine Eselsbrücke – Die Großbuchstabenbefehle erledigen auch umfangreichere (größere) Aufgaben, als die Kleinbuchstabenoperatoren.

Da wir nun im Schnell navigieren auf der aktuellen Zeile geübt sind, sehen wir uns mal an, wie man sich zu neuen Zeilen hinbewegt. Wenn Sie eine neue Zeile eröffnen [**o**'pen] wollen, um darauf zu schreiben, drücken Sie den **o**-Operator.

- Drücken Sie **o**
- Tippen Sie **Ich bin ein Rapper.**
- **<ESC>** drücken, um in den Normalmodus zu gelangen

Hmmm, wär schon scharf, wenn wir den neuen Satz, den wir geschrieben haben, als eigenen Absatz haben könnten.

- Drücken Sie **O** (Großbuchstabe 'O')
- Drücken Sie **<ESC>** um in den Normalmodus zurückzugelangen.

Fassen wir die beiden eben gelernten Befehlsoperatoren, nochmal zusammen:

Befehl	Wirkung
o	eine neue Zeile unterhalb eröffnen
O	eine neue Zeile oberhalb eröffnen

Beachten Sie, daß die Großbuchstaben-, Kleinbuchstaben-Befehle, hinsichtlich der Richtung, in der sie die Zeilen eröffnen, das jeweilige Gegenteil sind.

War da gerade was falsch in dem Text, den wir eben eingegeben haben? Aah, es sollte 'dapper' und nicht 'rapper' heißen! Wir müssen einen einzelnen Buchstaben ändern. Und was ist wohl die effektivste Methode hierfür?

Wir *könnten* **i** drücken, um in den Einfügemodus zu schalten, oder **** um das r zu löschen, **d** tippen und dann wieder **<ESC>** drücken, um in den Einfügemodus zurückzugelangen. Aber das sind vier Einzelschritte, für so eine banale Korrektur! Gibts dafür was besseres? Hier können Sie den **s**-Operator nehmen [**s**'ubstitute].

- Bewegen Sie den Cursor zum Buchstaben r (oder drücken Sie einfach **b** um an den Wortanfang zu gelangen [b='b'ack])
- **s** drücken
- **d** tippen
- und mit **<ESC>** wieder zurück in den Einfügemodus

Na schön, das hat uns jetzt nicht wirklich viel gebracht. Aber stellen Sie sich bitte vor, Sie müssten eine solche Abfolge den ganzen Tag über, immer und immer wieder machen. So eine profane Arbeit schnellstmöglich zu erledigen, nützt uns, weil wir unsere Kraft auf schöpferische und interessantere Dinge lenken können. Wie Linus Torvalds sagt, „*Es geht nicht nur darum, Dinge schneller zu erledigen, aber weil es so schnell ist, wird es Ihre Arbeitsweise dramatisch ändern.*“

Also nochmal, es gibt eine Großvariante des **s**-Operators, mit **S** tauschen Sie die ganze Zeile aus, anstatt den aktuellen Buchstaben.

- Drücken Sie **S**
- Tippen Sie **be a sinner**
- Drücken Sie **<ESC>** um in den Normalmodus zu gelangen.

Befehl Wirkung

s	ersetzt den aktuellen Buchstaben
S	ersetzt die aktuelle Zeile

Gehen wir nochmal zu unserer letzten Aktion zurück... Können wir das nicht effektiver machen, wo wir doch lediglich einen einzelnen Buchstaben austauschen wollen? Ja – können wir. Wir nehmen dazu den **r**-Operator [**r**'eplace].

- Bewegen Sie den **Cursor** auf den ersten Buchstaben des Wortes sinner.
- Drücken Sie **r**
- Tippen Sie **d**

Gemerkt? Wir sind jetzt schon im Normalmodus, und das, ohne **<ESC>** zu drücken.

Auch für **r** gibt es eine Großversion, die nennt sich **R**, und ersetzt fortlaufende Buchstaben.

- Bewegen Sie den **Cursor** auf das 'i' in sinner
- Drücken Sie **R**
- Tippen Sie **app** (das Wort wird nun zu 'dapper')
- Drücken Sie **<ESC>** um in den Normalmodus zu gelangen.

Befehl	Wirkung
--------	---------

r	den aktuellen Buchstaben ersetzen
R	fortlaufende Buchstaben ersetzen

Der Text sollte nun so aussehen:

„Dapping means being determined about being determined and being passionate about being passionate. Be a dapper.“

Puh! Damit haben wir vieles abgedeckt in diesem Kapitel, aber ich garantiere Ihnen, das ist der einzige, schwierigste Teil.

Wenn Sie das erstmal verstanden haben, verstehen Sie Herz & Seele der Arbeitsweise des Vim, und die ganze andere Pracht und Herrlichkeit des Vim, ist dann nur noch das Sahnehäuptchen oben drauf. Um es noch einmal zu sagen; das Verständnis über Modi, und wie das Hin- und Herschalten zwischen Modi funktioniert, ist der Schlüssel ein Vimmer zu werden. Wenn Sie also die obigen Beispiele noch nicht ganz verinnerlicht haben, lesen Sie sie ruhig nochmal, und nehmen Sie sich alle dafür notwendige Zeit.

Wenn Sie mehr lesen möchten über die Einzelheiten dieser Befehle, schauen Sie sich `:help inserting` und `:help replacing` an.

5.6 Visueller Modus

Stellen Sie sich vor, Sie wollen ein paar Worte markieren, um sie durch einen völlig neuen Text zu ersetzen, den Sie schreiben wollen. Was tun Sie?

Eine Methode wäre es, mit der Maus auf den Anfang des in Betracht kommenden Textes zu klicken, die linke Maustaste gedrückt zu halten, die Maus dann bis ans Ende des zu wählenden Textes zu ziehen, und die linke Maustaste wieder los zulassen. Das sieht nach schrecklich viel Ablenkung aus.

Wir könnten die **Entf**- oder die **Zurück**-Taste nehmen, um alle Buchstaben zu löschen, aber das ist *noch* umständlicher.

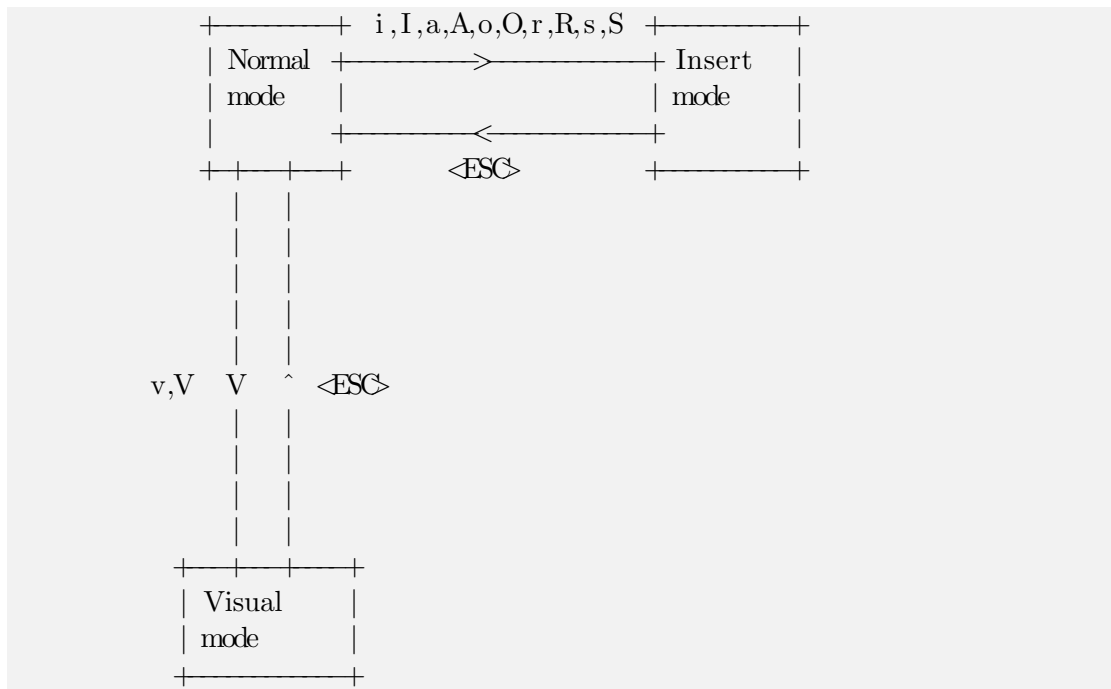
Die beste Methode wäre es wohl, den **Cursor** auf den Textanfang zu setzen, **v** zu drücken, um den Visuellen Modus aufzurufen, die Pfeiltasten oder jeden anderen Textbewegungsbefehl (sagen wir mal, **5e** drücken, um ans Ende des 5. Wortes zu kommen, gezählt von der aktuellen Position des **Cursors**) zu benutzen, um an das Ende des relevanten Textes zu gelangen, und dann **c** zu drücken um den Text zu ändern [`'c'hange`]. Das ist schon wirtschaftlicher.

Bei dieser speziellen Operation (dem **c**-Befehl), landen Sie nach Abschluss im Einfügemodus, also drücken Sie **<ESC>**, um in den Normalmodus zurückzugelangen.

Der **v**-Befehl arbeitet Buchstaben bezogen. Wollen Sie Zeilen bezogen arbeiten, nehmen Sie den Großbuchstaben **V**.

5.7 Zusammenfassung

Hier ist eine Zeichnung, die die Beziehung zwischen den verschiedenen Modi deutlich macht:



(Das Bild wurde mit Vim und Dr.Chip's DrawIt²⁶ Plugin erstellt.)

Sehen Sie sich `:help vim-modes-intro` und `:help mode-switching` für die Einzelheiten der verschiedenen Modi an, bzw., wie man zwischen ihnen hin- und herschaltet.

Wenn Sie jetzt immer noch nicht überzeugt davon sind, daß das Moduskonzept von Vim für die Vielfältigkeit und Einfachheit von zentraler Bedeutung sind, lesen Sie bitte die Artikel „Why Vi“²⁷ und weshalb das *Vi-Eingabemodell*²⁸ die bessere Art zu editieren ist.

6 Tippkompetenz

6.1 Einführung

Gerade eben haben wir gelernt, zwischen den drei grundlegenden Modi in Vim hin- und herzuschalten. Ist das nicht schwierig, sich all die Tastenkombinationen zu merken? Wie behält man sich, welcher Schlüssel für welche Operation ist? Nun, das ist die „Schlüssel-frage“. Und die Antwort darauf ist; Sie sollten sich eben nicht erinnern, sondern Ihre Finger sollten automatisch wissen, was zu tun ist! Es muss (buchstäblich) immer *zur Hand* sein.

Wie stellen wir das also an? Indem wir es uns zur Angewohnheit machen. Das *Ge-hen* wird uns Menschen auch nicht in die Wiege gelegt, aber nach ein paar Versuchen,

²⁶http://vim.sourceforge.net/scripts/script.php?script_id=40

²⁷<http://www.viemu.com/a-why-vi-vim.html>

²⁸<http://blog.ngedit.com/2005/06/03/the-vi-input-model/>

und durch Gewohnheit, lernen wir gehen. Genauso ist es mit Vim, nur daß es weniger anstrengend ist.

Vim wurde konzipiert für Leute, die mit der Tastatur vertraut sind. Und warum? Weil wir die meiste Zeit mit dem Editieren von Aufgaben verbringen, was den bedingungslosen und ausgiebigsten Gebrauch der Tastatur mit sich bringt, und je schneller wir tippen können, umso schneller sind wir mit unserer Arbeit fertig.

Fangen wir mal mit einer grundlegenden Technik an, damit Sie sich mit der Tastatur etwas wohler fühlen.

6.2 Die Grundreihen-Technik

Setzen Sie Ihre Finger auf die Grundreihe²⁹ der Tastatur, und zwar so, daß die Finger Ihrer linken Hand auf den Tasten **ASDF**, und die Finger Ihrer rechten Hand auf **JKLÖ** [→Ö=deutsche Tastatur] liegen, wie im Bild zu sehen (Künstler unbekannt)³⁰. Das Vertraut sein Ihrer Hände in dieser Position, ist einer der wichtigsten Schritte, um die Tastatur wirklich nutzbringend einsetzen zu können.



Die dahinter stehende Idee ist, daß Sie in der Lage sein sollten, jede Taste mit dem nächstgelegenen Finger anzuschlagen, und daß Ihr Finger danach automatisch in die Ursprungslage zurückfindet. Das scheint zuerst schwierig, aber versuchen Sie das ein paar mal, und dann merken Sie, daß Sie so viel schneller tippen können.

Beachten Sie, daß die meisten Tastaturen auf den Tasten **F** und **J** Markierungen haben, die Ihnen dabei helfen, diese Grundposition zu finden.

Versuchen Sie jetzt mal die Alphabete **A-Z** mit der Grundreihen-Technik zu tippen. Übrigens; es steht auch ein freies Tipptraining³¹ im Internet zur Verfügung, welches diese grundlegenden Schreibtechniken erklärt. Ich möchte Sie ermutigen, das nur mal für zehn Minuten auszuprobieren und kennenzulernen.

²⁹http://en.wikipedia.org/wiki/Home_row

³⁰<http://www.bigpants.ca/juggling/images/Controls.Keyboard.HomeRow.gif>

³¹<http://www.typeonline.co.uk/lesson1.html>

6.3 Der grafische Tastaturspickzettel des Vim

Wenn Sie wissen möchten, wie man mit dem Vim jede Taste einer nützlichen Funktion zuordnen lässt, werfen Sie einen Blick auf diesen grafischen Tastaturspickzettel von jng³².

Obwohl Sie da jede Menge Befehle aufgelistet sehen, genügt es erst einmal die grundlegenden Tasten 'hjk', die den Pfeiltasten (links, rechts, oben, unten) entsprechen, zu lernen. Darüber lernen Sie dann noch mehr im nächsten Kapitel.

version 1.1
April 1st, 06

vi / vim graphical cheat sheet

Esc
normal mode

~ toggle case	! external filter	@, play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat :s	% next ident	(begin sentence) end sentence	"soft" bol down	+ next line
. goto mark	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto-format

Q ex mode	W next WORD	E end WORD	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	} end parag.
q record macro	w next word	e end word	r replace char	t 'till	y yank	u undo	i insert mode	o open below	p paste after	[misc] misc

A append at eol	S subst line	D delete to eol	F "back" find ch	G eof/ goto ln	H screen top	J join lines	K help	L screen bottom	. ex cmd line	" reg. spec	bol/ goto col
a append	s subst char	d delete	f find char	g extra cmds	h	j	k	l	. repeat t/T/f/F	' goto mk. bol	\ not used!

Z quit	X back-space	C change to eol	V visual lines	B prev WORD	N prev (find)	M screen mid'l	< un-indent	> indent	? find (rev.)
z extra cmds	x delete char	c change	v visual mode	b prev word	n next (find)	m set mark	reverse y t/T/f/F	. repeat cmd	/ find

motion moves the cursor, or defines the range for an operator

command direct action command, if red, it enters insert mode

operator requires a motion afterwards, operates between cursor & destination

extra special functions, requires extra input

Q commands with a dot need a char argument afterwards

bol = beginning of line, eol = end of line, mk = mark, yank = copy

words: quux(foo, bar, baz)

WORDS: quux(foo, bar, baz)

Main command line commands ('ex'):

:w (save), :q (quit), :q! (quit w/o saving)

:e f (open file f), :%s/x/y/g (replace 'x' by 'y' filewide), :h (help in vim), :new (new file in vim),

Other important commands:

CTRL-R: redo (vim), CTRL-F/-B: page up/down, CTRL-E/-Y: scroll line up/down, CTRL-V: block-visual mode (vim only)

Visual mode:

Move around and type operator to act on selected region (vim only)

Notes:

(1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,") (e.g.: "ay\$ to copy rest of line to reg 'a')

(2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)

(3) duplicate operator to act on current line (dd = delete line, >> = indent line)

(4) ZZ to save & quit, ZQ to quit w/o saving

(5) zt: scroll cursor to top, zb: bottom, zz: center

(6) gg: top of file (vim only), gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

6.4 Zusammenfassung

Seien Sie sich darüber im Klaren, daß die Effektivität den Vim zu nutzen proportional ist, zu Ihren Fähigkeiten mit der Tastatur.

7 Navigation

7.1 Einführung

Wenn Sie erstmal den Anfangstext geschrieben haben, verlangt das Editieren und Umschreiben viele Bewegungen zwischen den verschiedenen Teilen des Dokuments. Sie

³²http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html

schreiben zum Beispiel an einer Geschichte, und plötzlich fällt Ihnen ein neuer Handlungsstrang ein, aber um den zu entwickeln, müssen Sie zu dem Teilstück zurückgehen, wo der Protagonist in die neue Stadt kommt (oder so was in der Art)... Wie bewegen Sie sich in Ihrem Text so schnell, daß Sie Ihren Gedankengang nicht gleich wieder verlieren? Hier sind ein paar Beispiele, die die Schnelligkeit des Vim demonstrieren sollen.

- Den **Cursor** zum nächsten Wort bewegen? Drücken Sie **w**.
- Zum nächsten Absatz gehen? Drücken Sie **}**.
- Zum dritten Erscheinen des Buchstabens 'h' gehen? Drücken Sie **3fh**.
- 35 Zeilen nach unten gehen? Drücken Sie **35j**.
- Wollen Sie nach einer der obigen Bewegungen, wieder zur Ausgangsposition? Drücken Sie **Ctrl-o**.
- Lust, zu lernen, wie das alles funktioniert? Tauchen wir ab.

Zunächst öffnen (erstellen) Sie eine Datei mit Namen `chandrayaan.txt` und geben den folgenden Text ein (aus der Wikipedia³³):

Chandrayaan-1 ist Indiens erste Mondmission, initiiert von der Nationalen Indischen Raumfahrtagentur, der ISRO (Indian Space Research Organisation). Die unbemannte Monderforschungsmission umfasst einen Mondumkreiser und einen Stoßkörper. Der Raumflugkörper wurde durch eine modifizierte Version der PSLV XL am 22. Oktober 2008 vom Satish Dhawan Raumfahrtzentrum, Sriharikota, Andrah Pradesh um 6:23 Uhr IST (00:52 UTC) hochgeschossen. Der Raumflugkörper wurde erfolgreich in die Mondumlaufbahn am 8. November 2008 gebracht. Die Mondmesssonde wurde erfolgreich am Südpol des Mondes um 20:31 Uhr, am 14. November 2008 abgesetzt.

Der ferngesteuerte Satellit hatte zum Startzeitpunkt eine Masse von 1,380 kg (3,042 lb) und 675 kg (1,488 lb) in der Mondumlaufbahn, und trägt hochauflösende Gerätschaften für den sichtbaren Bereich (Infrarotbereich), und für harte und weiche Röntgenstrahlungsfrequenzen. Über einen Zeitabschnitt von zwei Jahren, ist beabsichtigt die Mondoberfläche zu vermessen, um eine vollständige Karte seiner chemischen Eigenschaften und eine dreidimensionale Topografie anzufertigen. Die Polregionen sind dabei von besonderem Interesse, da sie möglicherweise Eis enthalten. Die Mondmission befördert fünf ISRO-Ladungen und sechs Ladungen aus anderen internationalen Raumfahrtagenturen einschließlich der NASA, ESA, und der bulgarischen Raumfahrtagentur, die kostenlos transportiert wurden.

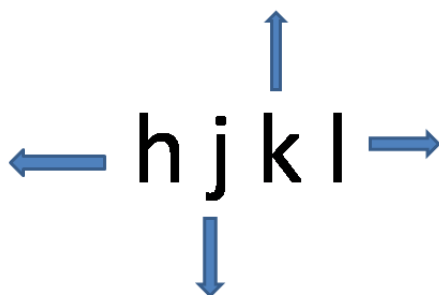
³³<http://en.wikipedia.org/wiki/Chandrayaan-1>

7.2 Den **Cursor** bewegen nach Vim-Art

Die grundlegendsten Tasten, die Sie benutzen sollten sind **'h j k l'**. Diese vier Tasten entsprechen jeweils den Pfeiltasten links, runter, hoch und rechts. Beachten Sie, daß sich diese Tasten bereits unter den Fingern Ihrer rechten Hand befinden, wenn Sie die Hand schon in der Grundhaltung haben.

Aber warum denn die Pfeiltasten nicht direkt benutzen? Das Problem hierbei ist, daß sie in einem eigenen Bereich der Tastatur untergebracht sind, und das würde wieder eine Handbewegung erforderlich machen. Dann könnten Sie auch gleich wieder die Maus nehmen.

Halten Sie fest, die Finger Ihrer rechten Hand sollten sich immer auf den Tasten **j k l ö** befinden (und der Daumen auf der Leertaste). Jetzt wollen wir die vier Tasten mal kennenlernen.



h,j,k,l anstatt der Pfeiltasten benutzen

- h** Sie müssen Ihren Zeigefinger nach links ausstrecken, (der ist auf 'j') um 'h' zu drücken. Dies ist die am weitesten links liegende Taste, gibt also an, daß es nach links geht.
- j** Das fallende 'j', weist die Richtung nach unten.
- k** Das nach oben weisende 'k' heißt, es geht nach oben.
- l** Der am weitesten rechts stehende Buchstabe 'l' signalisiert, es geht nach rechts.

Beachten Sie, daß wir die Operation wiederholen können, indem wir eine Zahl davor setzen. Zum Beispiel, **2j** wird den Befehl 'j' zweimal ausführen.

Öffnen Sie nun die **chandrayaan.txt**-Datei und probieren Sie diese Tasten aus:

- Setzen Sie Ihre Schreibmarke auf das erste 'C' des Dokuments.
- Geben Sie **2j** ein und die aktuelle lange Zeile und die Leerzeile sollte übersprungen werden, und Sie sollten nun auf der zweiten Zeile, d.h. den zweiten Absatz gelangt sein.
- Geben Sie **2k** ein, um an Ihren Ausgangspunkt zurückzukehren. Sie können aber auch **ctrl-o** drücken, um zurückzuspringen.

- Geben Sie **5l** ein, um 5 Zeichen nach rechts zu gehen.
- Geben Sie **5h** ein, um 5 Zeichen nach links zu gehen, oder Sie drücken ctrl-o.

Machen Sie es sich zur Angewohnheit die Tasten '**h j k l**' anstatt der Pfeiltasten zu benutzen. Nach ein paar Versuchen werden Sie gemerkt haben, wieviel schneller Sie mit diesen Tasten sein können.

Es gibt, in dieser Hinsicht, noch mehr einfache Tasten, die die folgenden Spezialbewegungen ersetzen. All dies dient dazu, Handbewegungen zu reduzieren. In diesem besonderen Fall neigen die Leute dazu nach diesen Spezialtasten auf die Jagd zu gehen, aber das können wir alles weglassen.

Herkömmlich

Vim

'Pos1'-Taste springt zum Zeilenanfang

^-Taste (denken Sie an 'verankert am Start')

'Ende'-Taste springt zum Zeilenende

\$-Taste (denken Sie an 'Der Dollar stoppt hier')

'Bild hoch'-Taste, eine Seite nach oben

ctrl-b das bedeutet, eine Seite zurück (backward)

'Bild runter'-Taste, eine Seite nach unten

ctrl-f das bedeutet, eine Seite weiter (forward)

Wenn Sie die Zeilennummer ganz genau kennen, zu der Sie springen möchten, sagen wir mal Zeile 50, tippen Sie **50G** und Vim springt direkt zur Zeile 50. Wenn keine Nummer angegeben wurde, kommen Sie mit **G** zur letzten Zeile Ihres Dokumentes. Wie kommen Sie an den Anfang Ihrer Datei? Einfach **1G** tippen. Sehen Sie, was man mit *einer einzigen* Taste alles erreichen kann?

- Setzen Sie den **Cursor** auf die erste Zeile, indem Sie **1G** eingeben.
- 20 Buchstaben nach rechts, indem Sie **20l** eingeben.
- Zurückgehen zum ersten Buchstaben mit **^**.
- Mit **\$** zum letzten Buchstaben springen.
- Geben Sie **G** ein, um zur letzten Zeile zu gelangen.

Was ist, wenn Sie sich in die Mitte des Textes bewegen wollen, der gerade Ihren Bildschirm füllt?

- Geben Sie **H** ein, um so '**h**'och wie möglich zu springen (erste Zeile des Bildschirms)
- Geben Sie **M** ein, um in die '**M**'itte zu springen
- Geben Sie **L** ein, um so tief ('**l**ow) wie möglich zu springen (letzte Bildschirmzeile)

Es kann Ihnen nicht entgangen sein, welchen Nachdruck wir auf das Blindschreiben gelegt haben, und daß Sie mit Ihren Händen die Grundhaltung noch nie verlassen mussten. Das ist eine gute Sache.

7.3 Worte, Sätze, Absätze

Wir haben gesehen, wie man navigiert, bezogen auf Buchstaben und Zeilen. Aber wir neigen dazu unseren Text aus Wörtern bestehend zu betrachten, und wie wir diese Wörter zusammensetzen - Sätze, Absätze, Abschnitte, usw. Warum also, sollte man solche Textteile nicht als „Textobjekte“ sehen?

Nehmen wir mal die ersten paar Worte unseres Beispieltexes:

```
Die Polregionen sind dabei von besonderem Interesse, da sie möglicherweise Eis
enthalten.
```

Zuerst setzen Sie den **Cursor** auf den ersten Buchstaben durch Drücken von **^**.

```
[D]ie Polregionen sind dabei von besonderem Interesse, da sie möglicherweise
Eis enthalten.
```

Wir benutzen jetzt eckige Klammern, um die aktuelle **Cursor**position hervorzuheben.

Wollen Sie zum nächsten 'W'ort? Drücken Sie **w**. Der **Cursor** sollte nun auf dem 'P' von 'P'olregionen sein.

```
Die [P]olregionen sind dabei von besonderem Interesse, da sie möglicherweise
Eis enthalten.
```

Wie wärs, wenn Sie jetzt mal 2 Wörter weiterspringen? Fügen Sie einfach das Zählpräfix für 'w' hinzu, also: **2w**.

```
Die Polregionen sind [d]abei von besonderem Interesse, da sie möglicherweise
Eis enthalten.
```

Ähnlich verhält es sich, wenn Sie sich ans 'E'nde des nächsten Wortes bewegen möchten. Drücken Sie **e**.

```
Die Polregionen sind dabe[i] von besonderem Interesse, da sie möglicherweise
Eis enthalten.
```

Um ein Wort zurückzuspringen, drücken Sie **b** [**b**=backward]. Mit einem Zähler davor, **2b**, springen Sie 2 Worte zurück.

```
Die Polregionen [s]ind dabei von besonderem Interesse, da sie möglicherweise
Eis enthalten.
```

Für Details, sehen Sie sich auch **:help word-motions** an.

Wir haben gesehen, wie man buchstabenweise und wortweise navigieren kann, jetzt gehen wir weiter zu Sätzen.

[C]handrayaan-1 ist Indiens erste Mondmission, initiiert von der Nationalen Indischen Raumfahrtagentur, der ISRO (Indian Space Research Organisation). Die unbemannte Monderforschungsmission umfasst einen Mondumkreiser und einen Stoßkörper. Der Raumflugkörper wurde durch eine modifizierte Version der PSLV XL am 22. Oktober 2008 vom Satish Dhawan Raumfahrtzentrum, Sriharikota, Andrah Pradesh um 6:23 Uhr IST (00:52 UTC) hochgeschossen. Der Raumflugkörper wurde erfolgreich in die Mondumlaufbahn am 8. November 2008 gebracht. Die Mondmesssonde wurde erfolgreich am Südpol des Mondes um 20:31 Uhr, am 14. November 2008 abgesetzt.

Setzen Sie den **Cursor** auf den ersten Buchstaben (^). Um zum nächsten Satz zu springen, drücken Sie).

Chandrayaan-1 ist Indiens erste Mondmission, initiiert von der Nationalen Indischen Raumfahrtagentur, der ISRO (Indian Space Research Organisation). [D]ie unbemannte Monderforschungsmission umfasst einen Mondumkreiser und einen Stoßkörper. Der Raumflugkörper wurde durch eine modifizierte Version der PSLV XL am 22. Oktober 2008 vom Satish Dhawan Raumfahrtzentrum, Sriharikota, Andrah Pradesh um 6:23 Uhr IST (00:52 UTC) hochgeschossen. Der Raumflugkörper wurde erfolgreich in die Mondumlaufbahn am 8. November 2008 gebracht. Die Mondmesssonde wurde erfolgreich am Südpol des Mondes um 20:31 Uhr, am 14. November 2008 abgesetzt.

Wie cool ist das denn?

Um zum vorhergehenden Satz zu springen, drücken Sie (.

Na los, probieren Sie es aus, und erfahren Sie selbst, wie schnell Sie damit navigieren können. Nochmal, Sie können einen Zähler voranstellen, wie 3), um 3 Sätze weiterzuspringen.

Jetzt nehmen Sie mal den ganzen Text, und versuchen Sie sich absatzweise zu bewegen. Drücken Sie } um zum nächsten und {, um zum vorherigen Absatz zu springen.

Beachten Sie, daß die 'größeren' Klammern auf die größeren Textobjekte angewandt werden. Wenn Ihnen dies eh schon aufgefallen ist, na dann *Herzlichen Glückwunsch*, damit denken Sie nämlich bereits wie ein *Gewinner*, oder besser: *wie ein Vimmer*.

Nochmal: versuchen Sie erst gar nicht, sich diese Operatoren zu *merken*, versuchen Sie es, sich derart zur Gewohnheit zu machen, daß Sie diese Tasten ganz automatisch verwenden.

Für Details, sehen Sie sich auch `:help cursor-motions` an.

7.4 Setzen Sie eine Marke

Sie schreiben einen Text, aber plötzlich fällt Ihnen ein, daß Sie noch einen damit verbundenen Abschnitt im selben Dokument noch auf den neuesten Stand bringen müssen, aber Sie wollen Ihrer aktuellen Position nicht verlustig gehen, weil Sie später genau an dieser Stelle weitermachen wollen. Was tun Sie jetzt?

Normalerweise würde dies bedeuten, zu diesem Abschnitt zu fahren, ihn auf den neuesten Stand bringen, und dann zum Ausgangspunkt zurückzufahren. Alles ein bisschen viel Aufwand, und abgesehen davon neigen wir dazu zu vergessen, von wo wir gekommen sind.

Mit Vim können wir das alles ein bisschen intelligenter anstellen. Bewegen Sie den **Cursor** zur 3. Zeile des folgenden Textes (die Worte von John Lennon). Verwenden Sie `ma`, um eine Markierung mit dem Namen 'a' zu erstellen. Setzen Sie den Cursor hin, wo immer Sie wollen, z.B. `4j`.

Bin ganz gespannt auf meine nächst Enttäuschung. –Ashleigh Brilliant

Jedermanns Gedächtnis ist sein privates Schrifftum. –Aldous Huxley

Leben ist das, was mit dir gerade passiert, während du damit beschäftigt bist andere Pläne zu schmieden. –John Lennon

Das Leben ist wirklich einfach, aber wir bestehen darauf es kompliziert zu machen. –Konfuzius

Verweile nicht in der Vergangenheit, träume nicht von der Zukunft, fokussiere den Geist auf den gegenwärtigen Augenblick. –Buddha

Je mehr Entscheidungen man gezwungen ist alleine zu treffen, desto mehr gewahr wird man sich seiner Freiheit etwas auszuwählen. –Thornton Wilder

Drücken Sie **'a** (einfaches Gänsefüßchen gefolgt vom Namen der Markierung), und - Bitte sehr - schon springt Vim (zurück) zu der markierten Zeile.

Sie können hierfür jedes Alphabet nehmen (a-z A-Z), um eine Markierung zu kennzeichnen, was bedeutet, daß Sie bis zu 52 Markierungen für jede Datei vergeben können.

7.5 Herumspringen

Mit den verschiedenen Bewegungsbefehlen, die wir erlernt haben, wollen wir oft zum vorhergehenden, oder zum nächsten Ort nach einem Bewegungsbefehl springen. Um dies zu tun, drücken Sie einfach **ctrl-o** um zur vorhergehenden Position, und **ctrl-i**, um wieder vorwärts, zur nächsten Position zu springen.

7.6 Textteile

Es gibt zahlreiche Wege, Textobjekte im Vim zu definieren, um sie anschließend einem Befehl zu übergeben. Sie wollen z.B. einen Textteil sichtbar selektieren und anschließend alle Großbuchstaben in Kleinbuchstaben, bzw. umgekehrt, mit dem **~** Operator (Tildezeichen) umwandeln.

Öffnen Sie die **dapping.txt**-Datei, die wir im vorhergehenden Kapitel erstellt haben. Benutzen Sie die zahlreichen Tastaturbefehle, um auf den ersten Buchstaben des Wortes **'dapper'** im zweiten Absatz zu springen. Tipp: versuchen Sie **}**, **j**, **w**.

Dapping means being determined about being determined and being passionate about being passionate.

Be a dapper.

Drücken Sie **v**, um in den visuellen Modus zu gelangen, und tippen Sie **ap**, um **'a'** **'p'**aragraph (Absatz) zu selektieren. Drücken Sie **~**, um die Groß- / Kleinbuchstaben zu verkehren. Wenn Sie die Selektion aufheben wollen, drücken Sie einfach **<ESC>**.

Dapping means being determined about being determined and being passionate about being passionate.

bE A DAPPER.

Andere Textobjektgedächtniskürzel sind `aw`, was 'a' 'w'ord bedeutet, `a"` heißt eine in Anführungsstriche gesetzte Zeichenfolge (wie „Dies ist eine in Anführungsstriche gesetzte Zeichenfolge“), `ab` bedeutet 'a' 'b'lock, was wiederum heißt, alles innerhalb eines Klammernpaares, usw.

Sehen Sie sich hierzu auch `:help object-motions` und `:help text-objects` für nähere Einzelheiten an.

7.7 Zusammenfassung

Wir haben die mannigfachen Möglichkeiten gesehen, die uns Vim bereitstellt, um uns innerhalb eines Textes zu bewegen (navigieren). Es ist nicht nötig, sich jeden einzelnen dieser Navigationsbefehle im Kopf zu behalten, vielmehr sollten Sie sich diese Befehle angewöhnen, wann immer Sie die Gelegenheit dazu haben, speziell die Befehle, die für Sie am wichtigsten sind. Und wenn Sie sich daran gewöhnt haben, werden auch die Handbewegungen weniger, Sie werden schneller, und letztendlich verbringen Sie mehr Zeit damit, darüber nachzudenken, was Sie gerade schreiben, als darüber, welches Programm Sie zum Schreiben nutzen.

Sehen Sie sich hierzu `:help various-motions` als auch `:help motion` für interessantere Bewegungsarten an.

8 Hilfe

8.1 Einführung

Vim kennt eine solche Fülle von Befehlen, Tastaturkürzel, Zwischenspeicher, usw. Es ist unmöglich sich daran zu erinnern, wie die alle funktionieren. Tatsächlich ist es nicht mal nützlich, alle zu kennen. Idealerweise wissen Sie, wo Sie nach einer bestimmten Funktion in Vim zu suchen haben, wann immer Sie sie brauchen.

Sie möchten es z.B. vermeiden, jedesmal einen langen Namen tippen zu müssen, und dann fällt Ihnen siedendheiß ein, daß es eine Abkürzungsfunktion in Vim gibt, die Ihnen hilft genau dieses Problem zu lösen, aber Sie wissen nicht mehr, wie die Funktion angewandt wird. Was tun Sie?

Jetzt schauen wir uns mal die verschiedenen Möglichkeiten an, Hilfe zu finden, wie man Vim benutzen kann.

8.2 Der `:help`-Befehl

Die erste und wichtigste Anlaufstelle, zu versuchen, Hilfe zu finden, ist die programmimmanente Dokumentation, und Vim besitzt eine der bestverständlichsten Benutzerhandbücher, die ich je gesehen habe.

In unserem Fall, tippen Sie einfach `:help abbreviation` und Sie werden zur Hilfe für Abkürzungen geführt, und hier können sie nun lesen, wie Sie die `:ab` und `:iab`-Befehle verwenden.

Manchmal ist es genau so einfach, wie eben. Wenn Sie nicht wissen wonach Sie suchen, dann geben Sie einfach `:help user-manual` ein und blättern sich durch die Inhaltsliste des gesamten Benutzerhandbuches, und lesen das Kapitel, von dem Sie glauben, daß es Sie der Lösung Ihres Problems, an dem Sie gerade arbeiten, am nächsten bringt.

8.3 Wie man das `:help topic` [→Thema] liest

Nehmen wir mal einen Textauszug von `:help abbreviate`:

```
:ab[breviate] [<expr>] {lhs} {rhs}
    add abbreviation for {lhs} to {rhs}. If {lhs} already
    existed it is replaced with the new {rhs}. {rhs} may
    contain spaces.
    See |:map-<expr>| for the optional <expr> argument.
```

Beachten Sie, daß es einen festgelegten Standard bezüglich des Vim-Hilfesystems gibt, um es uns leicht zu machen, genau die Teile herauszufinden, die wir benötigen, anstatt zu versuchen, den Befehl komplett zu verstehen.

Die erste Zeile erklärt die Schreibweise, d.h., wie der Befehl anzuwenden ist.

Die eckigen Klammern in `:ab[breviate]` weisen darauf hin, daß der letzte Teil des vollen Names beliebig ist. Sie müssen mindestens `:ab` eingeben, damit Vim den Befehl als solchen erkennen kann. Sie können aber auch `:abb` oder `:abbr` oder `:abbre` usw. verwenden, bis zum vollen Namen `:abbreviate`. Die meisten Leute neigen zur kürzest möglichen Form.

Die eckigen Klammern in `[<expr>]` weisen, auch hier, darauf hin, daß 'expression' wahlfrei ist.

Die geschweiften Klammern in `{lhs} {rhs}` besagen, daß es sich hier um Platzhalter für aktuell einzufügende Argumente handelt. Die Namen sind Abkürzungen für 'linker Hand' bzw. 'rechter Hand' [left hand side / right hand side].

Was auf die erste Zeile folgt, ist ein eingerückter Absatz, der kurz erklärt, was dieser Befehl tut.

Beachten Sie, den zweiten Absatz, der Sie auf weitere Informationen hinweist. Sie können den Cursor auf den Text zwischen den beiden Pipe-Symbolen `[|]` setzen und dann `ctrl-]` drücken, um dem Link des zugeordneten `:help topic` [Hilfethemas] zu folgen. Zurück gelangen Sie mit `ctrl-o`.

8.4 Das `:hilfeallgemeinerregulärausdruckanzeige`-Kommando

Ok - das war jetzt kein Übersetzungsgeniestreich. Tatsächlich steht 'grep' aber für **g**eneral **r**egular **e**xpression **p**rint (also 'Anzeige' / Ausgabe eines allgemeinen, regulären Ausdrucks). `:helpgrep` ist demzufolge eine Befehlskombination von `help` & `grep`.

Wenn Sie den Namen des Themas nicht kennen, dann können Sie das ganze Dokument nach einem Satz durchsuchen, mit Hilfe von `:helpgrep`. Angenommen Sie möchten wissen, wie Sie nach dem Beginn eines Wortes zu suchen haben, tippen Sie einfach `:helpgrep beginning of a word`.

Sie können **:cnext** und **:cprev** nutzen, um zum nächsten/vorhergehenden Teil der Dokumentation zu springen, wo dieser Satz auftaucht. Um eine komplette Liste zu sehen, wo der Satz jeweils auftaucht, nehmen Sie **:clist**.

8.5 Schnelle Hilfe

Erstellen Sie eine Datei (z.B. mit dem Namen quickhelp) mit Vim, und geben Sie den folgenden Text ein.

```
:let &keywordprg=':help'
```

Verlassen Sie den Einfügemodus mit **<ESC>** und geben Sie die gleiche Zeile im Befehlsmodus nochmal ein.

Setzen Sie nun den **Cursor** irgendwo auf das Wort **keywordprg** und drücken Sie einfach **K**. Sofort gelangen Sie zur Hilfe für dieses Wort. Dieses Tastaturkürzel verhindert, daß Sie **:help keywordprg** tippen müssen.

8.6 Internetforum und IRC (→Internet Relay Chat)

Sollten Sie immer noch nicht in der Lage sein, herauszufinden, was Sie tun wollen, dann wird das nächstbeste wohl sein, andere Vim-Benutzer zu kontaktieren um Ihnen auszu-helfen. Keine Bange, das ist ganz einfach, und die Hilfsbereitschaft anderer *'Vimmer'* ist erstaunlich.

Suchen Sie in einem ersten Schritt nach der Vim-Mailingliste, um herauszufinden, ob jemand Ihre Frage bereits beantwortet hat. Gehen Sie auf die Vim-Gruppen-Such-Seite³⁴, und geben Sie die Schlüsselwörter Ihrer Frage ein. Meistens werden viele allgemeine Fragen bereits beantwortet sein, da dies so eine hochfrequentierte Mailingliste ist, d.h. viele, viele Leute stellen in dieser Gruppe Fragen und geben Antworten.

Wenn Sie keine passende Antwort finden, dann besuchen Sie das Vim IRC-Forum. Öffnen Sie eine IRCAnwendung wie z.B. XChat³⁵ (gibt's für Windows, Linux, BSD) oder Colloguy³⁶ (für Mac OS X), verbinden Sie sich mit dem „FreeNode“-Netzwerk, betreten den **#vim** Kanal, stellen manierlich Ihre Frage, und warten dann, bis jemand antwortet. Meistens wird jemand innerhalb von Minuten antworten.

Wenn niemand Ihre Frage beantwortet, sind wahrscheinlich alle gerade beschäftigt. Dann stellen Sie Ihre Frage zu einem späteren Zeitpunkt nochmals oder formen die Frage so um, daß es anderen leicht fällt Ihnen zu helfen. Ansonsten hinterlassen Sie eine Nachricht in der oben erwähnten Mailingliste.

8.7 Zusammenfassung

Es gibt eine Fülle von Informationen, wie sich die Dinge bei der Benutzung des Vim erledigen lassen, und viele *Vimmer* helfen Ihnen dabei auch gerne. Die Vim-Gemeinschaft

³⁴<http://tech.groups.yahoo.com/group/vim/msearch.adv>

³⁵<http://www.silverex.org/download/>

³⁶<http://colloquy.info/>

ist eine der größten Stärken des VimEditors. Nutzen Sie also die bestehenden Hilfsquellen reichlich, und treten auch Sie dieser Gemeinschaft bei.

Wahres Entzücken liegt eher im *Herausfinden* als im *Wissen*. –Isaac Asimov

9 Grundlagen des Bearbeitens von Dateien [→Editing]

9.1 Einführung

Jetzt lernen wir mal die grundlegenden Editierungskommandos in Vim für das Lesen und Schreiben von Dateien, Herausschneiden/Kopieren/Einfügen/Zurücknehmen/Wiederherstellen und Suchen.

9.2 Lesen und Schreiben von Dateien

9.2.1 Zwischenspeicher

Wenn Sie eine Datei bearbeiten, bringt Vim den Textinhalt der Datei von der Festplatte in den Arbeitsspeicher des Computers. Dies bedeutet, dass eine Kopie der Datei im Speicher des Rechners angelegt wird, und jegliche Änderungen, die Sie vornehmen, auch im Arbeitsspeicher des Computers vollzogen, und Ihnen unmittelbar angezeigt werden. Nachdem Sie die Datei bearbeitet haben, können Sie sie abspeichern, was bedeutet, dass Vim den im Arbeitsspeicher befindlichen Text in die Datei auf der Festplatte zurückschreibt. Der Arbeitsspeicher des Rechners, der hier temporär genutzt wird, wird als Zwischenspeicher bezeichnet [„buffer“]. Es ist dieses nämliche Konzept, das uns zwingt, Dateien sämtlicher Editoren oder Textverarbeitungssysteme mit denen wir arbeiten zu „Speichern“.

Öffnen Sie Vim, schreiben Sie die Worte **Hallo Welt** und speichern Sie es als **hallo.txt** ab. Eine kleine Erinnerungshilfe, wie man dies tut, gibt's im Kapitel 'Erste Schritte'.

9.2.2 Auslagern

Nun werden Sie bemerkt haben, daß im selben Verzeichnis eine weitere Datei erstellt wurde, die sowas wie **.hallo.txt.swp** heißt. Führen Sie den folgenden Befehl aus, um den exakten Namen der Datei herauszufinden:

```
:swapname
```

Was ist das denn für eine Datei? Vim behält eine Sicherungskopie des Zwischenspeichers in einer Datei, die es in regelmäßigen Abständen auf die Festplatte schreibt, damit Sie, sollte doch einmal etwas schiefgehen (daß der Rechner oder sogar Vim selbst abschmiert) eine Sicherungskopie der Änderungen haben, seit Ihrem letzten Abspeichern der Originaldatei. Diese Datei nennt sich „Auslagerungs-Datei“, weil Vim den Inhalt des Zwischenspeichers im Arbeitsspeicher des Rechners kontinuierlich mit dem Inhalt der Datei auf der Festplatte austauscht. Sehen Sie hierzu auch **:help swap-file** für weitere Details.

9.2.3 Sichere meine Datei

Jetzt wo die Datei geladen wurde, lassen Sie uns eine geringfügige Änderung vornehmen. Drücken Sie die `~`-Taste [Das Tildezeichen kriegen Sie, indem Sie **AltGr** gedrückt halten und zweimal hintereinander die Tildetaste drücken!], um das Zeichen, auf dem der **Cursor** steht, von Großbuchstabe auf Kleinbuchstabe vice versa zu ändern. Sie werden bemerkt haben, daß Vim die Datei nun als *verändert* markiert hat (ein **+** Zeichen erscheint z.B. in der Titelleiste der grafischen Version des Vim). Jetzt können Sie die Datei mit einem anderen Editor öffnen, und überprüfen, daß noch nichts verändert wurde. Das heißt, Vim hat die Änderungen lediglich im Zwischenspeicher vorgenommen, aber noch nicht auf die Festplatte geschrieben.

Wir können den Zwischenspeicher in die Originaldatei auf der Festplatte zurückschreiben, indem wir dies ausführen:

```
:write
```

9.2.4 Hinweis

Um sich das Speichern einfacher zu machen, fügen Sie Ihrer **vimrc** folgendes hinzu:

```
" Speichern ctrl-s.  
nmap <c-s> :w<CR>  
imap <c-s> <Esc>:w<CR>a
```

Jetzt drücken Sie einfach nur noch **ctrl-s**, um die Datei abzuspeichern.

9.2.5 Arbeiten in meinem Verzeichnis

Vim startet voreingestellt in Ihrem Heimatverzeichnis, und alle Operationen werden in diesem Verzeichnis durchgeführt.

Um Dateien aus anderen Verzeichnissen zu öffnen, können Sie volle oder relative Pfadangaben benutzen, wie solche:

```
:e ../tmp/test.txt  
:e C:\shopping\monday.txt
```

Oder Sie veranlassen Vim, in dieses Verzeichnis zu wechseln:

```
:cd ../tmp
```

:cd ist die Abkürzung für *wechsle* Verzeichnis [**'c'**hange **'d'**irectory]

Um herauszufinden in welchem aktuellen Verzeichnis Vim gerade nach Dateien sucht:

```
:pwd
```

:pwd ist die Abkürzung für *zeige das aktuelle Verzeichnis* [**→'p'**rint **'w'**orking' **'d'**irectory]

9.2.6 Herausschneiden, Kopieren und Einfügen

Wie Sean Connery im Film *'Forrester gefunden'*³⁷ sagt:

Nicht denken - das kommt später. Schreibe Deinen ersten Entwurf mit dem Herzen. Verbesserungen machst Du mit dem Kopf. Der Anfang des Schreibens ist.. zu schreiben, nicht nachzudenken!

Wenn wir redigieren, ordnen wir die Absätze oder Abschnitte immer wieder um, d.h. wir müssen in der Lage sein Text herauszuschneiden, zu kopieren und einzufügen. Für Vim gilt hier eine leicht unterschiedliche Terminologie:

Klicki-Bunti	die Welt des Vim	Operation
herausschneiden	löschen	d
kopieren	'etwas herausziehen' 'to yank'	y
einfügen	einfügen 'to paste'	p

In der normalen Terminologie der grafischen Benutzeroberfläche, bedeutet Text *'herausschneiden'*, den Text zu entfernen und ihn ans *Klembrett zu schieben (* clipboard heißt wörtlich Klembrett, bezeichnet aber auch nur einen Bereich des Arbeitsspeichers.). Dieselbe Operation in Vim bedeutet, daß der Text aus dem Dateizwischenspeicher gelöscht, und in einem 'Register' (einen Teil des Computerarbeitsspeichers) gelagert wird. Und weil wir uns das Register, wo wir Text ablegen können, aussuchen können, nennt es sich Lösch-Operation.

Ganz vergleichbar, bedeutet in der normalen Terminologie der grafischen Benutzeroberfläche, 'Kopieren' von Text, daß eine Kopie des Textes ans Klembrett platziert wird. Vim tut dasselbe, es zieht den Text heraus (*to yank*) und legt ihn in einem Register ab. Keinen Bedeutungsunterschied im Sprachgebrauch gibt es für den Begriff Einfügen.

Wir haben gesehen, wie Sie Herausschneiden-/Kopieren-/und Einfüge-Operationen mit Vim erledigen. Aber wie legen Sie fest, auf welchen Text diese Befehle angewandt werden? Also - das haben wir ja bereits im vorhergehenden Abschnitt über Textobjekte behandelt.

Kombinieren wir nun die Befehlsschreibweise mit der Textobjektschreibweise, bedeutet dies, daß wir eine schier endlose Zahl an Möglichkeiten haben, um Texte zu manipulieren. Schauen wir uns mal ein paar Beispiele an.

Schreiben Sie diesen Text (genauso wie er hier steht) mit Vim:

```
This is the rthe first paragraph.
This is the second line.
```

```
This is the second paragraph.
```

Setzen Sie den **Cursor** an die oberste, linke Stelle, indem Sie **1G** drücken und **|** was Sie zur ersten Zeile, bzw. zur ersten Reihe bringt.

Nehmen wir mal den ersten Fall: Ein 'r' haben wir zuviel getippt, das müssen wir löschen. Drücken Sie **3w**, um drei Worte vorwärts zu springen.

³⁷<http://www.imdb.com/title/tt0181536/>

Jetzt müssen wir einen Buchstaben an der aktuellen Position des **Cursors** löschen. Beachten Sie, daß es hier zwei Teile gibt:

Operation	Text Objekt/Bewegung
Löschen	Ein Buchstabe an aktueller Position
d	l

Um einen Buchstaben zu löschen, drücken wir also lediglich **dl** ! Wie Sie sehen, benutzen wir hier **l** , obwohl dies ein Bewegungsbefehl ist.

Jetzt fällt uns auf, daß das ganze Wort eigentlich überflüssig ist, da wir es doppelt haben. Überlegen Sie mal ganz scharf; was wäre wohl die schnellste Tastenkombination, um dies zu korrigieren?

Denken Sie ruhig ein Weilchen darüber nach, um das selber rauszukriegen. Wenn nicht, lesen Sie bitte weiter.

Operation	Text Objekt/Bewegung
Löschen	Ein Buchstabe an aktueller Position
d	w

Also: **dw** drücken, und – Voila! – schon haben Sie ein Wort gelöscht. So einfach und wunderschön. Die Schönheit an solch einfachen Konzepten liegt aber gerade darin, daß sie miteinander kombiniert werden können, um einen derartigen Bereich an Möglichkeiten bereitzustellen.

Wie erreichen wir nun dasselbe für Zeilen? Nun, Zeilen sind in Vim etwas ganz besonderes, weil Zeilen üblicherweise das sind, wie wir über unseren Text denken. Kurz gesagt, wenn Sie den Befehlsnamen zweimal verwenden, wird er auf die Zeile angewandt. Also **dd** löscht die aktuelle Zeile und **yy** zieht die aktuelle Zeile heraus (*kopiert sie*).

Unser Beispiel in Vim sollte nun so aussehen:

```
This is the rthe first paragraph.
This is the second line.
```

```
This is the second paragraph.
```

Gehen Sie, durch drücken von **j** auf die zweite Zeile, und drücken Sie **dd** , jetzt sollte sie gelöscht sein. Sie sollten jetzt das hier sehen:

```
This is the rthe first paragraph.
```

```
This is the second paragraph.
```

Betrachten wir mal einen größeren Fall: Wie ziehen wir den aktuellen Abschnitt heraus (*kopieren*)?

Operation	Text Objekt/Bewegung
Kopieren	aktueller Absatz
y	ap

yap kopiert also den aktuellen Absatz. Jetzt wo wir den Text kopiert haben – wie fügen wir ihn wieder ein? Einfach mit **'p'**!

Sie sollten jetzt dies hier sehen:

```
This is the rthe first paragraph.
This is the rthe first paragraph.
```

```
This is the second paragraph.
```

Beachten Sie, daß die Leerzeile auch kopiert wird, wenn wir **yap** ausführen, **p** fügt diese zusätzliche Leerzeile mit hinzu.

Es gibt zwei Einfügemethoden, die genauso funktionieren, wie die beiden Einfügearten, die wir vorher schon gesehen haben:

- p (Kleinbuchstabe) Einfügen nach der aktuellen Schreibmarkenposition
- P (Großbuchstabe) Einfügen vor der aktuellen Schreibmarkenposition

Wenn man diese Idee nun erweitert, dann ergeben sich daraus ganz starke Möglichkeiten. Zwei Buchstaben vertauschen? Drücken Sie **xp**.

- x→ein Buchstabe an der aktuellen Position löschen
- p→einfügen nach der aktuellen Position

Wie vertauschen Sie zwei Wörter? Drücken Sie **dwwP**.

- d→löschen
- w→ein Wort
- w→zum nächsten Wort bewegen
- P→einfügen vor der aktuellen Position

Das Wichtige hierbei ist, diese Befehle *nicht* auswendig zu lernen. Diese Befehlskombinationen und TextObjekt/Bewegungen sollten Ihnen in Fleisch und Blut übergehen, ohne daß Sie darüber nachdenken müssen. Und genau das passiert auch, wenn Sie sie sich durch Benutzung zur Gewohnheit machen.

9.3 Markieren Sie Ihr Gebiet

Sie schreiben gerade, und plötzlich bemerken Sie, daß Sie Sätze in einem vorhergehenden Abschnitt ändern müssen, um das zu unterstreichen, was Sie in diesem Abschnitt schreiben. Das Problem hierbei ist, daß Sie sich daran erinnern müssen, wo Sie jetzt gerade sind, damit Sie später an diese Stelle zurückkehren können. Kann Vim diese Arbeit nicht für mich übernehmen? Vim kann - mit der Verwendung von Markierungen.

Eine Markierung können Sie erstellen, indem Sie **m** drücken, gefolgt vom Namen der Markierung, die ein einzelner Buchstabe ist, von **a-z** bzw. **A-Z**. Drücken Sie z.B. **ma**, wird eine Markierung namens **'a'** erstellt.

Das Drücken von 'a bringt den Cursor auf die Zeile der Markierung. Drücken Sie 'a werden Sie exakt zur Zeile und Reihe der Markierung gebracht.

Das beste daran ist, daß Sie jederzeit danach zu diesen Markierungen springen können. Sehen Sie sich hierzu auch `:help mark-motions` an.

9.4 Eine Zeitmaschine mit 'zurücknehmen' / 'wiederherstellen' [→undo/redo]

Nehmen wir mal an, Sie redigieren einen Absatz, aber es endet damit, daß Sie sich böse verzetteln, mit dem was Sie versucht haben umzuändern, und jetzt möchten Sie zu Ihrem früheren Ausgangspunkt zurückkehren. An diesem Punkt können wir nun das, was wir eben getan haben *zurücknehmen*. Wenn wir aber wieder zurückkehren wollen, zu dem was wir jetzt haben, dann können wir die Änderungen, die wir gemacht haben, *wiederherstellen*. Beachten Sie, eine Änderung, bedeutet eine Änderung am Text. Bewegungen der Schreibmarke, und andere Dinge, die nicht in direktem Zusammenhang mit dem Text stehen, werden nicht berücksichtigt.

Nehmen Sie mal diesen Text hier:

Ich habe für mich einen Satz geprägt - 'Cut to the G':

1. Konzentrieren →concentrate
2. Verstehen →understand
3. Nachdenken →think
4. Dinge erledigen →get things done

Schritt 4 gefällt Ihnen vielleicht am besten, aber die Schritte 2 und 3 sind genauso wichtig. Wie Abraham Lincoln einmal sagte: „Wenn ich acht Stunden Zeit habe, um einen Baum zu fällen, dann würde ich sechs Stunden meine Axt schärfen.“ Und um zu diesem Punkt zu kommen, müssen Sie Punkt 1 beherzigen, was dann auf eine Sache hinausläuft - *Es ist die Welt der Gedanken*. Deshalb ist es so schwierig.

Editieren wir nun die erste Zeile:

1. Drücken Sie **S**, um die ganze Zeile zu ersetzen (*substitute*).
2. Geben Sie den Text **Durch reifliche Überlegung, habe ich mir einen neuen Satz geprägt, um meine Herangehensweise zu festigen:.**
3. Drücken Sie **<ESC>**

Jetzt denken Sie über die Änderung, die wir gerade gemacht haben nach. Ist der Satz besser? Hmm.. war der Text vorher besser? Wie schalten wir jetzt hin und her?

Drücken Sie **u**, um die letzte Änderung zurück zu nehmen, und zu sehen, was wir vorher hatten. Sie werden jetzt den Text **Ich habe für mich einen Satz geprägt - 'Cut to the G':** sehen. Um zur letzten Änderung zurückzukommen, drücken Sie **ctrl-r**,

um nun die Zeile **Durch reifliche Überlegung, habe ich mir einen neuen Satz geprägt, um meine Herangehensweise zu festigen:** zu sehen.

Ganz wichtig hierbei ist, daß Vim unbegrenzt 'zurücknehmen' / 'wiederherstellen'-Stapel zur Verfügung stellt, in der Praxis ist es aber durch die **undolevels**-Einstellungen in Vim, und der Menge an zur Verfügung stehendem Arbeitsspeicher Ihres Rechners begrenzt.

Jetzt wollen wir uns mal ein paar Dinger reinziehen, die die fortschrittliche 'zurücknehmen' / 'wiederherstellen'-Funktion erst so richtig herausstellt. Eine Sache, auf die andere Editoren neidisch sein werden: Vim ist nicht nur Ihr Editor, sondern er arbeitet auch wie eine Zeitmaschine.

Zum Beispiel,

```
:earlier 4m
```

wirft Sie zeitlich 4 Minuten zurück, d.h. zum Status vor 4 Minuten.

Die Stärke hierbei ist, daß es sämtlichen 'rückgängig' / 'wiederholen'-Aktionen überlegen ist. Wenn Sie z.B. eine Änderung machen, dann rückgängig machen, und dann weiter redigieren, dann ist diese Änderung mit dem einfachen **u**-Operator, nie wiederzuerlangen. Aber mit Vim ist es möglich den **:earlier**-Befehl zu benutzen.

Sie können in der Zeit auch vorwärts gehen:

```
:later 45s
```

Was Sie zum Zustand 45 Sekunden später bringen wird. Eine einfachere Herangehensweise, 5 Änderungen zurückzugehen, ist natürlich:

```
:undo 5
```

Hiermit können Sie sich die Baumstruktur ansehen:

```
:undolist
```

Sehen Sie sich **:help :undolist** an, um eine Erklärung des Ergebnisses dieses Befehls zu erhalten.

Sehen Sie sich **:help :undo-redo** und **:help usr_32.txt** an, für weitere Einzelheiten.

9.5 Eine starke Suchmaschine aber kein .com

Vim beinhaltet eine starke Suchmaschine, die Sie nutzen können, um genau das zu finden wonach Sie suchen. Es dauert ein wenig, bis man sich an die ihr innewohnenden Stärken gewöhnt hat. Fangen wir also an.

Kommen wir zu unserem vertrautem Beispiel zurück:

[Hier nehme ich den Originaltext, sonst lässt sich nämlich die Übung kaum machen. d.Übersetzer]

I have coined a phrase for myself 'CUT to the G':

1. Concentrate
2. Understand
3. Think
4. Get Things Done

Step 4 is eventually what gets you most moving, but Steps 2 and 3 are equally important. As Abraham Lincoln once said If I had eight hours to chop down a tree, I'd spend six hours sharpening my axe. And to get to this stage, you need to do Step 1 which boils down to one thing I'ts all in the mind. That's why it's so hard.

Mal angenommen, wir wollen nach dem Wort Step suchen. Tippen Sie dazu im Normalmodus `/Step<RET>` (`→RETURN`-Taste drücken). Das bringt Sie zum ersten Erscheinen dieser Buchstabenansammlung. Drücken Sie `n`, um zum nächsten Treffer zu gelangen und `N` für die entgegengesetzte Suchrichtung, d.h. den vorherigen Treffer.

Was, wenn Sie nur einen Teil des Satzes kennen, oder die exakte Schreibweise nicht kennen? Wäre es nicht hilfreich, wenn Vim mit der Suche schon mal anfangen könnte, während Sie noch den Suchbegriff eintippen?

Mit

```
set incsearch
```

können Sie das einschalten.

Sie können Vim auch beibringen, daß er die Groß-, Kleinschreibung Ihres Suchbegriffes ignoriert:

```
set ignorecase
```

Oder Sie nehmen:

```
set smartcase
```

Wenn Smartcase eingeschaltet ist:

- Wenn Sie nach `/step`, also alles in Kleinbuchstaben eingegeben suchen, dann wird nach jeglicher Kombination in Klein- und Großbuchstaben gesucht. Das wird z.B. alle vier folgenden Treffer liefern - „Step“, „Stephen“, „stepbrother“, „misstep“.
- Wenn Sie nach `/Step` suchen, der eingegebene Begriff enthält also einen Großbuchstaben, dann wird NUR nach Text gesucht, das dem Muster exakt entspricht. Treffer wären, in diesem Beispiel also, „Step“ und „Stephen“, aber *nicht* „stepbrother“ oder „misstep“.

Hinweis Ich rate Ihnen an, diese beiden Zeilen in Ihre vimrc-Datei (erkläre ich später, aber eine schnelle Einführung erhalten Sie mit `:help vimrc-intro`) zu übernehmen, damit dies von vornherein eingeschaltet ist.

Jetzt, da wir 'Suchen', dem Prinzip nach verstanden haben, erforschen wir die wahren Suchfähigkeiten des Vim. Beachten Sie als erstes, was Sie Vim liefern, kann nicht nur ein einfacher Satz, sondern ein „Ausdruck“ sein. Ein Ausdruck erlaubt Ihnen nämlich die *Art* von Text festzulegen, nach dem gesucht wird, und nicht bloß den genauen Text.

Sie werden bemerkt haben, daß `/step` Ihnen **steps** genauso als Treffer anzeigt wie **step** und sogar **footstep**, wenn es ein solches Wort gibt. Was, wenn Sie nun nach dem exakten Wort **step** suchen möchten, ohne Teil eines anderen Wortes zu sein? Dann nehmen Sie das hier `/\ <step\ >`. Das `\ <` und `\ >` weisen jeweils auf den Anfang und das Ende eines „Wortes“ hin.

Oder Sie wollen mal nach einer Nummer suchen. `/\d` wird nach einer 'Ziffer' suchen. Aber eine Zahl ist nur eine Gruppe zusammengesetzter Ziffern. Damit können wir festlegen eine oder mehr Ziffern zusammen als `/\d\+`. Wenn wir nach keinen bzw. mehreren Ziffern suchen, nehmen wir den `*` anstelle des `+`.

Es gibt eine ganze Bandbreite solch zauberhaften Zeugs, das wir uns für unsere Suchmuster zu eigen machen können. Schlagen Sie es nach `:help pattern`.

9.6 Zusammenfassung

Wir haben einige der grundlegenden Editierungskommandos erforscht, die wir bei unserer täglichen Arbeit mit Vim nutzen werden. *Ganz wichtig, daß Sie diese Konzepte nochmals durchgehen, und sie sich zur Gewohnheit machen.* Es ist nicht wichtig, jeden einzelnen Befehl mit all seinen Optionen und Feinheiten auswendig zu kennen. Wenn Sie nur wissen wie Sie den Befehl verwenden, und wenn Sie wissen wo Sie zu suchen haben, für das was Sie gerade brauchen, dann sind Sie eine richtiger *Vimmer*.

Jetzt legen Sie aber los – fangen Sie an zu editieren!

10 Vielfältigkeit

10.1 Einleitung

In diesem Kapitel schauen wir uns an. Wie Vim uns bei der Arbeit mit verschiedenen Teilen einer Datei, verschiedenen Dateien, diversen „Fenstern“ und verschiedenen „tabs“ zum mehrere Dinge gleichzeitig in der Hand zu haben unterstützt.

10.2 Viele Abschnitte verwenden Falten

Wenn Sie ein langes Dokument bearbeiten, wäre es dann nicht einfacher Sie könnten alle Abschnitte schließen, und sich auf eines zu einer Zeit konzentrieren?

Das nennen wir in Vim Zusammenfalten.

Nehmen wir das Beispiel, wo Ihr Dokument derart strukturiert ist, daß jede Textebene ein Stück weiter eingerückt ist, so wie das folgende Stück Text:

Book I

The Shadow of the Past

Three Rings for the Elven-kings under the sky,
Seven for the Dwarf-lords in their halls of stone,
Nine for Mortal Men doomed to die,
One for the Dark Lord on his dark throne
In the Land of Mordor where the Shadows lie.
One Ring to rule them all, One Ring to find them,
One Ring to bring them all and in the darkness bind them
In the Land of Mordor where the Shadows lie.

Three is Company

The Road goes ever on and on
Down from the door where it began.
Now far ahead the Road has gone,
And I must follow, if I can,
Pursuing it with weary feet,
Until it joins some larger way,
Where many paths and errands meet.
And whither then? I cannot say.

Hinweis Dieser Text stammt von WikiQuote³⁸.

Nachdem Sie diesen Text eingetippt haben, geben Sie ein `:set foldmethod=indent`, platzieren Ihren **Cursor** auf den Text, den Sie einrücken möchten, drücken `zc` und sehen den Text nun einfalten. Mit `zo`, klappen Sie den Text wieder aus.

Die grundlegenden Kommandos sind `zo` und `zc`, mit denen wir Einfaltungen öffnen bzw. schließen. Sie können `za` benutzen, um zwischen öffnen/schließen einer Einfaltung hin- und herzuwechseln (`'a'`lternate). Sie können es sich sogar noch einfacher machen, indem Sie die Leertaste im Normalmodus nutzen, um eine Faltung zu öffnen und zu schließen.

Persönlich bevorzuge ich die Leertaste als Tastaturkürzel, um Einfaltungen zu öffnen und zu schließen. Dies erreichen Sie, indem Sie folgendes in Ihre vimrc-Datei hinzufügen:

```
:noremap <space> za
```

³⁸http://en.wikiquote.org/wiki/The_Fellowship_of_the_Ring

Einfaltungen sind ein ganz großes Thema für sich, mit mehreren Möglichkeiten (händisch, Markierung, Ausdruck), und zahlreichen Möglichkeiten Hierarchien von Einfaltungen zu öffnen und zu schließen, usw. Schauen Sie sich hierzu auch `:help foldings` an.

10.3 Mehrere Zwischenspeicher

Angenommen Sie wollen mehr als eine Datei zur selben Zeit, mit demselben Vim editieren. Wie machen Sie das?

Rufen Sie sich in Erinnerung, daß Vim Dateien in Zwischenspeicher lädt. Vim kann auch mehrere Zwischenspeicher zur selben Zeit laden. Damit können Sie also mehrere Dateien zur gleichen Zeit geöffnet haben, zwischen denen Sie hin- und herschalten können.

Sagen wir mal, Sie haben zwei Dateien, `part1.txt` und `part2.txt`:

`part1.txt`

```
I have coined a phrase for myself - 'CUT to the G':
```

1. Concentrate
2. Understand
3. Think
4. Get Things Done

`part2.txt`

```
Step 4 is eventually what gets you moving, but Steps 2 and 3 are equally
important. As Abraham Lincoln once said "If I had eight hours to chop down
a tree, I'd spend six hours sharpening my axe." And to get to this stage,
you need to do Step 1 which boils down to one thing - It's all in the mind.
That's why it's so hard.
```

Führen Sie nun `:e part1.txt` aus, und dann `:e part2.txt`. Machen Sie sich bewußt, daß Sie nun die zweite Datei zum Editieren geöffnet haben. Wie schalten Sie nun rüber zur ersten Datei? In diesem besonderen Fall können Sie einfach dies ausführen `:b 1`, um die Zahl des Zwischenspeichers zu wechseln ('b'uffer Nummer '1'). Sie können auch `:e part1.txt` ausführen, um den existierenden Zwischenspeicher sichtbar zu machen.

Sie sehen jetzt, welche Zwischenspeicher geladen wurden und, im Zusammenhang damit, welche Dateien gerade bearbeitet werden, indem Sie dies ausführen `:buffers` oder in einer kürzeren Form `:ls` (\rightarrow list buffers).

Zwischenspeicher werden automatisch entfernt, wenn Sie Vim schließen. Sie müssen also nichts besonderes mehr tun, außer sicher zu gehen, daß Sie Ihre Dateien abgespeichert haben. Wie auch immer: Wenn Sie einen Zwischenspeicher wirklich entfernen wollen, z.B. um Arbeitsspeicher freizugeben, dann können Sie `:bd 1` nehmen, (*bd* \rightarrow buffer delete) für Zwischenspeicher Nummer 1, usw.

Sehen Sie sich auch `:help buffer-list` an, für die zahlreichen Dinge, die Sie mit Zwischenspeicher tun können.

10.4 Mehrere Fenster

Wir haben gesehen, wie wir mehrere Dateien gleichzeitig bearbeiten, was aber, wenn wir zwei verschiedene Dateien gleichzeitig sehen wollen? Sie möchten z.B. zwei verschiedene Kapitel Ihres Buches geöffnet vor sich haben, damit Sie das zweite Kapitel so schreiben können, daß es mit den Worten und Beschreibungen des ersten Kapitels übereinstimmt. Oder Sie möchten etwas aus der ersten, in die zweite Datei Kopieren / Einfügen.

Im letzten Abschnitt, haben wir dieselbe Ansicht benutzt, um mehrere Zwischenspeicher zu bearbeiten. Vim nennt diese Ansichten Fenster. Der Begriff Fenster sollte hier **nicht** mit dem Fenster Ihrer Desktop-Anwendung verwechselt werden, womit man in der Regel das ganze Programm meint. Bei 'Fenster' denken Sie bitte einfach an 'Ansichten' auf verschiedene Dateien.

Nehmen wir also nochmal dieselben Textbeispiele aus dem vorherigen Abschnitt `part1.txt` und `part2.txt`. Laden Sie zuerst `part1.txt` mit `:e part1.txt`.

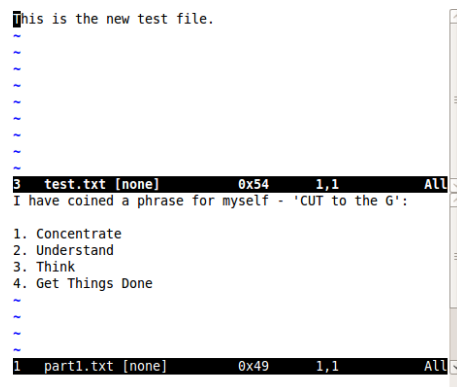
Jetzt lassen Sie uns einen neuen Zwischenspeicher öffnen, indem wir das Fenster teilen um ein neues Fenster zu erstellen – führen Sie `:new` aus.

Jetzt sollten Sie in der Lage sein, im neuen Zwischenspeicher, im neuen Fenster, jegliches normales Editieren zu bewerkstelligen, abgesehen davon, daß Sie den Text nicht abspeichern können, weil mit dem Zwischenspeicher noch kein Name verknüpft ist. Um den Zwischenspeicher abzuspeichern, nehmen Sie `:w test.txt`.

Wie wechseln Sie zwischen diesen beiden Fenstern hin und her? Benutzen Sie einfach `ctrl-w` *<Bewegungstaste>*, um zwischen den Fenstern umzuschalten. Bewegungstaste meint hier `h`, `j`, `k`, `l` oder sogar die Pfeiltasten (lediglich `↑↓` macht in diesem Beispiel Sinn). Merken Sie sich, `ctrl-w`-Operationen nehmen Bezug auf 'Fenster' (→'w'indows).

Als ein schneller Tastaturschlüssel können Sie `ctrl-w` auch zweimal drücken (also `ctrl-w ctrl-w`), um durch alle geöffneten Fenster zu zirkulieren.

Ein Sonderfall, bei dem mehrere Fenster nützlich sind, liegt vor, wenn Sie sich zwei verschiedene Teile der selben Datei gleichzeitig ansehen wollen. Führen Sie einfach `:sp` aus, um ein geteiltes (→'sp'lit) Fenster zu erstellen, und dann können Sie jedes Fenster an eine andere Position rollen, und weiterarbeiten. Weil beide „Fenster“ im gleichen Zwischenspeicher liegen, werden Änderungen des einen Fensters unmittelbar im anderen Fenster abgebildet. Alternativ zu `:sp` können Sie auch `ctrl-w s` benutzen.



Eine vertikale Aufteilung bekommen Sie mit `:vsp` oder `ctrl-w v`. Um ein Fenster zu schließen, führen Sie, wie gewohnt, `:q` aus.

Nachdem wir nun gesehen haben, wie man mehrere Fenster öffnet und benutzt, wollen wir mit der Ansicht noch ein bisschen weiter herumspielen.

- Nehmen wir mal an, Sie haben zwei geteilte Fenster, die Sie aber verkehren möchten, so daß Sie Ihre Augen auf den oberen bzw. unteren Teil Ihres Bildschirms lenken können, gemäß Ihrer Voreinstellung. Drücken Sie `ctrl-w r`, um die Fenster zu vertauschen (`'r'`otate).
- Wollen Sie das aktuelle Fenster zur höchstmöglichen Stelle bringen? Drücken Sie `ctrl-w K`.
- Wollen Sie die Größe eines Fensters verändern, um es größer oder kleiner zu machen? Führen Sie `:resize 10` aus, um die Ansicht auf 10 Zeilen zu bringen, usw.
- Wollen Sie das aktuelle Fenster auf Maximalgröße bringen, damit Sie sich besser konzentrieren können? Drücken Sie `ctrl-w _`. Beim Unterstrich denken Sie einfach daran, daß die anderen Fenster so klein wie möglich werden sollen.
- Wollen Sie die Fenster wieder auf 'gleiche' Höhe bringen? Drücken Sie `ctrl-w =`.

Sehen Sie sich hierzu `:help windows` an, um nähere Einzelheiten zu erfahren, was Sie mit Fenstern alles tun können.

10.5 Mehrere Tabs (→Reiter)

Wenn Sie Firefox benutzen, haben Sie vielleicht schon einmal die Möglichkeit der **Tabs** genutzt, die es Ihnen, in einem einzigen Fenster erlauben, mehrere Webseiten aufzurufen, womit Sie zwischen diesen Webseiten hin- und herschalten können, ohne sich darüber den Kopf zu zerbrechen, zwischen mehreren Fenstern umzuschalten. Tja also, mit Vim funktionieren diese **Tabs** ganz genauso, nur mit dem Unterschied, daß sie *'Tab-Seiten'* genannt werden.

Führen Sie aus `:tabnew` um einen neuen **Tab** mit neuem Zwischenspeicher zu öffnen (entsprechend zu `:new`). Wie wechseln Sie zwischen den **Tabs**? Durch drücken von `gt`, um zum nächsten **Tab** zu gehen (`'g'o 't'ab`), und `gT`, um die Laufrichtung zu ändern (also zum vorherigen **Tab**).

Ich persönlich bevorzuge die Tasten `alt-j` und `alt-k` für die gleichen Operationen, die analog für die Buchstabenbewegungen (`j` und `k`) gelten, und auch `ctrl-w j` und `ctrl-w k`, um Fenster (horizontal) zu teilen. Um dies zu aktivieren, fügen Sie die folgenden Zeilen in Ihre *vimrc* ein:



```
" Tastaturkürzel, um sich zwischen verschiedenen Reitern zu bewegen
" Alt-j um zum Reiter links zu gelangen
noremap <A-j> gT
" Alt-k um zum rechten Reiter zu springen
noremap <A-k> gt
```

Um einen **Tab** zu schließen, führen Sie **:tabc** oder **:q** aus.

Sie können sogar Texte, die in einem neuen Fenster geöffnet würden, stattdessen in einem neuen **Tab** öffnen. Zum Beispiel **:help tabpage** öffnet die Hilfe in einem horizontal geteilten Fenster. Um es aber in einem neuen **Tab** zu sehen, nehmen Sie das hier **:tab help tabpage**.

Wenn Sie **Tabs** umordnen möchten, dann machen Sie das mit **:tabmove**. Um z.B. den ersten **Tab** an die erste Stelle zu schieben, benutzen Sie **:tabmove 0** usw.

Sehen Sie sich **:help tabpage** für weitere Einzelheiten zu **Tab**-Seiten und anderen Operationen, die Sie damit ausführen können an, wie **:tabdo** um an allen geöffneten **Tab**-Seiten zu arbeiten. Für die individuelle Gestaltung der Titelleiste der **Tab**-Seiten gehen Sie zu **:help setting-guitablabe** usw.

10.6 Zusammenfassung

Vim bietet viele Möglichkeiten um mehrere Dateien gleichzeitig zu bearbeiten Zwischen-speicher, Fenster und **Tabs**. Diese Funktionen zu nutzen, hängt ganz von Ihrem persönlichen Geschmack ab. Wenn Sie z.B. mehrere **Tabs** benutzen, umgehen Sie damit die Benutzung von mehreren Fenstern. Wichtig ist, das zu nutzen, was für Sie am angenehmsten und bequemsten ist.

11 Persönliche Informationsverwaltung

11.1 Einführung

Ein Kapitel über 'persönliche Informationsverwaltung' in einem Buch über ein Editor-programm befremdet ein wenig. Nicht wahr? Nun, es gibt jede Menge „professionelle Programme“, die eine persönliche Informationsverwaltung für sich proklamieren. Jetzt wollen wir mal sehen, ob wir einen einfachen Texteditor wie Vim, zu diesem Zweck benutzen können.

Bei persönlicher Informationsverwaltung geht es darum alle Sie betreffenden Infor-mationen zu organisieren – als da wären eine '*zu erledigen-Liste*' (\rightarrow *todo-list*), Tage-bucheintragen, Referenzmaterial (wichtige Telefonnummern), Schmierzettel, usw. Dies alles an einem schönen Ort zu verstauen, kann schon verdammt praktisch sein. Dies, und ein paar andere **Plugins**, schauen wir uns jetzt mal mit Vim an.

Ich glaube ein persönliches Informationsverwaltungssystem organisiert man am bes-ten wie ein *Wiki*. Ein *Wiki* ist eine schnelle Methode, verschiedene Dokumente zu ver-knüpfen, die zwar miteinander in Zusammenhang stehen, davon unabhängig aber auch

für sich selbst stehen. Und so überrascht es nicht, daß das Wort *'wiki'*, in der hawaiianischen Sprache *'schnell'* bedeutet. Denken Sie an eine Webseite – es gibt eine *'Heimatseite'* (→Homepage), und es gibt, im Zusammenhang stehende Seiten, dargestellt als [Links](#), und jede dieser Seiten hat ihren eigenen Inhalt, kann aber wiederum auf andere [Links](#) verweisen. Ist das nicht eine einfache Methode, Webseiten zu organisieren? Wie wärs jetzt, wenn Sie dies auch mit Ihren persönlichen Informationen tun könnten? Lesen Sie sich mal diesen *LifeHack*-Artikel durch *'Wikify Your Life: How to Organize Everything'*³⁹. Da gibt es ein paar großartige Beispiele, was Sie alles tun können.

Aber braucht es dafür wirklich ein besonderes *Wiki-Programm*? Was, wenn Sie das gleiche, mit einfachen Textdateien, bei der Benutzung von Vim tun könnten? Tauchen wir ab.

11.2 Viki installieren

Hinweis Das `$vimfiles`-Verzeichnis steht in Verbindung mit `~/ .vim` auf Linux/Mac, `C:\Documents und Settings\<<Ihr-Benutzername>\vimfiles` auf Windows und `C:\Users\<<Ihr-Benutzername>\vimfiles` auf Windows Vista. Sehen Sie hierzu `:help vimfiles` für nähere Einzelheiten.

Wir installieren jetzt Viki samt den benötigten [Plugins](#):

1. Laden Sie [multvals.vim](#)⁴⁰ herunter und speichern Sie es als `$vimfiles/plugin/multvals.vim`
2. Laden Sie [genutils.zip](#)⁴¹ herunter, und entpacken Sie diese Datei nach `$vimfiles`
3. Laden Sie [Viki.zip](#)⁴² herunter, und entpacken Sie auch diese Datei nach `$vimfiles` (Stellen Sie sicher, daß alle Verzeichnisse und Dateien unter dem Verzeichnisnamen 'Viki' direkt im `$vimfiles`-Verzeichnis liegen.)

Losgelegt

1. Öffnen Sie die grafische Version des Vim (`gvim`)
2. `:e test.txt`
3. `:set filetype=viki`
4. Geben Sie den folgenden Text ein: `[[http://deplate.sourceforge.net/Markup.html][Viki syntax]]`
5. `:w`

³⁹<http://www.lifehack.org/articles/lifehack/wikify-your-life-how-to-organize-everything.html>

⁴⁰http://www.vim.org/scripts/script.php?script_id=171

⁴¹http://www.vim.org/scripts/script.php?script_id=197

⁴²http://www.vim.org/scripts/script.php?script_id=861

6. Stellen Sie den **Cursor** auf den obigen Text und drücken Sie **ctrl+Enter** oder alternativ dazu, drücken Sie `colorred \vf`
7. Jetzt sollte sich ein **Browser** öffnen, mit der eben genannten Webseite.

Ebenso können Sie jeden Dateinamen (mit gültiger Pfadangabe) eingeben – ob es sich nun um eine .doc-Datei oder eine .pdf-Datei handelt. Mit **ctrl+Enter** öffnen Sie die Datei mit dem entsprechenden Programm.

Die Idee dabei ist, daß Sie einfache Textdateien benutzen können, um all Ihre Gedanken zu bündeln, und mit **ctrl+Enter** verzweigen Sie, wohin immer Sie wollen.

Beachten Sie, daß wir oben die eckigen Klammern paarweise eingeben mussten, um das Ziel des **Links** und die den Link beschreibenden Worte zu identifizieren. Dies ist die grundlegende Schreibweise der Seitenbeschreibungssprache, die wir als nächstes untersuchen werden.

11.3 Seitenbeschreibungssprache

Die Viki-Syntaxseite⁴³ (die Sie mit Ihrem **Browser** gerade geöffnet haben) erklärt, wie Sie den Text schreiben müssen, damit *Viki* Teile Ihres Satzbaus hervorheben kann, als auch wie der Link zwischen 'wikiseiten' zu bewerkstelligen ist, und sogar wie Sie *Viki*-spezifische Kommentare schreiben können.

Die grundlegenden Kenntnisse der Satzbauhervorhebung sind nützlich, weil Sie nicht ausschließlich auf Ihre geistige Vorstellungskraft angewiesen sind, sondern die Teile Ihrer Textdatei mit Ihren eigenen Augen betrachten können. Nehmen Sie z.B. die '*Zu erledigen-Liste', um daraus einen Dateikopf zu machen, und dann erstellen Sie mit Bindestrichen eine Liste:

** Zu erledigen-Liste*

- *Blogeintrag zu Brahmagiri trek zu Ende bringen*
- *'footer'-Programmierfehler der IONLAB Webseite beheben*
- *CD-Rohlinge kaufen*
- *Motorrad pflegen lassen*

11.4 KamelSchrift abschalten

Schreiben mit KamelSchrift kann in *Viki* einen **Link** erstellen, aber ich persönlich mag das nicht. Ich will, daß nur ausdrücklich bezeichnete **Links** wie `[[KamelSchrift]]` erlaubt sind, um Fälle zu vermeiden, wo ich einen echten Namen - mit KamelSchrift - benutzt habe, der aber kein **Link** sein soll (z.B. das Wort „JavaScript“). Um die KamelSchrift-Schreibweise zu unterbinden tippen Sie, `:edit $vimfiles/plugin/viki.vim`, und ändern die folgende Zeile (Zeile 120 gemäß dieses Schreibens):

```
if !exists("g:wikiNameTypes") | let g:wikiNameTypes = "csSeuix"
```

⁴³<http://deplate.sourceforge.net/Markup.html>

```
| endif "{{2
```

```
zu
```

```
if !exists("g:wikiNameTypes") | let g:wikiNameTypes = "sSeuix"
| endif "{{2
```

Haben Sie gesehen – das 'c' ist weg!

11.5 Was erledigt kriegen

Einer der Hauptgründe, dieses *'wiki'* für mich selbst zu erstellen, ist die Pflege eines 'Dinge geregelt kriegen'-Systems (→ „GTD“=Getting Things Done).

„GTD“ ist ein System, das von David Allen entwickelt wurde, damit Sie Ihren *'Kram'* in den Griff kriegen – was, von Ihren Karriereplänen bis zur täglichen Hausarbeit, alles einschließt. Eine gute Einführung in dieses System finden Sie auf www.bnet.com⁴⁴.

Aus David Allens Buch: „Machen Sie Ihren Kopf komplett frei. Treffen Sie Entscheidungen über notwendige Handlungen, wenn sie auftauchen und nicht wenn sie Ihnen um die Ohren fliegen. Organisieren Sie Erinnerungshilfen Ihrer Projekte, und die anstehenden Entscheidungen in dazu passenden Kategorien. Halten Sie Ihr System auf dem Laufenden, komplett, und ausreichend überprüft, um jederzeit Ihren spontanen Entscheidungen, was Sie gerade tun (oder nicht tun), zu vertrauen.“

Das GTD-System besteht im Wesentlichen darin, Ihre Daten in bestimmte Seiten/Verzeichnisse zu organisieren:

1. Sammeleimer
2. Projektliste
3. Nächste Handlungen
4. Kalender
5. Irgendwann/vielleicht
6. Nachschlagewerk
7. Warten auf

Ich habe ein *Viki* erstellt, das diesem System gerecht wird, mit folgender Methode:

1. Zuallererst erstellen Sie eine Startseite, die buchstäblich die Startseite zu Ihrem persönlichen Organisationssystem ist (hiermit einfach „Ihr Viki“ genannt).

⁴⁴http://www.bnet.com/2403-13074_23-52958.html

2. Danach erstellen Sie eine Liste für die Hauptabschnitte Ihres Wikis:

```
* Getting Things Done

1. [[Collect][In Basket]]
2. [[Project][Projects List]]
3. [[NextActions][Next Actions]]
4. [[Calendar]]
5. [[SomedayMaybe][Someday/Maybe]]
6. [[Reference][Reference Material]]
7. [[Waiting][Waiting For]]
```

3. Entsprechend, verschachteln Sie so tief Sie wollen, erstellen Sie z.B. ein [[Referenz.Karriere]], um Ihre Karrierepläne hinzuschmieren, und [[Projekt.StrengGeheim]], wo Sie Gedanken für Ihr nächstes Projekt sammeln, usw.
4. Jedesmal wenn Sie sich was notieren wollen, benutzen Sie die [[Sammeln]]-Seite, und dann tun Sie, organisieren Sie, überprüfen Sie, und letztendlich, unternehmen Sie Ihre nächsten physischen Handlungen.
5. Es braucht seine Zeit, bis Sie sich daran gewöhnt haben, dieses System zu benutzen, aber wenn Sie sich daran gewöhnt haben, kriegen Sie damit einen klaren Kopf, Vertrauen darin, alle Faktoren in Ihrem Leben berücksichtigt zu haben, und, das wichtigste, einen Sinn für die wichtigen Dinge in Ihrem Leben.

Beachten Sie, wie wir ein ganzes System mit einfachem Text verwalten!

11.6 Zusammenfassung

Wir haben gerade gesehen, wie Sie mit Vim ein persönliches Informationsverwaltungssystem⁴⁵ erstellen können. Schon faszinierend, dass wir dazu keinerlei komplizierte Programme für so ein System benötigen; einfache Textdateien und Vim erledigt den Rest.

Sehen Sie sich Abhijit Nadgouda's Artikel über die Benutzung des Vim für ein persönliches Wiki für einen alternativen Weg, dasselbe Problem zu lösen, durch die Benutzung programmimmanenter Vim-Funktionen an.

12 Scripting

(lat: *scribere* = Schreiben. 'Script' = Unix-Programm. Scriptdatei. Scriptsprache. Beim 'Scripting' handelt es sich also um das Verfassen einer Skriptdatei in einer Skriptsprache, und damit um das Erstellen (Schreibens) eines Programmes für unixoide Systeme.)

Auszüge aus: <http://de.wikipedia.org/wiki/Skript>

⁴⁵<http://ifacethoughts.net/2008/05/02/vim-as-a-personal-wiki/>

12.1 Einführung

Wenn Sie irgendein Programm individuell einrichten möchten, werden Sie mit höchster Wahrscheinlichkeit die verschiedenen Einstellungen in dem Programm ändern, um es Ihren Bedürfnissen und Ihrem Geschmack anzupassen. Was aber, wenn Ihnen das nicht ausreicht? Zum Beispiel, Bedingungen überprüfen wie **wenn GUI-Version, benutze dieses Farbschema, ansonsten benutze dieses Farbschema**? Hier kommt nun „Scripting“ ins Spiel. Scripting bedeutet eigentlich, eine Sprache zu benutzen, wo Sie Bedingungen festlegen, und Befehle zusammen in einer 'Skript'-Datei ablegen können.

Man kann sich beim 'scripting' in Vim von zwei Seiten annähern. Zum einen kann man die programmimmanente Vim-Skriptsprache verwenden, oder man benutzt eine vollwertige Programmiersprache wie z.B. Python oder Perl, die mittels Modulen Zugriff auf die Vim-Innereien haben (vorausgesetzt Vim wurde mit den entsprechenden Compileroptionen in Maschinsprache übersetzt).

Dieses Kapitel setzt einiges Programmierwissen voraus. Wenn Sie noch keine Programmiererfahrung haben, werden Sie trotzdem verstehen worum es geht, obwohl das vielleicht etwas knapp werden könnte. Wenn Sie programmieren lernen möchten, darf ich Sie auf mein anderes freies Buch „A Byte of Python“ hinweisen.

Für wiederverwendbare Funktionalität gibt es in Vim zwei Wege: Makros anwenden und Skripte schreiben.

12.2 Makros

Um Makros zu benutzen, können wir Befehlsabfolgen aufzeichnen, und sie dann in verschiedenen Zusammenhängen, ablaufen lassen.

Nehmen wir als Beispiel mal diesen Text hier:

```
tansen is the singer
daswant is the painter
todarmal is the financial wizard
abul fazl is the historian
birbal is the wazir
```

Da gibt es viel zu korrigieren.

1. Ändern Sie den ersten Buchstaben des Satzes in einen Großbuchstaben um.
2. Ändern Sie 'is' zu 'was'
3. Ändern Sie 'the' zu 'a'.
4. Beenden Sie den Satz mit in „Akbar's court“.

Eine Möglichkeit wäre eine Serie von Ersetzungsbefehlen zu verwenden, wie `:s/^\\w/\\u\\0/` aber dafür bräuchten wir vier Ersetzungsbefehle, und wenn der Ersetzungsbefehl Textstücke ändert, die wir unverändert lassen wollten, dann wird es haarig. Da sind Makros schon besser.

1. Setzen Sie Ihren **Cursor** auf den ersten Buchstaben, der ersten Zeile: **tansen is the singer**
2. Tippen Sie im Normalmodus **qa** um die Aufzeichnung des Makros mit der Bezeichnung **a** zu starten.
3. Tippen Sie **gU** um den ersten Buchstaben in einen Großbuchstaben umzuwandeln.
4. Geben Sie **w** ein, um zum nächsten Wort zu springen.
5. Tippen Sie **cw** um das Wort zu ändern.
6. Geben Sie **was** ein.
7. Drücken Sie **<ESC>**.
8. Geben Sie **w** ein, um zum nächsten Wort zu springen.
9. Tippen Sie **cw** um das Wort zu ändern.
10. Tippen Sie **a**.
11. Drücken Sie **<ESC>**.
12. Tippen Sie **A** um am Zeilenende Text einzugeben.
13. Geben Sie ein **in Akbar's court**.
14. Drücken Sie **<ESC>**.
15. Drücken Sie **q** um die Aufzeichnung zu beenden.

Sieht ganz nach einer langen Prozedur aus, aber manchmal ist das einfacher als dass man sich irgendwelche komplizierten Ersetzungsbefehle zusammenkocht!

Zum Schluß sollte die Zeile jedenfalls so aussehen:

```
Tansen was a singer in Akbar's court.
```

Super! Jetzt wollen wir dies auch auf die anderen Zeilen anwenden. Bewegen Sie sich einfach zum ersten Buchstaben der zweiten Zeile und drücken Sie **@a**. Na bitte – die Zeile sollte sich ändern zu:

```
Daswant was a painter in Akbar's court.
```

Das zeigt, dass Makros komplizierte Operationen aufzeichnen, und sehr leicht wiedergegeben werden können. Damit kann der Benutzer komplizierte Editierarbeiten an vielen Stellen erneut ablaufen lassen. Das ist eine Methode von Wiederbenutzung der Änderungen, die Sie am Text vornehmen können. Als nächstes werden wir formale Wege kennen lernen, Text zu manipulieren.

Hinweis Wenn Sie einfach nur die letzte Aktion wiederholen möchten, und nicht eine ganze Serie von Aktionen, müssen Sie dafür kein Makro benutzen. Drücken Sie einfach den **.** (Punkt).

12.3 Elementarskripting

Vim hat eine programmimmanente Skriptsprache, mit deren Hilfe Sie Ihre eigenen Skripte schreiben können, um Entscheidungen zu treffen, Dinge „tun“, und den Text zu manipulieren.

12.4 Aktionen

Wie ändern Sie das Farbschema, das Vim benutzt? Führen Sie dies aus:

```
:colorscheme desert
```

Hier benutze ich das 'Wüsten'-Farbschema, welches auch mein Lieblingsschema ist. Die anderen, zur Verfügung stehenden Schemata können Sie sich ansehen mit **:colorscheme**. Und mit der **<TAB>**-Taste können Sie alle Schemata durchlaufen.

Was ist denn, wenn Sie wissen möchten, wieviele Buchstaben die aktuelle Zeile enthält?

```
:echo strlen(getline("."))
```

Beachten Sie die Namen '**strlen**' und '**getline**'. Das sind Funktionen. Funktionen sind fertig erstellte Skriptstücke, denen man bereits einen Namen gegeben hat, damit wir sie immer wieder benutzen können. Die **getline**-Funktion z.B. holt sich eine Zeile, und mit dem **.** (Punkt) deuten wir schon an welche, nämlich die aktuelle Zeile. Das Ergebnis, das wir von der **getline**-Funktion kriegen, reichen wir an die **strlen**-Funktion weiter, welche die Buchstaben in dem Text zählt, und schließlich übergeben wir das Ergebnis der **strlen**-Funktion an den **:echo**-Befehl, der uns das Resultat einfach nur anzeigt. Vergegenwärtigen Sie sich den Informationsfluß dieses Befehls.

strlen(getline(".")) nennt man einen Ausdruck. Die Ergebnisse solcher Ausdrücke können wir speichern, indem wir Variablen benutzen. *Variablen* tun das, was der Name schon andeutet – Sie sind Namen, die auf Werte zeigen, und diese Werte können alles mögliche sein, d.h. es variiert. Die Länge können wir z.B. so speichern:

```
:let len = strlen(getline("."))  
:echo "We have" len "characters in this line."
```

Wenn Sie diese Zeile ausführen, angewandt auf die zweite Zeile oberhalb dieses Textes, kriegen Sie folgendes Ergebnis:

```
We have 46 characters in this line.
```

Beachten Sie, wie wir Variablen in anderen 'Ausdrücken' verwenden können. Die Möglichkeiten, was Sie mit Hilfe dieser Variablen, Ausdrücke und Befehle erreichen können, sind endlos.

Vim stellt über Präfixe viele Typen von Variablen zur Verfügung, wie **\$**, für Umgebungsvariablen, **&**, für Optionen, und **@**, für Register:

```
:echo $HOME
:echo &filetype
:echo @a
```

Sehen Sie sich auch `:help function-list` an, mit einer riesigen Liste von verfügbaren Funktionen.

Sie können auch Ihre eigenen Funktionen erstellen:

```
:function CurrentLineLength()
:   let len = strlen(getline("."))
:   return len
:endfunction
```

Jetzt setzen Sie Ihren `Cursor` mal auf irgendeine Zeile und führen den folgenden Befehl aus:

```
:echo CurrentLineLength()
```

Sie sollten eine Nummer ausgedruckt sehen.

Funktionsnamen müssen mit einem Großbuchstaben beginnen. Dies dient der Unterscheidung der programmimmanenten Funktionen, die mit einem Kleinbuchstaben beginnen, und benutzerdefinierten Funktionen, die mit einem Großbuchstaben beginnen.

Wenn Sie die Ausführung einer Funktion einfach nur aufrufen wollen, ohne dass die Inhalte angezeigt werden, können Sie folgendes tun `:call CurrentLineLength()`.

12.5 Entscheidungen

Angenommen Sie möchten ein anderes Farbschema anzeigen, abhängig davon, ob Vim im Terminal oder in der grafischen Variante läuft, d.h. Sie brauchen das Skript, um Entscheidungen zu fällen. Dann nehmen Sie dies hier:

```
:if has("gui_running")
:   colorscheme desert
:else
:   colorscheme darkblue
:endif
```

Das funktioniert so:

- `has()` ist eine Funktion, die angewandt wird, um zu entscheiden, ob eine bestimmte Fähigkeit des Vim unterstützt wird, der auf dem aktuellen Computer gerade installiert ist. Sehen Sie hierzu `:help feature-list` um die verfügbaren Programmfähigkeiten kennenzulernen.
- Die `if`-Anweisung überprüft die gegebene Bedingung. Wenn die Bedingung erfüllt ist, setzen wir bestimmte Dinge in gang. „`Else`“, heißt, wir nehmen die Alternative.

- Denken Sie daran, dass zu einer **if**-Anweisung auch ein entsprechendes **endif** gehört.
- Und es gibt natürlich auch noch **elseif**, wenn Sie mehrere Bedingungen und Aktionen verketteten möchten.

Die Schleifenausdrücke '**for**' und '**while**' kennt Vim auch:

```
:let i = 0
:while i < 5
:    echo i
:    let i += 1
:endwhile
```

Das kommt dabei heraus:

```
0
1
2
3
4
```

Mit den programmimmanenten Funktionen des Vim, kann dasselbe auch so geschrieben werden:

```
:for i in range(5)
:    echo i
:endfor
```

- `range()` ist eine programmimmanente Funktion, mit der man eine Reihe von Zahlen generiert. Sehen Sie sich hierzu auch **:help range()** an.
- Die **continue** und **break** Ausdrücke stehen auch zur Verfügung.

12.6 Datenstrukturen

Das Erstellen von Skripten mit Vim unterstützt auch Listen und Wörterbücher. Bei Verwendung jener, können Sie komplizierte Programme und Datenstrukturen aufbauen.

```
:let fruits = ['apple', 'mango', 'coconut']

:echo fruits[0]
" apple

:echo len(fruits)
" 3
```

```

:call remove(fruits, 0)
:echo fruits
" ['mango', 'coconut']

:call sort(fruits)
:echo fruits
" ['coconut', 'mango']

:for fruit in fruits
:  echo "I like" fruit
:endfor
" I like coconut
" I like mango

```

Es stehen viele Funktionen zur Verfügung schauen Sie sich die Abschnitte '**List manipulation**' und '**Dictionary manipulation**' in **:help function-list** an.

12.7 Ein Vim-Skript schreiben

Wir schreiben jetzt ein Skript, das man in Vim laden kann, und deren Funktionalität wir dann aufrufen können, wann immer wir es wollen. Das ist was anderes, als das Skript innerhalb von Vim zu schreiben, und unmittelbar auszuführen, wie wir es bis jetzt getan haben.

Gehen wir mal ein einfaches Problem an – wie wär's damit: Der erste Buchstabe eines jeden Wortes, eines ausgewählten Zeilenbereiches, soll in einen Großbuchstaben umgewandelt werden. Der Anwendungsfall ist einfach Wenn ich Titelköpfe in einem Dokument schreibe, sieht das mit Großbuchstaben besser aus, aber ich bin zu faul, um das selbst zu tun. Ich kann den Text also in Kleinbuchstaben eintippen, und dann einfach die Funktion aufrufen, um ihn mit Großbuchstaben zu versehen.

Wir fangen mit einem einfachen Vorlageskript an. Speichern Sie das folgende Skript als **capitalize.vim**:

```

" Vim global plugin for capitalizing first letter of each word
"      in the current line.
" Last Change: 2008-11-21 Fri 08:23 AMIST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php

if exists("loaded_capitalize")
    finish
endif
let loaded_capitalize = 1

" TODO : The real functionality goes in here.

```

Funktionsweise:

- Die erste Zeile der Datei sollte ein Kommentar sein, der erklärt, worum es überhaupt geht.
- 2-3 Standardköpfe werden erwähnt, die die Datei betreffen, wie z.B. 'Last Changed:', das erklärt wie alt das Skript ist. Der 'Maintainer:'-Hinweis, damit Endanwender den Skriptautor kontaktieren können, sollten irgendwelche Probleme aufgetaucht sein, oder wenn man sich auch einfach nur bedanken möchte.
- Die 'Lizenz'-Kopfzeile muss nicht unbedingt angegeben werden, ist aber stark empfohlen. Ein von Ihnen geschriebenes Vim-Skript oder [Plugin](#), könnte auch für viele andere von Nutzen sein, weshalb Sie für das Skript eine Lizenz festlegen können. In der Folge können andere Leute Ihre Arbeit verbessern, und davon profitieren letztendlich wieder Sie selbst.
- Ein Skript kann mehrere Male geladen sein. Wenn Sie z.B. zwei verschiedene Dateien in einer VimSitzung öffnen, und beide sind .html-Dateien, dann öffnet Vim das HTML-Syntaxhervorhebungsskript für beide Dateien. Um zu vermeiden, das gleiche Skript zweimal auszuführen, und die Dinge umzubennenen, bauen wir eine Sicherung ein, indem wir überprüfen, ob es den Namen `'loaded_capitalize'` schon gibt, und schließen es, wenn es bereits geladen wurde.

Jetzt machen wir weiter, indem wir die aktuelle Funktion schreiben.

Wir können eine Funktion festlegen, um die Umformung – Umwandlung des ersten Buchstaben eines jeden Wortes in einen Großbuchstaben zu erreichen, und diese Funktion nennen wir `Capitalize()`. Da sich die Funktion auf einen Bereich bezieht, können wir festlegen, dass sie in einem Bereich von Zeilen arbeitet.

```
" Vim global plugin for capitalizing first letter of each word
"   in the current line
" Last Change: 2008-11-21 Fri 08:23 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php

" Make sure we run only once
if exists("loaded_capitalize")
    finish
endif
let loaded_capitalize = 1

" Capitalize the first letter of each word
function Capitalize() range
    for line_number in range(a:firstline, a:lastline)
        let line_content = getline(line_number)
```



```

" Luckily, the Vim manual had the solution already!
" Refer ":help s%" and see 'Examples' section
    let line_content = substitute(line_content, "\\w\\+", "\\u\\0", "g")
    call setline(line_number, line_content)
endfor
endfunction

```

Funktionsweise:

- `a:firstline` und `a:lastline` stellen die Argumente der Funktion dar, die auf den Anfang, bzw. das Ende des Zeilenbereiches verweisen.
- Um jede Zeile im Bereich abzuarbeiten, benutzen wir eine For-Schleife (das machen wir mit `getline()`).
- Wir benutzen die `substitute()`-Funktion, um einen regulären Ausdruck Suchen-und-Ersetzen an der Zeichenkette durchzuführen. Hier schreiben wir die Funktion so, dass sie nach Worten sucht (das `'\\w\\+'` zeigt dies an), was ein Wort bedeutet (d.h. ein zusammenhängender Satz von Zeichen, die Teil eines Wortes sind). Wenn solche Worte gefunden wurden, werden sie mit `\\u\\0` umgewandelt - das `\\u` heißt: wandle den ersten Buchstaben, der auf diese Sequenz folgt in einen Großbuchstaben um. Das `\\0` zeigt auf das Ergebnis, das von `substitute()` gefunden wurde, die mit den Worten zusammenhängen. Damit ändern wir den ersten Buchstaben jedes Wortes in einen Großbuchstaben um.
- Wir rufen die `setline()`-Funktion auf, um die Zeile in Vim mit der modifizierten Zeichenkette zu ersetzen.

Um diesen Befehl auszuführen:

1. Öffnen Sie Vim und geben Sie irgendeinen Text ein, wie `'das ist ein Test'`.
2. Führen Sie aus `:source capitalize.vim` – das liest die Datei ein, so als ob die Befehle innerhalb Vims ausgeführt werden, wie wir es schon zuvor getan haben.
3. Führen Sie aus `:call Captialize()`.
4. In der Zeile sollte nun `'Das Ist Ein Test'` stehen.

Jedesmal `:call Captialize()` auszuführen, ist jedoch ermüdend, und deshalb weisen wir dem Aufruf ein Tastaturkürzel zu:

```

" Vim global plugin for capitalizing first letter of each word
" in the current line
" Last Change: 2008-11-21 Fri 08:23 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php

```

```

" Make sure we run only once
if exists("loaded_capitalize")
    finish
endif
let loaded_capitalize = 1

" Refer ':help using-<Plug>'
if !hasmapto('<Plug>Capitalize')
    map <unique> <Leader>c <Plug>Capitalize
endif
noremap <unique> <script> <Plug>Capitalize <SID>Capitalize
noremap <SID>Capitalize :call <SID>Capitalize(<CR>

" Capitalize the first letter of each word
function s:Capitalize() range
    for line_number in range(a:firstline, a:lastline)
        let line_content = getline(line_number)
" Luckily, the Vim manual had the solution already!
" Refer ":help s%" and see 'Examples' section
        let line_content = substitute(line_content, "\\w\\+", "\\u\\0", "g")
        call setline(line_number, line_content)
    endfor
endfunction

```

- Die Funktion 'Capitalize' haben wir einfach in 's:Capitalize' geändert – dies um anzuzeigen, dass die Funktion innerhalb des verfassten Skripts lokal verwandt wird, und sie global nicht zur Verfügung stehen sollte, weil wir eine Beeinflussung anderer Skripten damit vermeiden wollen.
- Mit dem **map**-Befehl definieren wir das Tastaturkürzel.
- Die **<Leader>**-Taste ist üblicherweise der linksseitige Schrägstrich (→backslash).
- Wir bilden nun **<Leader>c** (also den **Leader**-Schlüssel, gefolgt von der '**c**'-Taste) auf eine Funktion ab.
- Wir benutzen **<Plug>Capitalize** um auf die **Capitalize()**-Funktion zu zeigen, die in einem **PlugIn**, also in unserem eigenen Skript beschrieben ist.
- Jedes Skript besitzt eine ID (Identität), repräsentiert durch **<SID>**. Wir können jetzt also den Befehl **<SID>Capitalize** auf einen Aufruf der lokalen Funktion **Capitalize()** abbilden.

Nun wiederholen Sie mal diese zuvor erwähnten Schritte, um das Skript zu testen. Doch jetzt können Sie **\\c** ausführen anstatt **:call Capitalize()** einzugeben, um Zeilen in Großbuchstaben umzuwandeln.

Diese letzten Schritte waren jetzt vielleicht ein bisschen kompliziert, aber das war nur um zu zeigen, dass es eine Fülle von Möglichkeiten gibt, Vim-Skripte zu erstellen, mit denen man komplexe Dinge erledigen kann.

Wenn irgendetwas nicht klappt in Ihren Skripten, können Sie sich mit `v:errmsg` die letzte Fehlermeldung anzeigen lassen, die Ihnen vielleicht eine Idee davon gibt, was schief gelaufen ist.

Beachten Sie, dass Sie mit `:help` Hilfe zu allem finden können, das wir oben schon diskutiert haben, von `:help \\w` zu `:help setline()`.

12.8 Das Verwenden eigenständiger Programmiersprachen

Viele Leute haben keine Lust, Zeit damit zu verbringen, Vims eigene Skriptsprache zu erlernen, und ziehen es vor eine Programmiersprache zu verwenden, die sie bereits kennen, um [Plugins](#) für Vim in dieser Sprache zu schreiben. Das ist möglich, weil Vim das Schreiben von [Plugins](#) in Python, Perl, Ruby und vielen anderen Sprachen unterstützt.

In diesem Kapitel schauen wir auf ein einfaches, in *Python* geschriebenes, Programmiererweiterungsmodul an, aber wir könnten genauso gut jede andere unterstützte Programmiersprache nehmen.

Wie schon erwähnt, wenn Sie Bock haben Python zu lernen, interessiert Sie möglicherweise mein anderes freies Buch „A Byte Of Python“.

Zuerst müssen wir mal checken, ob Python auf Ihrem System schon eingerichtet ist.

```
:echo has("python")
```

Wenn Sie hier eine 1 zurückkriegen, kann's losgehen, ansonsten sollten Sie *Python* auf Ihrem Rechner installieren, und es dann nochmal versuchen.

Angenommen, Sie stellen einen [Blog](#) ins Internet. Ein Blogger möchte in der Regel, dass soviel Leute wie nur möglich, seinen/ihren [Blog](#) lesen. Eine Methode, solche [Blogs](#) im Internet zu finden, ist, eine Suchmaschine abzufragen. Wenn Sie also über ein Thema schreiben, (sagen wir mal, 'C V Raman', der berühmte indische Physiker, der einen Nobelpreis für seine Arbeit über die 'Streuung von Licht' gewonnen hat), dann wollen Sie vielleicht wichtige Sätze verwenden, damit mehr Leute Ihren [Blog](#) finden, wenn sie nach diesem Thema suchen. Wenn jemand z.B. nach 'c v raman' sucht, könnte ihnen auch der 'raman-effekt' einfallen, deshalb möchten Sie dies in Ihrem [Blog](#)-Eintrag erwähnen.

Wie finden wir nun solche, im Zusammenhang stehenden, Sätze? Wie sich herausstellt ist die Lösung recht einfach, dank [Yahoo! Search](#).

Zuerst wollen wir mal herausfinden, wie wir *Python* dazu nutzen können um auf den aktuellen Text zuzugreifen, um die im Zusammenhang stehenden Sätze zu generieren.

```
" Vim plugin for looking up popular search queries related
"      to the current line
" Last Updated: 2008-11-21 Fri 08:36 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php
```

```

" Make sure we run only once
if exists("loaded_related")
    finish
endif
let loaded_related = 1

" Look up Yahoo Search and show results to the user
function Related()
python <<EOF

import vim

print 'Length of the current line is', len(vim.current.line)

EOF
endfunction

```

Die grundsätzliche Herangehensweise **Plugins** in mit Schnittstellen ausgestattete Programmiersprachen zu schreiben, ist die gleiche wie unter programmimmanenten Skriptsprachen.

Der Hauptunterschied besteht darin, dass wir den in *Python* programmierten Code, dem *Python*-Interpreter übergeben müssen. Dies wird mit einer EOF-Anweisung, wie oben gezeigt, erreicht. Der ganze Text des `python <<EOF`-Befehls, bis hin zum `EOF`, wird dem *Python*-Interpreter übergeben.

Sie können dieses Programm ausprobieren, indem Sie Vim wieder separat öffnen, und ausführen `:source related.vim`, und dann `:call Related()`. Dies sollte sowas wie **Length of the current line is 54** anzeigen.

Jetzt steigen wir aber hinab in die Funktionalitätstiefen des Programmes. **Yahoo! Search** hat was, das nennt sich VergleichbareVorschlag-Suche (RelatedSuggestion query⁴⁶), auf die wir durch einen Webservice Zugriff haben. Auf diesen Webservice kann man zugreifen indem man eine Python-API* nutzt, die von **Yahoo! Search** `pYSearch`⁴⁷ zur Verfügung gestellt wird: *(API=Application Programming Interface Programmierschnittstelle)

```

" Vim plugin for looking up popular search queries related
" to the current line
" Last Updated: 2008-11-21 Fri 08:36 AM IST
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php

" Make sure we run only once

```

⁴⁶<http://developer.yahoo.com/search/web/V1/relatedSuggestion.html>

⁴⁷<http://pysearch.sourceforge.net>

```

if exists("loaded_related")
    finish
endif
let loaded_related = 1

" Look up Yahoo Search and show results to the user
function Related()
python <<EOF

import vim
from yahoo.search.web import RelatedSuggestion

search = RelatedSuggestion(app_id='vimsearch', query=vim.current.line)
results = search.parse_results()

msg = 'Related popular searches are:\n'
i = 1
for result in results:
    msg += '%d. %s\n' % (i, result)
    i += 1
print msg

EOF
endfunction

```

Beachten Sie, dass wir die aktuelle Zeile in Vim, als den aktuellen Text verwenden, der uns interessiert. Dies können Sie auf jeden beliebigen Text anwenden, wie z.B. das aktuelle Wort, usw.

Wir benutzen die `yahoo.search.web.RelatedSuggestion` Klasse um Yahoo! Search nach Sätzen abzufragen, die mit der Abfrage in Zusammenhang stehen, die wir festlegen. Die Ergebnisse kriegen wir zurück, indem wir `parse_results()` auf das resultierende Objekt anwenden. Die Ergebnisse legen wir dann zurück, und zeigen sie dem Benutzer an.

1. Führen Sie aus `:source related.vim`
2. Tippen Sie den Text `c v raman`.
3. Führen Sie `:call Related()` aus
4. Was dabei rauskommt, sollte ungefähr so aussehen:

```

Related popular searches are:
1. raman effect
2. c v raman india
3. raman research institute
4. chandrasekhara venkata raman

```

12.9 Zusammenfassung

Wir haben einen Ausflug in die Welt des 'Skripte schreiben' unternommen, und zwar sowohl mit Vim's eigener Skriptsprache, als auch mit eigenständigen Skripting- und Programmiersprachen. Das ist wichtig, weil die Funktionalität des Vim unbegrenzt erweitert werden kann.

Sehen Sie hierzu auch `:help eval`, `:help python-commands`, `:help perl-using` und `:help ruby-commands` für Einzelheiten.

13 Plugins

13.1 Einführung

Wie wir im vorhergehenden Kapitel gesehen haben, helfen uns Skripte, die bestehende Funktionalität des Vim zu erweitern, um mehr Aufgaben zu erledigen. Diese Skripte, die die Funktionalität erweitern, bzw. hinzufügen, nennen wir „*Plugins*“ (→to plug in = einstöpseln).

Es gibt verschiedene Arten von *Plugins*, die man schreiben kann:

- vimrc
- globales *Plugin*
- dateispezifisches *Plugin*
- Syntaxhervorhebungs-*Plugin*
- Compiler-*Plugin*

Sie können nicht nur Ihre eigenen *Plugins* schreiben, sondern auch die von anderen geschriebenen *Plugins*, herunterladen und benutzen. ⁴⁸

13.2 Individualisierung des Vim mit der vimrc

Wenn ich eine neue Linux-Distribution installiere, oder Windows neu aufsetze, ist das erste, was ich mache, nachdem ich Vim installiert habe, mir meine aktuellste *vimrc* aus meiner Sicherungskopie zu krallen, und Vim erst dann benutze. Warum das wichtig ist? Weil die *vimrc*-Datei viele Einstellungen enthält, die ich mag, und das macht den Vim für mich erst brauchbar, und (in der Bedienung) angenehm.

Es gibt zwei Dateien, die Sie erstellen können, um Vim ganz nach Ihrem Geschmack einzurichten:

1. `vimrc` - für allgemeine Einstellungen
2. `gvimrc` - für Einstellungen des grafischen Vim

Die sind gespeichert als:

⁴⁸<http://www.vim.org/scripts/index.php>

- `%HOME%/_vimrc` und `%HOME%/_gvimrc` bei Windows, und
- `$HOME/.vimrc` und `$HOME/.gvimrc` auf Unixoiden Systemen (GNU/ Linux/ BSD/ MacOSX)

Schauen Sie sich `:help vimrc` an, für den exakten Speicherort Ihres Systems.

Diese vimrc und gvimrc-Dateien können jegliche Vim-Befehle enthalten. Folgendes soll hierbei gelten: einfache Einstellungen nehmen wir in der vimrc vor, kompliziertere Dinge lesen wir mit *Plugins* ein.

Hier ist z.B. ein Auszug aus meiner *vimrc*:

```
" My Vimrc file
" Maintainer: www.swaroopch.com/contact/
" Reference: Initially based on http://dev.gentoo.org/~ciaranm/docs/vim-guide/
" License: www.opensource.org/licenses/bsd-license.php

" Use Vim settings, rather than Vi settings (much better!).
" This must be first, because it changes other options as a side effect.
set nocompatible

" Enable syntax highlighting.
syntax on

" Automatically indent when adding a curly bracket, etc.
set smartindent

" Tabs should be converted to a group of 4 spaces.
" This is the official Python convention
" (http://www.python.org/dev/peps/pep-0008/)
" I didn't find a good reason to not use it everywhere.
set shiftwidth=4
set tabstop=4
set expandtab
set smarttab

" Minimal number of screen lines to keep above and below the cursor.
set scrolloff=999

" Use UTF-8.
set encoding=utf-8

" Set color scheme that I like.
if has("gui_running")
```

```

    colorscheme desert
else
    colorscheme darkblue
endif

" Status line
set laststatus=2
set statusline=
" buffer number
set statusline+=%-33n\
" filename
set statusline+=%f\
" status flags
set statusline+=%t/%m/%w
" file type
set statusline+=\|{strlen(&ft)?&ft:'none'}]
" right align remainder
set statusline+=%r=
" character value
set statusline+=%c%b
" line, character
set statusline+=%-14%l,%c%l%
" file position
set statusline+=%P

" Show line number, cursor position.
set ruler

" Display incomplete commands.
set showcmd

" To insert timestamp, press F3.
nmap <F3> a<CR>=strftime("%Y-%m-%d %a %l:%M%p")<CR><Esc>
imap <F3> <CR>=strftime("%Y-%m-%d %a %l:%M%p")<CR>

" To save, press ctrl-s.
nmap <c-s> :w<CR>
imap <c-s> <Esc>w<CR>a

" Search as you type.
set incsearch

" Ignore case when searching.
set ignorecase

```



```

" Show autocomplete menus.
set wildmenu

" Show editing mode
set showmode

" Error bells are displayed visually.
set visualbell

```

Beachten Sie, dass diesen Befehlen der vorangestellte Doppelpunkt fehlt. Der Doppelpunkt ist nicht zwingend bei Dateiskripten, weil hier angenommen wird, dass es sich um Befehle des Normalmodus handelt.

Wenn Sie den eingehenden Gebrauch jeder einzelnen oben erwähnten Einstellung erlernen wollen, verweise ich auf **:help**.

Noch einen Teil aus meiner *gvimrc*:

```

" Size of GVim window
set lines=35 columns=99

" Don't display the menu or toolbar. Just the screen.
set guioptions=m
set guioptions=T

" Font. Very important.
if has('win32') || has('win64')
" set guifont=Monaco:h16
" http://jeffmilner.com/index.php/2005/07/30/windows-vista-fonts-now-available/
set guifont=Consolas:h12:cANSI
elseif has('unix')
let &guifont="Monospace 10"
endif

```

Da draussen gibt es jede Menge an Beispielen von *vimrc*'s⁴⁹ auf die Sie unbedingt mal einen Blick werfen sollten, um die zahlreichen Arten von individuellen Einstellungsmöglichkeiten kennenzulernen, und sich dann genau diejenigen herauszupicken, die Ihnen am besten gefallen, um Sie in Ihre eigene *vimrc* einzubauen.

Ein paar gute, die ich vormals gefunden habe, sind:

- **vi-improved.org**'s⁵⁰ *vimrc*
- **Amir Salihefendic**'s⁵¹ *vimrc*

⁴⁹<http://dotfiles.org/.vimrc>

⁵⁰<http://www.vi-improved.org/vimrc.php>

⁵¹<http://amix.dk/vim/vimrc.html>

Es ist ein bekanntes Faktum, dass durch die *vimrc* eines Benutzers darauf geschlossen werden kann, wie lange derjenige/diejenige Vim schon benutzt.

13.3 Globales Plugin

Globale *Plugins* werden benutzt, um allgemeine Funktionen zur Verfügung zu stellen. Globale *Plugins* liegen an zwei Orten:

1. `$VIMRUNTIME/plugin/` für Standardplugins, die Vim schon bei seiner Installation einrichtet.
2. Um Ihre eigenen *Plugins* bzw. *Plugins*, die Sie von irgendwo heruntergeladen haben, zu installieren, nehmen Sie Ihr eigenes *Plugin*-Verzeichnis:
 - `$HOME/.vim/plugin/` auf Unixoiden Systemen (GNU/ Linux/ BSD/ MacOSX)
 - `%HOME%/vimfiles/plugin/` für Windows
 - Sehen sich sich `:help runtimepath` für Einzelheiten zu Ihren *Plugin*- Verzeichnissen an.

Jetzt wollen wir so ein *Plugin* einmal anwenden.

Ein nützliches *Plugin* ist `highlight_current_line.vim`⁵² von Ansuman Mohanty, das genau das tut, was der Name schon suggeriert. Laden Sie sich neuste Version der Datei `highlight_current_line.vim` herunter, und legen Sie sie in Ihr *Plugin*-Verzeichnis (wie oben erwähnt). Starten Sie Vim erneut, und öffnen Sie irgendeine Datei. Achten Sie darauf, wie die aktuelle Zeile hervorgehoben ist, im Vergleich zu den anderen Zeilen dieser Datei.

Aber was ist, wenn Ihnen das gar nicht gefällt? Na, dann löschen Sie das Plugin einfach wieder, und starten den Vim nochmal.

Ganz genauso können wir unser eigenes `related.vim` oder `capitalize.vim` aus dem 'Skripte'-Kapitel in unser *Plugin*-Verzeichnis installieren, und dies erspart uns den Ärger, jedesmal `:source` eingeben zu müssen. Letztendlich sollte jede Art von *Plugin*, das Sie schreiben, irgendwo in Ihrem `.vim/vimfiles/plugin`-Ordner landen.

13.4 Dateityppplugin

Dateityppplugins wurden geschrieben, damit man an bestimmten Arten von Dateien arbeiten kann. Programme z.B., die in der Programmiersprache C geschrieben wurden, können ihren eigenen Einrückstil, Faltungen, Syntaxhervorhebung und sogar ihre eigene Fehleranzeige haben. Solche *Plugins* sind nicht für allgemeine Zwecke, sie funktionieren für diese bestimmten Dateitypen.

⁵²http://www.vim.org/scripts/script.php?script_id=1652

13.5 Ein Filetype-Plugin benutzen

Versuchen wir mal ein Dateitypplugin für XML. XML ist eine beschreibende Sprache, die Marken benutzt, um die Dokumentstruktur selbst zu beschreiben. Wenn Sie z.B. einen Text wie diesen hier haben:

Iron Gods

Ashok Banker's next book immediately following the *Ramayana* is said to be a novel tentatively titled "Iron Gods" scheduled to be published in 2007. A contemporary novel, it is an epic hard science fiction story about a war between the gods of different faiths. Weary of the constant infighting between religious sects and their deities, God (aka Allah, Yahweh, brahman, or whatever one chooses to call the Supreme Deity) wishes to destroy creation altogether.

A representation of prophets and holy warriors led by Ganesa, the elephant-headed Hindu deity, randomly picks a sample of mortals, five of whom are the main protagonists of the book—an American Catholic, an Indian Hindu, a Pakistani Muslim, a Japanese Buddhist, and a Japanese Shinto follower. The mortal sampling, called a 'Palimpsest' is ferried aboard a vast Dyson's Sphere artifact termed The Jewel, which is built around the sun itself, contains retransplanted cities and landscapes brought from multiple parallel Earths and is the size of 12,000 Earths. It is also a spaceship travelling to the end of creation, where the Palimpsest is to present itself before God to plead clemency for all creation.

Meanwhile, it is upto the five protagonists, aided by Ganesa and a few concerned individuals, including Lucifer Morningstar, Ali Abu Tarab, King David and his son Solomon, and others, to bring about peace among the myriad warring faiths. The question is whether or not they can do so before the audience with God, and if they can do so peacefully—for pressure is mounting to wage one final War of Wars to end all war itself.

(Excerpt taken from

http://en.wikipedia.org/w/index.php?title=Ashok_Banker&oldid=86219280
under the GNU Free Documentation License)

In XML (genauer 'Docbook XML'), sieht das ganze dann so aus:

```
<DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
```

```

"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">
<article>

  <articleinfo>
    <author<firstname>Wikipedia Contributors</firstname></author>
    <title>Iron Gods</title>
  </articleinfo>

  <para>

    Ashok Banker's next book immediately following the Ramayana is
    said to be a novel tentatively titled "Iron Gods" scheduled to
    be published in 2007. A contemporary novel, it is an epic hard
    science fiction story about a war between the gods of
    different faiths. Weary of the constant infighting between
    religious sects and their deities, God (aka Allah, Yahweh,
    brahman, or whatever one chooses to call the Supreme Deity)
    wishes to destroy creation altogether.

  </para>

  <para>

    A representation of prophets and holy warriors led by Ganesa,
    the elephant-headed Hindu deity, randomly picks a sample of
    mortals, five of whom are the main protagonists of the
    book—an American Catholic, an Indian Hindu, a Pakistani
    Muslim, a Japanese Buddhist, and a Japanese Shinto follower.
    The mortal sampling, called a 'Palimpsest' is ferried aboard a
    vast Dyson's Sphere artifact termed The Jewel, which is built
    around the sun itself, contains retransplanted cities and
    landscapes brought from multiple parallel Earths and is the
    size of 12,000 Earths. It is also a spaceship travelling to
    the end of creation, where the Palimpsest is to present itself
    before God to plead clemency for all creation.

  </para>

  <para>

    Meanwhile, it is upto the five protagonists, aided by Ganesa
    and a few concerned individuals, including Lucifer
    Morningstar, Ali Abu Tarab, King David and his son Solomon,
    and others, to bring about peace among the myriad warring

```

faiths. The question is whether or not they can do so before the audience with God, and if they can do so peacefully—for pressure is mounting to wage one final War of Was to end all war itself.

</para>

<sidebar>

<para>

(Excerpt taken from
http://en.wikipedia.org/w/index.php?title=Ashok_Banerjee&oldid=86219280
 under the GNU Free Documentation License)

</para>

</sidebar>

</article>

Beachten Sie, dass die Struktur des Dokumentes in der XML-Version noch ausführlicher ist. Dies macht es für Konvertierungsprogramme einfacher XML in andere Formate zu übertragen, einschließlich PDF und Druckversionen. Der Nachteil wiederum ist, dass das Erstellen in XML schwieriger für denjenigen ist, der die Seite schreibt.

Wollen mal sehen inwiefern **ftplugins** einem Vim-Benutzer beim Verfassen von XML helfen kann.

1. Laden Sie sich zuerstmal das **xmledit ftplugin**⁵³ runter, und werfen Sie es in Ihr **~/.vim/ftplugin/** Verzeichnis.
2. Fügen Sie Ihrer **~/.vimrc** noch folgende Zeile hinzu:

```
:autocmd BufNewFile,BufRead *.xml source ~/.vim/ftplugin/xml.vim
```

(Stellen Sie sicher, das richtige Verzeichnis anzugeben, gemäß Ihres Betriebssystems) Dies ruft das **xmledit ftplugin** jedesmal auf, wenn Sie eine Datei mit der Erweiterung **.xml** öffnen.

3. Öffnen Sie Vim und bearbeiten Sie eine Datei mit Namen **test.xml**.
4. Tippen Sie **<article**.
5. Jetzt geben Sie das schließende **>** ein, und sehen wie das **xmledit ftplugin** die schließende Marke automatisch für Sie einfügt. Ihr Dokument sollte nun so aussehen:

⁵³http://www.vim.org/scripts/script.php?script_id=301

```
<article>/article>
```

6. Tippen Sie nun ein weiteres mal > und sehen Sie wie die Marke expandiert, so dass Sie weitere Marken eingeben können. Das Dokument sollte nun so aussehen:

```
<article>
</article>
```

7. Sie werden bemerkt haben, dass auch der **Cursor** eingerückt ist, so dass Sie das Dokument ordentlich strukturiert schreiben können, was der Struktur des Dokuments entspricht.
8. Wiederholen Sie diesen Prozeß, bis Sie das ganze Dokument geschrieben haben.

Beachten Sie wie ein spezielles ftplugin für XML, es für Sie viel leichter macht XML-Dokumente zu schreiben. Dafür wurden ftplugins in der Regel gemacht.

13.6 Ein Dateityp-Plugin schreiben

Wir wollen einmal versuchen ein eigenes ftplugin zu schreiben.

In unserem vorherigen Beispiel, bei dem wir das **xmledit ftplugin** benutzt, und im XML-Format geschrieben haben, haben wir gesehen, dass wir ein paar standard Kopfinformationen, ganz oben in jede *DocBook-XML-Datei* reinschreiben müssen (das ist genau das Format, das wir benutzt haben). Warum automatisieren wir das nicht in Vim, mit Hilfe des ftplugin?

Unser xml ftplugin muss im Grunde lediglich die folgenden Informationen an den Anfang einer jeden 'neuen' XML-Datei setzen:

```
<?xml version="1.0"?>
<DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
    "http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">
```

Jetzt wollen wir dafür sorgen, dass unser neues ftplugin, das wir, sagen wir mal **'xmlheader.vim'** nennen, nur dann funktioniert, beim Eintreten eines **'BufNewFile'**-Ereignisses. Fügen Sie also folgendes in Ihre **~/.vimrc**:

```
:autocmd BufNewFile *.xml source ~/.vim/ftplugin/xmlheader.vim
```

So! Alles was wir jetzt noch in **'xmlheader.vim'** tun müssen, ist das Setzen der ersten und zweiten Zeile der Datei:

```

" Vim plugin to add XML header information to a new XML file
call setline(1, '<?xml version="1.0"?>')
call setline(2, '<DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http
: //www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">')

```

Ok – starten Sie den Vim erneut, und stellen Sie sicher, dass es `'test.xml'` nicht schon gibt, und führen Sie `:e test.xml` aus. Sie sehen, dass der Dateikopf bereits eingefügt ist!

13.7 Syntaxschema

Im vorherigen Abschnitt haben wir gerade eben eine DocBook-XML-Datei erstellt. Wäre schon hilfreich gewesen, wenn es ein wenig Farbcodierung für die XML-Datei gegeben hätte, für (der Syntax entsprechende) gültige DocBook-Marken, um sicherzustellen, dass wir es korrekt eingegeben haben. Es stellt sich heraus, dass dies möglich ist, wenn wir lediglich `:set filetype=docbkxml` ausführen. Vim benutzt jetzt die in `$VIMRUNTIME/syntax/docbkxml.vim` hinterlegte Syntaxdatei.

Die Syntaxdatei bestimmt, wie die Teile der Datei zueinander stehen. Zum Beispiel, die Syntaxdatei für XML bestimmt was Marken (*Tags*) sind, und welche Farben den Marken-Namen zugewiesen werden sollen, etc.

13.8 Ein Syntaxschema benutzen

Lassen Sie uns so eine Syntaxdatei einmal hautnah bei der Arbeit erleben. Laden Sie sich `mkd.vim`⁵⁴ herunter - das ist ein Syntax-Skript für die *Markdown-Syntax*⁵⁵. *Markdown* ist im Grunde ein Format, das es Ihnen ermöglicht einfachen Text einzugeben, der später in HTML umgewandelt wird.

1. Öffnen Sie eine neue Datei mit Vim, die Sie `'test.markdown.txt'` nennen.
2. Führen Sie `:set syntax=mkd` aus.
3. Tippen Sie folgenden Text in die Datei:

⁵⁴http://www.vim.org/scripts/script.php?script_id=1242

⁵⁵<http://daringfireball.net/projects/markdown/>

```
# Bengaluru

The name Bangalore is an anglicised version of the city's name in
the Kannada language, Bengaluru.

> A popular anecdote (although one contradicted by historical
> evidence) recounts that the 11th-century Hoysala king Veera Ballala
> II, while on a hunting expedition, lost his way in the forest. Tired
> and hungry, he came across a poor old woman who served him boiled
> beans. The grateful king named the place "benda kaal-ooru"
> (literally, "town of boiled beans"), which was eventually
> colloquialised to "Bengaluru".

***

(This information has been retrieved from
[Wikipedia] (http://en.wikipedia.org/wiki/Bangalore)
under the GNU Free Documentation License.)
```

4. Beachten Sie, wie verschiedene Teile der Datei, z.B. der Dateikopf und hervorgehobene Wörter automatisch hervorgehoben wurden. Dies sollte das Schreiben in *Markdown*-Syntax hoffentlich einfacher machen.

13.9 Ein Syntaxschema schreiben

Lassen Sie uns jetzt versuchen eine eigene *Syntax*-datei für das *AmiFormat*⁵⁶-Textformat zu schreiben.

Bei der *Syntax*-hervorhebung dreht es sich im Kern um zwei Schritte - zuerst müssen wir die Art des Textformates bestimmen, nachdem wir suchen, und das zweite ist die Beschreibung, wie es angezeigt wird.

Angenommen wir wollen z.B. sämtliche Belegstellen finden von `` irgendein Wort `` und es fettgedruckt angezeigt kriegen. Zuerst müssen wir ein solches Muster in unserem Text finden, und dann den Namen dieses Musters mit der Art, wie wir es angezeigt haben wollen, verknüpfen:

```
:syntax match ourBold /<b>.*</b>/
:highlight default ourBold term=bold cterm=bold gui=bold
```

Die erste Zeile sagt, dass wir einen neuen *Syntax*-typ erschaffen, auf der Grundlage ein Muster zu finden, mit Namen `'ourBold'` und das regex-Muster, wie oben angegeben (*regex* → *regular expression*).

Die zweite Zeile sagt, dass wir die `'ourBold'`-*Syntax* hervorheben wollen. Das Schema das wir festlegen ist das Standardschema, was bedeutet, dass es von anderen Benutzern oder Farbschemata überschrieben werden kann. Wir können drei verschiedene Arten festlegen wie *ourBold* vertreten wird, für die drei verschiedenen Anzeigearten, die im Vim

⁵⁶<http://orangoo.com/labs/AmiNation/AmiFormat/>

laufen - schwarz/weiß-Terminals, Farbterminals und die grafische Benutzerschnittstelle (GUI → Graphical User Interface).

Manchmal wollen wir eine Aufgabe in unserem Text als *todo*-Element festlegen, und für gewöhnlich schreiben wir dies in Großbuchstaben 'TODO'. Aber was ist, wenn uns das noch nicht reicht?

```
:syntax keyword ourTodo TODOFIXMEXXX
:hi def link ourTodo Todo
```

Zuerst legen wir fest, dass 'ourTodo' aus Schlüsselwörtern besteht - das sind ganz einfache Wörter, die hervorgehoben werden müssen, und wir verknüpfen diese Mustergruppe 'ourTodo' mit einer schon vorhandenen Gruppe 'Todo' in Vim. Es gibt viele dieser bereits vorhandenen Gruppen in Vim, die vorbestimmte Farbschemata haben. Das Beste ist es, unsere *Syntaxe*eigenheiten mit schon vorhandenen Gruppen zu verknüpfen. Sehen Sie sich hierzu auch **:help group-name** an, für eine Liste der zur Verfügung stehenden Gruppen.

Als nächstes können wir spezifizierte Code-Blöcke in der Gruppe haben, die in **[code]** .. **[/code]** eingeschlossen sind. Wie heben wir diese hervor?

```
:syn region amiCode excludenl start=/\[code\]/ end=/\[\/code\]/
:hi def link amiCode Identifier
```

Zuerst legen wir fest, dass wir eine Textregion definieren, mit einem Start- und Endmuster (was sehr einfach ist, in unserem Fall), und dann verknüpfen wir dies mit der schon vorhandenen '**Identifier**'-Klasse.

So ähnlich können wir auch vorgehen um Definitionen zu anderen Textteilen hinzuzufügen, so wie es im *AmiFormat*-Nachschlagehandbuch⁵⁷ beschrieben ist, und das endgültige Skript könnte dann so aussehen:

```
" Vim syntax file for AmiFormat
" Language: AmiFormat
" Version: 1
" Last Change: 2011 Aug 11
" Maintainer: www.swaroopch.com/contact/
" License: www.opensource.org/licenses/bsd-license.php
" Reference: http://orangoo.com/labs/AmiNation/AmiFormat/

"""""""" Initial Checks """"""""""

" To be compatible with Vim 5.8. See ':help 44.12'
if version < 600
    syntax clear
```

⁵⁷<http://orangoo.com/labs/AmiNation/AmiFormat/online%20reference/>

```

elseif exists("b:current_syntax")
" Quit when a (custom) syntax file was already loaded
    finish
endif

"""""""""" Patterns """"""""""

" Emphasis
syn match amiItalic /\&lt;i>.\{-}</i>/
syn match amiBold /\&lt;b>.\{-}</b>/

" Todo
syn keyword amiTodo TODOFIXMEXXX

" Headings
syn match amiHeading /\^h[1-6]\.\s\+.\{-}$ /

" Lists
syn match amiList /\^\s*\*\s\+ /
syn match amiList /\^\s*d\+\.\s\+ /

" Classes
syn match amiClass /\^\s*%(w\+).*% /
syn match amiClass /\^\s*%{.*}% /

" Code
syn region amiCode excludenl start=/\[code\]/ end=/\[\/code\]/

" HML
syn region amiEscape excludenl start=/\[escape\]/ end=/\[\/escape\]/

" Link
syn match amiLink /\.\{-}":(\{-}) /

" Image
syn match amiImage /\!\{-}(\{-})! /

"""""""""" Highlighting """"""""""

hi def amiItalic term=italic cterm=italic gui=italic
hi def amiBold term=bold cterm=bold gui=bold

hi def link amiHeading Title
hi def link amiTodo Todo

```

```

hi def link amiList PreProc
hi def link amiClass Statement
hi def link amiCode Identifier
hi def link amiEscape Comment
hi def link amiLink String
hi def link amiImage String

"""""""""" Finish """"""""""

" Set syntax name
let b:current_syntax = "amifmt"

```

Jetzt wo das Skript tatsächlich funktioniert, habe ich es in die Vim-Skripte-Abteilung⁵⁸ hochgeladen, und zwar als ich es geschrieben habe! Jetzt kann jeder auf dem Planeten die *AmiFormat-Syntax* hervorhebung in Vim benutzen.

Um noch mehr über *Syntax* hervorhebungsskripte für den Vim zu erfahren, nehmen Sie:

- `:help syntax`
- `:help usr_44.txt`
- `:help group-name`
- `:help pattern-overview`
- `:help mysyntaxfile`
- `:help new-filetype`

Hinweis Wenn Sie Ihren Bildschirm mal zeichnerisch auffrischen möchten, weil die *Syntax*datei, Ihre Ausgabe verhunzt, drücken Sie **CTRL-L**.

Hinweis Sie haben es vielleicht schon erraten, dass, wenn wir früher den Dateityp gesetzt haben, Vim die *Syntax* seinerseits automatisch auf denselben Namen setzt.

13.10 Compiler-Plugins

Compiler-Plugins werden benutzt um Programme, die in verschiedenen Sprachen geschrieben wurden, in Maschinsprache zu übersetzen. Sie sind überall dort nützlich, wo eine Umformung nötig ist, von einem einfachen Text in ein anderes Format. Selbst wenn Sie einen einfachen Text in *Markdown* nach HTML umwandeln wollen, benutzen Sie ein Umwandlungsprogramm.

Lassen Sie uns die Benutzung eines Compiler-Plugins für Python untersuchen.

⁵⁸http://www.vim.org/scripts/script.php?script_id=1745

1. Laden Sie sich das `compiler/python.vim`⁵⁹-Skript herunter und schieben Sie es in Ihr `~/.vim/compiler/`-Verzeichnis.
2. Fügen Sie die folgende Zeile in Ihre `~/.vimrc` ein:

```
:autocmd BufNewFile,BufRead *.py compiler python
```

3. Starten Sie den Vim erneut, und öffnen Sie eine *Python*-Datei, sagen wir mal `test.py`, und geben Sie das folgende Programm ein:

```
#!/python
print 'Hello World'
```

4. Führen Sie `:make` aus, und Sie sollten eine erfolgreiche Übersetzung sehen.
5. Lassen Sie uns absichtlich einen Fehler in das Programm einfügen, indem wir `'print'` → `(drucken)`, ändern in `'pritrn'`:

```
pritrn 'Hello World'
```

Führen Sie jetzt wieder `:make` aus, und Sie sehen, dass der Fehler angezeigt wird, und Vim den **Cursor** automatisch auf die fehlerbehaftete Zeile setzt!

6. Führen Sie `:clist` aus, um die ganze Fehlerliste zu sehen.
7. Nachdem Sie einen Fehler behoben haben, können Sie `:cnext` ausführen, um zum nächsten Fehler zu springen.

Wenn Sie das `compiler/python.vim`-Skript, das wir heruntergeladen haben, öffnen, werden Sie feststellen, dass es geradezu trivial ist es sind nur zwei Variablen definiert – eine davon ist `makeprg`, welche definiert, wie die Datei 'gemacht' werden soll, d.h. wie sie zu kompilieren ist, und die zweite ist `errorformat`, welche die Form der Compilerfehlerausgabe bestimmt.

Ich habe mit den gleichen beiden Variablen – ein Compiler-**Plugin** für *Adobe Flex*⁶⁰ geschrieben.

Für weitere Details, sehen Sie sich hierzu auch `:help write-compiler-plugin` und `:help quickfix` an, wie Sie Ihre eigenen Compiler-**Plugins** schreiben.

⁵⁹http://www.vim.org/scripts/script.php?script_id=1439

⁶⁰http://www.vim.org/scripts/script.php?script_id=1746

13.11 Hausaufgabe : Schreiben Sie ein globales Plugin

Um Ihre neuerworbenen Plugin-Programmierfähigkeiten zu praktizieren, versuchen Sie sich mal an dieser Übung:

Schreiben Sie ein Plugin, dass sämtliche doppelten Zeilen, und überflüssige Leerzeilen des Dokumentes löscht.

Entweder Sie benutzen hierfür Vim's eigene Skriptsprache oder jede andere Sprache, die eine Schnittstelle zu Vim hat.

Wenn Sie einen kleinen 'Anstubser' brauchen, nehmen Sie diesen Tipp⁶¹. Oder wie wär's hiermit?

Schreiben Sie ein Skript um sich die Bedeutung und Synonyme für das aktuelle Wort unter der Schreibmarke zu holen.

Nochmal; den 'Anstubser' kriegen Sie über mein lookup.vim-Plugin⁶².

13.12 Plugins abschalten

Angenommen Sie finden, dass Vim schräge Dinge tut, und Sie haben als Ursache hierfür ein Plugin in Verdacht, dann können Sie Vim zu einer selektiven Initialisierung veranlassen, mit dem `-u` Befehlszeilenargument.

`vim -u NONE` z.B., startet den Vim ohne irgenwelche Initialisierungsskripte auszuführen. Das ist der reine Vim, wie er in seinem Rohzustand läuft. Benutzen Sie `vim -u ihre-minimale-initialisierung.vim`, um ausschließlich ganz bestimmte Initialisierungen laufen zu lassen, die Sie benötigen. Diese Möglichkeit hilft Ihnen bei der Fehlerbeseitigung, wenn irgendwelche Probleme im Vim auftauchen oder durch ein Plugin eingeschleppt wurden.

Sehen Sie hierzu `:help -u` und `:help starting` für Einzelheiten.

13.13 Zusammenfassung

Wir haben die verschiedenen Arten von Plugins gesehen, die es für den Vim gibt, wie man solche Plugins benutzt und schreibt. Jetzt haben wir eine Ahnung davon erhalten wie erweiterbar Vim ist, und wie wir Plugins schreiben können, um uns das Leben zu erleichtern.

14 Ein Editor für Programmierer

14.1 Einführung

Vim neigt schwer dazu von Programmierern eingesetzt zu werden. Die Funktionalität, die Bedienerfreundlichkeit, und Flexibilität, die Vim mitliefert, machen ihn zu einer guten

⁶¹http://vim.wikia.com/wiki/Remove_unwanted_empty_lines

⁶²http://www.vim.org/scripts/script.php?script_id=2001

Wahl für Leute die häufig programmieren. Das sollte nicht allzu überraschend sein, da Programmieren auch viel mit Editieren zu tun hat.

Lassen Sie es mich noch einmal wiederholen, dass Schreibfertigkeit für einen Programmierer von entscheidender Bedeutung ist. Wenn Sie unsere obigen Ausführungen noch nicht überzeugen konnten, dann wird es hoffentlich dieser Artikel von *Jeff Atwood* – *'We are Typist First, Programmers Second'*⁶³ (→ *'Zu allererst sind wir Tipper, dann Programmierer'*).

Wenn Sie keinerlei Programmiererfahrung besitzen, können Sie dieses Kapitel natürlich überspringen.

Für alle, die ins Programmieren ganz verliebt sind, lassen Sie uns eintauchen: in Sachen *Programmieren* hat es Vim echt drauf.

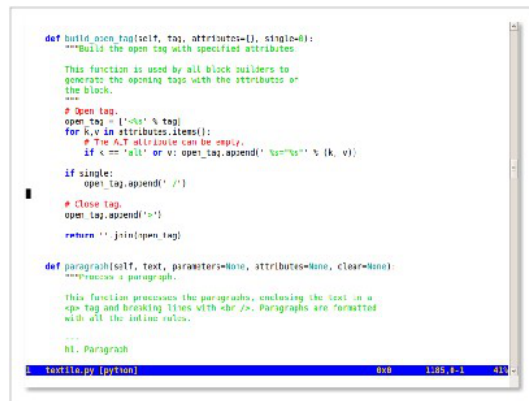
14.2 Einfaches Zeug

Die einfachste Funktion von Vim, die Sie für sich nutzbar machen können beim Programmieren, ist die Syntaxhervorhebung. Dies ermöglicht Ihnen Ihren Code 'sichtbar' zu machen, und damit können Sie Ihren Code auch schneller lesen und schreiben, und Sie machen weniger ganz offensichtliche Fehler.

14.2.1 Syntaxhervorhebung

Angenommen Sie editieren eine vim-syntax-Datei. Führen Sie `:set filetype=vim` aus, und sehen Sie wie Vim Farbe hinzufügt. Ähnlich ist es, wenn Sie eine Python-Datei editieren. Führen Sie `:set filetype=python` aus. Um eine Liste der zur Verfügung stehenden Sprachtypen zu sehen überprüfen Sie das `$VIMRUNTIME/syntax/`-Verzeichnis.

Tip Wenn Sie die ganze Pracht der Syntaxhervorhebung für jegliche Unix-Shell-Ausgabe haben wollen, leiten Sie es einfach durch die Röhre (→to pipe) an Vim, z.B. `svn diff | vim -R -`. Beachten Sie den Bindestrich am Ende, der Vim sagt, dass er Text von seiner Standardeingabe einlesen soll.



14.2.2 Intelligentes Einrücken

Der Code eines erfahrenen Programmierers ist normalerweise sauber eingerückt, was ihm ein 'uniformes' Aussehen verleiht, und die Struktur offensichtlicher erscheinen lässt. Vim erledigt das Einrücken für Sie, während Sie sich schön auf Ihren Code konzentrieren.

⁶³<http://www.codinghorror.com/blog/archives/001188.html>

Wenn Sie eine bestimmte Zeile einrücken und möchten, dass die nachfolgenden Zeilen auf gleicher Höhe eingerückt werden, dann können Sie die `:set autoindent`-Einstellung verwenden.

Wenn Sie mit einem neuen Block von Anweisungen anfangen, und die nächste Zeile automatisch zur nächsten Stufe eingerückt haben wollen, verwenden Sie die `:set smartindent`-Einstellung. Beachten Sie hierbei, dass das Verhalten dieser Einstellung, von der jeweiligen Programmiersprache abhängt, die gerade benutzt wird.

14.2.3 Hüpfen

Wenn die Programmiersprache Ihrer Wahl geschwungene Klammern benutzt, um Anweisungsblöcke abzugrenzen, setzen Sie Ihren **Cursor** auf eine der geschwungenen Klammern, und drücken Sie den `%`-Schlüssel, um zur entsprechenden geschwungenen Klammer zu springen. Mit diesem *'Hüpf'*-Schlüssel können Sie schnell zwischen Blockanfang und Blockende springen.

14.2.4 Shell-Befehle

Shell-Befehle können Sie vom Vim aus mit dem `:!`-Kommando ausführen.

Zum Beispiel, wenn der `date`-Befehl auf Ihrem Betriebssystem verfügbar ist, führen Sie `!:date` aus, und schon sollten Sie das Datum und die aktuelle Uhrzeit sehen.

Das ist nützlich in Situationen, wo Sie mal schnell was in Ihrem Dateisystem checken wollen, z.B. den Inhalt des gerade benutzten Verzeichnisses mit `!:ls` oder `!:dir` usw.

Wenn Sie Zugriff auf eine vollständige **Shell** haben möchten, führen Sie `:sh` aus.

Wir können diese Einrichtung nutzen um den aktuell bearbeitenden Text durch externe Filter laufen zu lassen. Wenn Sie z.B. einen Haufen Zeilen haben, die Sie sortieren möchten, können Sie das hier ausführen `:%!sort`. Dies übergibt den aktuellen Text an den `sort`-Befehl der Shell, und die Ausgabe dieses Befehls ersetzt dann den aktuellen Inhalt der Datei.

14.2.5 Herumspringen

Es gibt viele Möglichkeiten im Programmcode herumzuspringen.

- Setzen Sie Ihren **Cursor** auf einen Dateinamen im Code und drücken Sie dann `gf`, um die Datei zu öffnen.
- Setzen Sie Ihren **Cursor** auf einen Variablennamen und drücken Sie `gd`, um sich zur lokalen Definition des Variablennamens zu bewegen. `gD` erreicht dasselbe für die globale Deklaration, indem es die Datei von Anfang an durchsucht.
- Benutzen Sie `]]`, um sich zur nächsten `{` der ersten Reihe zu bewegen. Es gibt noch jede Menge ähnliche Bewegungsarten für Einzelheiten, sehen Sie sich hierzu `:help object-motions` an.

- Sehen Sie sich `:help 29.3`, `:help 29.4`, und `:help 29.5` für mehr solcher Kommandos. Zum Beispiel `[I` zeigt Ihnen alle Zeilen an, die das Schlüsselwort enthalten unter dem `Cursor`!

14.3 Teile des Codes durchstöbern

14.3.1 Dateisystem

Benutzen Sie `:Vex` oder `:Sex` um das Dateisystem innerhalb von Vim zu durchstöbern, und anschließend die entsprechende Datei zu öffnen.

14.3.2 ctags (→etwa: C-Kennzeichnungen / C-Markierungen)

Wir haben nun gesehen, wie wir einfache Bewegungen innerhalb ein und derselben Datei bewerkstelligen. Aber was ist, wenn wir uns zwischen unterschiedlichen Dateien bewegen und Querverweise haben wollen? Dann können wir hierfür `tags` benutzen.

Für einfaches Durchsuchen einer Datei, nehmen wir das `taglist.vim`-Plugin.

1. Installieren Sie das üppig überbordende (→*exuberant*) `ctags`⁶⁴-Programm.
2. Installieren Sie das `taglist.vim`⁶⁵-Plugin. Schlagen Sie die „Installationserläuterungen“ in der Skriptseite nach.
3. Führen Sie aus `:TlistToggle`, um das Taglist-Fenster zu öffnen. Na bitte – jetzt können Sie die einzelnen Teile Ihres Programmes, wie Makros, Typendefinitionen, Variablen und Funktionen durchgehen.
4. Sie können `:tag foo` verwenden, um zur Definition `foo` zu springen.
5. Platzieren Sie Ihren `Cursor` auf irgendein Symbol und drücken Sie `ctrl-]`, um zur Definition dieses Symbols zu springen. Drücken Sie `ctrl-t`, um zum vorherigen Code zurückzukehren.
6. Benutzen Sie `ctrl-w]` um zur Symbolbeschreibung zu springen, in einem geteilten Fenster.
7. Benutzen Sie `:tnext`, `:tprev`, `:tfirst`, `:tlast`, um sich zu den entsprechenden Markierungen hinzubewegen.

⁶⁴<http://ctags.sourceforge.net>

⁶⁵http://www.vim.org/scripts/script.php?script_id=273



Beachten Sie bitte, dass das herrlich überschäumende und prickelnde Programm *Ctags*, zum Zeitpunkt der Fertigstellung dieses Buches, 33 Programmiersprachen unterstützt, und es leicht auf weitere Programmiersprachen erweitert werden kann.

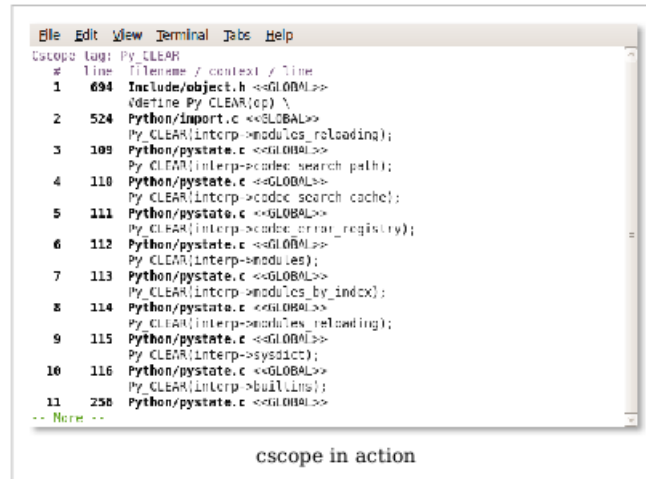
`:help taglist-intro` gibt Ihnen hierzu die Einzelheiten.

14.3.3 cscope (wie ctags aber gedopt → :help cscope)

Um in der Lage zu sein, dateiübergreifend zu Definitionen zu springen, brauchen wir ein Programm wie *cscope*. Wie auch immer, der Name deutet es ja schon an, dieses spezielle Programm arbeitet nur in Zusammenhang mit der Programmiersprache C.

1. Installieren Sie *cscope*. Bemühen Sie `:help cscope-info` und `:help cscope-win32` was die Installation betrifft.
2. Kopieren Sie `cscope_maps.vim`⁶⁶ in Ihr `~/.vim/plugin`-Verzeichnis.
3. Wechseln Sie in Ihr Quellcodeverzeichnis und führen Sie `cscope -R -b` aus, um den Datenbestand (die Datenbank) `'r'`rekursiv für alle Unterverzeichnisse zu erstellen (\rightarrow `'b'`uild).
4. Starten Sie den Vim erneut, und öffnen Sie eine Quellcodedatei.
5. Führen Sie `:cscope show` aus, um sich bestätigen zu lassen, dass eine cscope-Verbindung erstellt wurde.
6. Führen Sie `:cscope find symbol foo` aus, um das Symbol foo zu lokalisieren. Der Befehl lässt sich auf folgende Weise abkürzen `:cs fs foo`.

⁶⁶http://cscope.sourceforge.net/cscope_maps.vim



Sie können auch:

- Diese Definition finden - `:cs f g`
- Funktionen finden, die von dieser Funktion aufgerufen wurden - `:cs f d`
- Funktionen finden, die diese Funktion aufrufen - `:cs f c`
- Diese Zeichenkette finden - `:cs f t`
- Dieses egrep-Muster finden - `:cs f e`

Sehen Sie sich hierzu auch `:help cscope-suggestions` für die angeratene Benutzung von cscope im Vim an.

Und das *QuellCodeGehorsams-Programmerweiterungsmodul*⁶⁷ (→Source Code Obedience plugin) ist es sicherlich auch wert, einmal ausprobiert zu werden, da es einfache Tastaturschlüssel für cscope/ctags zur Verfügung stellt.

Wo wir schon mal bei der Programmiersprache C sind das `c.vim`⁶⁸-Plugin kann auch verdammt praktisch sein.

14.4 Kompilierung (Programmübersetzung)

Im vorhergehenden Kapitel haben wir schon die Bedeutung von `:make` bezüglich der Programme, die wir schreiben gesehen, also wiederholen wir es hier nicht.

14.5 Schreiben leicht gemacht

14.5.1 Omnicompletion (→etwa: die Allesvervollständigung)

Eine der Funktionen, nach denen am meisten verlangt wurde, und die dem Vim 7 hinzugefügt wurde, ist „*omnicompletion*“, bei der Text auto-vervollständigt werden kann,

⁶⁷http://www.vim.org/scripts/script.php?script_id=1638

⁶⁸http://vim.sourceforge.net/scripts/script.php?script_id=213

auf Grundlage des aktuellen Kontextes. Wenn Sie z.B. einen langen Variablennamen verwenden, und diesen Variablennamen wiederholt benutzen, können Sie ein Tastaturkürzel nehmen, um Vim nach einer Auto-Vervollständigung zu fragen, und Vim erledigt den Rest.

Vim erreicht dies über ftplugins, besonders diejenigen mit dem Namen `ftplugin/<Sprache>complete.vim`, wie `pythoncomplete.vim`.

Anfangen wollen wir mit einem einfachen Pythonprogramm:

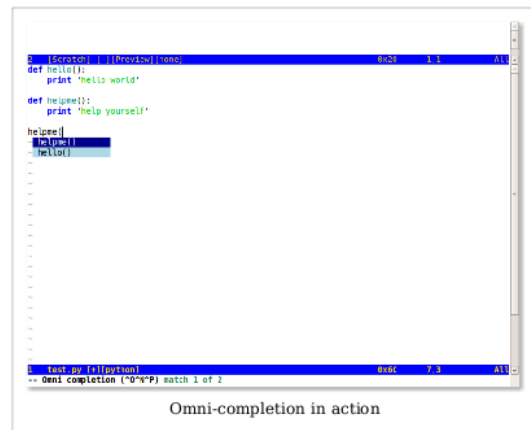
```
def hello():
    print 'hello world'

def helpe():
    print 'help yourself'
```

Nachdem Sie dieses Programm eingegeben haben, beginnen Sie eine neue Zeile in derselben Datei. Tippen Sie `'he'` und drücken Sie `ctrl-x ctrl-o`, was Ihnen Vorschläge für die Autovervollständigung anzeigen wird.

Wenn Sie eine Fehlermeldung wie `E764: Option 'omnifunc' is not set`, dann führen Sie `:runtime! autoload/pythoncomplete.vim` aus, um das omnicompletion-Plugin zu laden.

Damit Sie dies nicht jedesmal tun müssen, fügen Sie folgende Zeile in Ihre `~/.vimrc` ein:



```
:autocmd FileType python runtime! autoload/pythoncomplete.vim
```

Vim benutzt automatisch den ersten Vorschlag, mit `ctrl-n` und `ctrl-p`, können Sie zur nächsten, bzw. vorherigen Auswahl wechseln.

Wenn Sie die Benutzung von omnicompletion abbrechen möchten, drücken Sie einfach `<ESC>`.

Schlagen Sie `:help new-omni-completion` für Näheres sowohl zu den unterstützten Sprachen nach (C, HTML, JavaScript, PHP, Python, Ruby, SQL, XML,...), als auch, wie Sie Ihre eigenen omnicompletion-Skripte schreiben können.

Hinweis Wenn es für Sie bequemer ist die Pfeiltasten zu benutzen, um Ihre Auswahl von der omnicompletion-Liste zu treffen, sehen Sie sich hierfür Vim-Tipp 1228⁶⁹ an.

⁶⁹http://www.vim.org/tips/tip.php?tip_id=1228

Ich bevorzuge ein einfaches **ctrl-Leertaste** statt der sperrigen **ctrl-x ctrl-o**- Tastenkombination. Mit einem Eintrag in Ihre *vimrc*, können Sie das auch haben:

```
:imap <c-space> <c-x><c-o>
```

Das *PySmell*⁷⁰-Plugin, könnte in dieser Hinsicht hilfreich sein für Vim-Benutzer, die in Python programmieren.

14.5.2 Schnipsel benutzen

Codeschnipsel sind kleine Stücke Programmiercodes, bei denen Sie dazu neigen, sie immer wieder neu zu schreiben. Wie alle guten, faulen Programmierer, können Sie dazu ein **Plugin** benutzen. In unserem Fall nehmen wir das phantastische **SnippetsEmu**-Plugin.

1. Laden Sie sich das **snippetsEmu-Plugin**⁷¹ herunter.
2. Erstellen Sie Ihr `~/.vim/after/`-Verzeichnis, so es nicht bereits existiert
3. Starten Sie Vim indem Sie diesen **Plugin**namen in der Kommandozeile hinzufügen. Zum Beispiel: **gvim snippy_bundles.vba**.
4. Führen Sie aus **:source %**. Der 'vimball' ($\rightarrow .vba=vimball$) entpackt sich nun, und legt die vielen Dateien in den richtigen Verzeichnissen ab.
5. Wiederholen Sie denselben Arbeitsgang für **snippy_plugin.vba**.

So! Jetzt probieren wir das **Plugin** mal aus.

1. Öffnen Sie eine neue Datei, sagen wir mal, **test.py**.
2. Drücken Sie die Tasten **d, e, f** und dann **<tab>**.
3. Bitte schön! Haben Sie gesehen, wie snippetsEmu bereits die Struktur Ihrer Funktion erstellt hat? Jetzt sollten Sie dies hier in Ihrer Datei sehen:

```
def <{fname}>(<{args}>):
    """
    <{}>
    <{args}>"""
    <{pass}>
    <{}>
```

Hinweis In dem Fall das Sie nach dem Eintippen von **def<tab>** nichts passiert, dann ist das Schnipsel **Plugin** nicht geladen. Starten Sie **:runtime! ftpplugin/python_snippets.vim**.

⁷⁰<http://code.google.com/p/pysmell/>

⁷¹http://www.vim.org/scripts/script.php?script_id=1318

4. Ihr **Cursor** steht jetzt auf dem Funktionsnamen, d.h. **fname**.
5. Tippen Sie den Funktionsnamen, sagen wir, **test**.
6. Drücken Sie **<tab>** und der **Cursor** bewegt sich automatisch zu den Argumenten. **'tab'** nochmal drücken, um sich zum nächsten Punkt zu bewegen, der ausgefüllt werden muss.
7. Jetzt geben Sie einen Kommentar ein: **Einfach Hallo sagen**.
8. Nochmal auf **'tab'** drücken, und **print 'Hallo Welt'** eingeben.
9. Auf **'tab'** drücken.
10. Fertig ist Ihr Programm!

Sehen sollten Sie jetzt:

```
def test():  
    """  
    Just say Hi  
    """  
    print 'Hello World'
```

Das beste daran ist, dass SnippetsEmu eine Standardkonvention ermöglicht, die man zu befolgen hat, und dass niemand in der Arbeitsgruppe dies 'vergisst'.

14.5.3 Schnipsel erstellen

Wir machen jetzt mal unsere eigenen Schnipsel.

Lassen Sie uns einmal den Fall durchgehen, wo ich dazu neige, immer wieder dieselbe Art folgenden Programmcodes in ActionScript3 zu schreiben:

```
private var _foo:Object;  
  
public function get foo():Object  
{  
    return _foo;  
}  
  
public function set foo(value:Object)  
{  
    _foo = value;  
}
```

Das ist eine einfache *getter/setter-Kombination*, die eine stützende Variable benutzt. Das Problem ist nur, dass das fürchterlich viel 'Textbausteincode' ist, den man immer wieder schreiben muss. Das wollen wir automatisieren.

Die SnippetsEmu-Sprachschnipsel-**Plugins** nehmen **st** als Startmarkierung, und **et** als Endmarkierung. Das sind die gleichen pfeiltypischen Symbole die Sie im Zwischenraum unseres Codes sehen.

Fangen wir mit einem einfachen Beispiel an.

```
:exec "Snippet pubfun public function ".st.et." :".st.et."<CB>{<CB>".st.et."<CB>}<CB>"
```

Fügen Sie die obige Zeile in Ihr `~/.vim/after/ftplugin/actionscript_snippets.vim` ein.

Jetzt öffnen Sie eine neue Datei, sagen wir, `test.as`, und dann tippen Sie `pubfun` und `<tab>` und sehen es erweitert zu:

```
public function <{}>:<{}>
{
}

```

Der **Cursor** wird auf den Funktionsnamen gesetzt `'tab'`, um den Rückgabebetyp der Funktion zu bekommen nochmal `'tab'`, um den Funktionskörper einzugeben.

Aber zurück, zu unserem eigentlichen Problem. So habe ich das gelöst:

```
:exec "Snippet getset private var _".st."name".et.";<CB><CB>public function get ".st."name".et."():".st."type".et."<CB>{<CB><tab>return _".st."name".et.";<CB>}<CB><CB>public function set ".st."name".et."(value:".st."type".et.")<CB>{<CB><tab>_".st."name".et." = value;<CB>}<CB>"
```

Hinweis Alle Schnipsel für dieses **Plugin** *müssen* in einer einzigen Zeile eingegeben werden. Das ist eine technische Schranke.

Folgen Sie derselben Prozedur wie oben, um das neue Schnipsel zu benutzen:

1. Fügen Sie diese Zeile in Ihr `~/.vim/after/ftplugin/actionscript_snippets.vim` ein.
2. Öffnen Sie eine neue Datei, wie `test.as`.
3. Tippen Sie `getset`, drücken Sie `<tab>` und Sie werden das hier sehen:

```
private var _<{name}>;
```

```

public function get <{name}>():<{type}>
{
    return _<{name}>;
}

public function set <{name}>(value:<{type}>)
{
    _<{name}> = value;
}

```

4. Geben Sie **color** ein und drücken Sie **<tab>**. Beachten Sie, dass der Variablenname *color* überall ersetzt worden ist.
5. Tippen Sie **Number** und drücken Sie **<tab>**. Der Code sieht nun so aus:

```

private var _color;

public function get color():Number
{
    return _color;
}

public function set color(value:Number)
{
    _color = value;
}

```

Beachten Sie, wieviele Tastaturanschläge wir vermieden haben! Mit einer einzigen Vim-Skript-Zeile haben wir etwa 11 Zeilen, sich wiederholenden Programmiercodes ersetzt.

Mit solchen Schnipseln können wir unsere Programmierfaulheit betonen, und dies hilft uns dabei, uns auf die eigentliche Programmiertätigkeit zu konzentrieren.

Sehen Sie sich hierzu **:help snippets.emu.txt** für weitere Einzelheiten an (Diese Hilfedatei steht Ihnen nur dann zur Verfügung, wenn Sie das **Plugin** installiert haben).

14.6 Integrierte Entwicklungsumgebung (→IDE=integrated developer environment)

Mit ein paar **Plugins** kann Vim als Entwicklungsumgebung genutzt werden.

14.6.1 Projekt **Plugin**

Das Projekt **Plugin** wird verwendet, um eine Art Projekt-Manager-Benutzung für den Vim zu erschaffen.

1. Laden Sie sich das `project plugin`⁷² herunter.
2. Entpacken Sie es in Ihrem `~/vim/`-Verzeichnis.
3. Führen Sie `:helptags ~/.vim/doc/` aus.
4. Laden Sie sich den Vim-Quellcode von <http://www.vim.org/subversion.php> herunter.
5. Führen Sie `:Project` aus. Es öffnet sich links eine Seitenleiste, die Ihnen als 'Projekt-Fenster' dient.
6. Führen Sie `\c` aus (Gegenschrägstrich, gefolgt von 'c').
7. Beantworten Sie die folgenden Optionen
 - Eingangsname, sagen wir `'vim7_src'`
 - Verzeichnis, z.B. `C:\repo\vim7\src\`
 - CD-Option, das gleiche wie für Verzeichnis, oben-
 - Filter-Option, sagen wir `*.h *.c`.
8. Sie sehen wie sich die Seitenleiste auffüllt mit der Liste der Dateien, die mit dem Filter in dem angegebenen Verzeichnis übereinstimmen.
9. Mit den Pfeiltasten (oder `j/k`) können Sie in der Dateiliste rauf und runter fahren, mit Drücken der `<enter>`-Taste öffnen Sie die Datei im Hauptfenster.

Das gibt Ihnen die vertraute IDE-artige Schnittstelle, und das Gute daran ist, es gibt keine ausgefallenen Konfigurationsdateien oder schwachsinnige *Pfadeinrichtungen* in Entwicklungsumgebungen, die immer solche Belange haben. Die Funktionsweise des Projekt-Plugins ist einfach und geradeheraus.

Sie können die Standardfaltungsbefehle nehmen, um Projekte und ihre Einzelheiten zu öffnen und zu schließen.

Sie können auch Skripte ausführen am Anfang und Ende der Benutzung eines Projektes, damit richten Sie den Pfad ein, oder setzen Compileroptionen etc.

Sehen Sie sich hierzu `:help project.txt` an.

14.6.2 Programmcode aus dem Text heraus starten

Sie können Programmcode direkt aus Vim heraus starten, in dem Sie Plugins wie `EvalSelection.vim`⁷³ verwenden oder noch einfacher: `inc-python.vim`⁷⁴.

⁷²http://www.vim.org/scripts/script.php?script_id=69

⁷³http://www.vim.org/scripts/script.php?script_id=889

⁷⁴http://www.vim.org/scripts/script.php?script_id=1941

14.6.3 Integration von SCM (→SCM=software configuration management)

Wenn Sie damit anfangen eine Datei zu editieren, können Sie sie sogar automatisch durch *Perforce* überprüfen lassen, indem Sie das **perforce**-Plugin⁷⁵ verwenden. Ganz ähnlich verhält es sich mit dem CVS/SVN/SVK/Git Integrations-Plugin⁷⁶.

14.6.4 Mehr

Um weitere **Plugins** ausfindig zu machen, die ein IDE-artiges Verhalten im Vim erzeugen, sehen Sie sich hier um:

- Vim Tip: Der Vim als eierlegende Wollmilchsau-Entwicklungsumgebung⁷⁷
- C++/Python Vim+IDE **Plugin**-Liste⁷⁸.

Es gibt noch mehr sprachspezifische **Plugins**, mit denen Sie raffinierte Dinge tun können. Im Falle von Python könnten diese **Plugins** hilfreich sein:

- **SuperTab**⁷⁹ erlaubt Ihnen Omni-completion durch einfaches Drücken von **'tab'** aufzurufen, und die verschiedenen Optionen mit den Pfeiltasten auszuwählen.
- **python_calltips**⁸⁰ zeigt ein Fenster im unteren Bereich, das Ihnen eine Liste von Möglichkeiten für Ergänzungen gibt. Richtig klasse hierbei ist – im Vergleich zu Omni-completion – dass Sie sich die Dokumentation für jede dieser Möglichkeiten anschauen können.
- VimPdb⁸¹ hilft Ihnen beim Ausbessern von Pythonprogrammen, ohne Vim zu verlassen.

14.7 Seine eigenen **Plugins** schreiben

Um Vim in jeder erdenklichen Weise zu erweitern, können Sie Ihre eigenen **Plugins** schreiben. Hier ist mal eine Aufgabe, an die Sie sich ranmachen können:

Schreiben Sie ein Vim-**Plugin**, das das aktuelle Wort nimmt, und einen **Browser** öffnet, mit der Dokumentation zu genau diesem Wort (das Wort kann ein Funktionsname oder der Name einer Klasse etc. sein).

Wenn Ihnen so gar nicht einfallen will, wie Sie dies erreichen können, dann werfen Sie mal einen Blick auf den „*Online documentation for word under cursor*“-Tipp im Vim-Tipps-Wiki.⁸²

⁷⁵http://www.vim.org/scripts/script.php?script_id=240

⁷⁶http://www.vim.org/scripts/script.php?script_id=90

⁷⁷http://vim.wikia.com/wiki/Using_vim_as_an_IDE_all_in_one

⁷⁸<http://phraptured.net/vimmity-vim-vim.html>

⁷⁹http://www.vim.org/scripts/script.php?script_id=1643

⁸⁰http://www.vim.org/scripts/script.php?script_id=1074

⁸¹http://www.vim.org/scripts/script.php?script_id=2043

⁸²http://vim.wikia.com/wiki/Online_documentation_for_word_under_cursor

Ich habe denselben Tipp erweitert, und etwas allgemeiner gehalten:

```

" Add the following lines to your ~/.vimrc to enable online documentation
" Inspiration: http://vim.wikia.com/wiki/Online\_documentation\_for\_word\_under\_cursor

function Browser()
  if has("win32") || has("win64")
    let s:browser = "C:\\Program Files\\Mozilla Firefox\\firefox.exe -new-tab"
" Cygwin
  elseif has("win32unix")
    let s:browser = "'/cygdrive/c/Program Files/Mozilla Firefox/firefox.exe' -new-tab"
  elseif has("mac") || has("macunix") || has("unix")
    let s:browser = "firefox -new-tab"
  endif

  return s:browser
endfunction

function Run(command)
  if has("win32") || has("win64")
    let s:startCommand = "!start"
    let s:endCommand = ""
" TODO Untested on Mac
  elseif has("mac") || has("macunix")
    let s:startCommand = "!open -a"
    let s:endCommand = ""
  elseif has("unix") || has("win32unix")
    let s:startCommand = "!"
    let s:endCommand = "&"
  else
    echo "Don't know how to handle this OS!"
    finish
  endif

  let s:cmd = "silent " . s:startCommand . " " . a:command . " " . s:endCommand
" echo s:cmd
  execute s:cmd
endfunction

function OnlineDoc()
  if &filetype == "wiki"

```

```

" Dictionary
  let s:urlTemplate = "http://dictionary.reference.com/browse/<name>"
elseif &filetype == "perl"
  let s:urlTemplate = "http://perldoc.perl.org/functions/<name>.html"
elseif &filetype == "python"
  let s:urlTemplate = "http://www.google.com/search?q=<name>&domains=docs.python.org&sitesearch=docs.python.org"
elseif &filetype == "ruby"
  let s:urlTemplate = "http://www.ruby-doc.org/core/classes/<name>.html"
elseif &filetype == "vim"
  let s:urlTemplate = "http://vimdoc.sourceforge.net/search.php?search=<name>&docs=help"
endif

let s:wordUnderCursor = expand("<cword>")
let s:url = substitute(s:urlTemplate, '<name>', s:wordUnderCursor, 'g')

call Run(Browser() . " " . s:url)
endfunction

noremap <silent> <M-d> :call OnlineDoc()<CR>
inoremap <silent> <M-d> <Esc>:call OnlineDoc()<CR>a

```

14.8 Zugriff auf Datenbanken

Mit dem `dbext.vim`⁸³-Plugin können Sie sogar auf etwa 10 verschiedene Datenbanken aus Vim heraus zugreifen, von Oracle über MySQL über PostgreSQL über Sybase bis SQLite. Das beste daran ist, dass Sie mit diesem Plugin SQL editieren können, geschrieben innerhalb von PHP, Perl, Java, etc., und Sie können sogar eine direkte Datenbankabfrage ausführen, obwohl es in einer anderen Programmiersprache eingebettet ist, und Sie sogar nach Variablenwerten fragt.

14.9 Zusammenfassung

Wir haben gelernt wie man Vim mit Hilfe verschiedener Plugins und Einstellungen zum Programmieren benutzen kann. Wenn wir eine Funktion brauchen, wird dies durch das Schreiben eigener Plugins bewerkstelligt (so wie im Skripte-Kapitel besprochen).

Eine gute Quelle für Diskussionen in diesem Zusammenhang gibt es auf *Stack Overflow*⁸⁴ und Peteris Krumins's Blog⁸⁵.

⁸³http://www.vim.org/scripts/script.php?script_id=356

⁸⁴<http://stackoverflow.com/questions/tagged/vim>

⁸⁵<http://www.catonmat.net/tag/vim>

15 Mehr

15.1 Einführung

Wir haben so viele Funktionen des Vim erforscht, aber wir haben immer noch nicht alle abgedeckt, also lassen Sie uns einen wilden Ritt durch die verschiedenen Themen machen, die nützlich und lustig sind.

15.2 Moduszeile

Was, wenn Sie festlegen möchten, dass eine bestimmte Datei, während des editierens, immer reine *Tabs* und keine Leerzeichen benutzen soll? Können wir dies innerhalb der Datei erzwingen?

Ja! Legen Sie einfach ein `vim: noexpandtab` innerhalb der ersten oder letzten beiden Zeilen Ihrer Datei ab.

Ein Beispiel:

```
# Sample Makefile
.cpp:
    $(CXX) $(CXXFLAGS) $< -o $@

# vim: noexpandtab
```

Diese hinzugefügte Zeile nennt sich „Moduszeile“.

15.3 Vim für unterwegs

Wenn Sie öfters an ganz unterschiedlichen Computern sitzen, ist es dann nicht eine Qual, wenn Sie Ihre Vim-Einstellungen für jeden Rechner immer wieder auf den gleichen Stand bringen müssen? Wär's nicht sinnvoll, wenn Sie den Vim auf Ihrem eigenen USB-Stick (*→ disk deliberately changed to stick!*) mit sich herumtragen könnten? Genau dafür gibt es *Portable GVim*⁸⁶.

Einfach in ein Verzeichnis auf dem USB-Stick entpacken, und `GVimPortable.exe` ausführen. Sie können sogar Ihre eingene `vimrc` und andere damit in Zusammenhang stehende Dateien auf dem Stick abspeichern, und überall dort, wo ein Microsoft Windows Computer steht, benutzen.

15.4 Plugins aufrüsten (→upgrade)

Jeder ausreichend fortgeschrittene Vim-Anwender würde einen ganzen Haufen von **Plugins** und Skripten verwenden, die er in seinem `~/.vim` oder `~/vimfiles`-Verzeichnis ablegt. Was, wenn wir die alle aufrüsten wollen, auf die letzte Version? Sie könnten natürlich für jedes einzelne die Skriptseite aufrufen, herunterladen, und sie installieren. Aber es

⁸⁶<http://portablegvim.sourceforge.net>

gibt eine bessere Methode – Führen Sie einfach **:GLVS** aus (das steht für 'G'et 'L'atest 'V'im 'S'cripts).

:help getscripts gibt Ihnen die Details.

Es gibt Skripte mit denen Sie sogar von Vim *twittern* können⁸⁷ !

15.5 Dr. Chip's Plugins

„Dr. Chip“ hat ein paar ganz erstaunliche **Plugins**⁸⁸, über die Jahre, geschrieben. Meine Lieblingsskripte sind **drawit.vim**, womit Sie textbasierte Zeichnungen erstellen können, wie die ganzen ASCII-Diagramme, die Sie schon zuvor gesehen haben.

Ein anderes Lieblingsskript ist **Align.vim**⁸⁹, mit dem Sie aufeinanderfolgende Zeilen bündig ausrichten können. Nehmen wir mal dieses Stück Programmiercode:

```
a = 1
bbbb = 2
cccccccc = 3
```

Markieren Sie einfach die drei Zeilen visuell (**v** *im Normalmodus*), und drücken Sie dann **\t=**, und – tataa! – es ändert sich zu dem hier:

```
a          = 1
bbbb       = 2
cccccccc   = 3
```

Das ist viel leichter zu lesen, als der Ausgangstext, und Ihr Programmcode sieht dadurch professioneller aus.

Gehen Sie auf Dr. Chips Seite da gibt es noch jede Menge interessanter **Plugins**.

15.6 Aus Vim heraus bloggen

(→Blog oder Weblog = World Wide Web Log aus: www.wikipedia.de)

Wenn Sie das **Vimpress-Plugin**⁹⁰ verwenden, können Sie Ihren *Wordpress*-Blog direkt mit dem Vim erledigen.

15.7 Firefox soll sich wie Vim verhalten

Mit dem **Vimperator**-Zusatz**plugin**⁹¹ können Sie Ihren Firefox vim-ähnliches Verhalten beibringen, komplett mit allen Modi, Tastaturkürzel um Verknüpfungen anzusteuern, Statuszeile, Tab-Ergängung und sogar Markierungsunterstützung!

15.8 Bram's Gesprächsrunde über die sieben Angewohnheiten

Bram Moolenaar, der den Vim programmiert hat, hat vor langer Zeit einen Artikel geschrieben mit dem Titel „*Seven habits of effective text editing*“⁹² (→'Sieben Ange-

⁸⁷http://www.vim.org/scripts/script.php?script_id=1853

⁸⁸<http://mysite.verizon.net/astronaut/vim/>

⁸⁹http://www.vim.org/scripts/script.php?script_id=294

⁹⁰http://www.vim.org/scripts/script.php?script_id=1953

⁹¹<http://vimperator.mozdev.org/>

⁹²<http://www.moolenaar.net/habits.html>

wohnheiten für das effektive Editieren von Textdateien' → <http://aaron-mueller.de/artikel/seven-habits-german>), der erklärt, wie man einen guten Editor (wie den Vim) benutzt.

Bram hat kürzlich einen Vortrag mit dem Titel „Seven habits of effective text editing 2.0“⁹³ wo er sowohl die neueren Funktionen des Vim beschreibt, als auch wie man den Vim effektiv benutzt. Das ist ein guter Vortrag für jeden normalen Vim-Anwender.

15.9 Was beisteuern zu Vim

Sie können auf vielerlei Art etwas zu Vim beisteuern, indem Sie z.B. an der Entwicklung selbst mitarbeiten⁹⁴, Plugins und Farbschemata schreiben⁹⁵, Tipps beisteuern⁹⁶ und bei der Dokumentation helfen⁹⁷.

Wenn Sie bei der Entwicklung des Vim mitarbeiten möchten - `:help development`.

15.10 Die Gemeinschaft (→community)

Viele Vim-Anwender hängen in der vim@vim.org-Mailingliste rum⁹⁸ wo Fragen und Zweifel nachgefragt, und beantwortet werden. Die beste Art, mehr über Vim zu lernen, und anderen Anfängern den Vim beizubringen, ist es, sich immer wieder Emails aus der Mailingliste durchzulesen (und darauf zu antworten).

Auf *delicious*⁹⁹ und *reddit*¹⁰⁰ finden Sie weitere Artikel und Besprechungen.

15.11 Zusammenfassung

Wir haben einen großen Bereich Vim-bezogenen Stoffs gesehen, und wie wir davon profitieren. Fühlen Sie sich ermutigt diese, und viele weitere Skripte zu erkunden¹⁰¹ um Ihnen das Editieren leichter und bequemer zu machen.

16 Ausblick

16.1 Einführung

Wir haben so vieles vom Vim gesehen, was kommt also als nächstes?

Nun – wenn Sie gelernt, verstanden haben, und sich Vim zur *Gewohnheit*, gemacht haben, dann sind Sie jetzt ganz offiziell ein *Vimmer*. Herzlichen Glückwunsch!

Schicken Sie mir also sofort eine Mail¹⁰² und bedanken Sie sich bei mir für dieses Buch ;-). Das ist freibleibend aber empfohlen.

⁹³<http://video.google.com/videoplay?docid=2538831956647446078&q=%22Google+engEDU%22>

⁹⁴http://groups.google.com/group/vim_dev

⁹⁵<http://www.vim.org/scripts/>

⁹⁶<http://vim.wikia.com>

⁹⁷<http://vimdoc.sourceforge.net>

⁹⁸<http://www.vim.org/maillist.php#vim>

⁹⁹<http://delicious.com/popular/vim>

¹⁰⁰<http://www.reddit.com/r/vim/>

¹⁰¹<http://www.vim.org/scripts/>

¹⁰²<http://www.swaroopch.com/contact/>

Denken Sie auch noch einmal über eine Zuwendung¹⁰³ nach, um die fortlaufende Entwicklung dieses Buches zu unterstützen, oder spenden Sie dem Vim-Projekt, um Kindern in Uganda zu helfen.

Als nächstes empfehle ich, der Vim-Mailingliste¹⁰⁴ zu folgen, um die Art von Anfragen und Antworten zu sehen, die dort auftauchen. Sie werden überrascht sein über die Menge an Aktivität und die Bandbreite der Anpassungsfähigkeit des Vim, die bei all den Diskussionen demonstriert werden. Und – wer weiß – vielleicht können Sie auch selbst ein paar Anfragen beantworten.

Zwei weitere wichtige Quellen, die zu erforschen es Wert sind, ist die „*Best of Vim*“ Tips-Seite¹⁰⁵ und das hochwertige `:help user-manual`¹⁰⁶, was ein Register für alles erdenkliche in Vim ist. Stellen Sie sicher, dass Sie hin und wieder ins *user-manual* schauen, um damit vertraut zu werden, wie Sie die relevanten Informationen finden, um Befehle nachzuschlagen, wenn es nötig ist.

Zu guter Letzt wenn Sie die neuesten *cool* Funktionen des Vim kennenlernen wollen, dann schauen Sie sich `:help new-7` an.

16.2 Zusammenfassung

Wir sind durch eine sehr breit angelegte Übersicht des Möglichkeitenbereiches von Vim gegangen. Die wichtigste Sache, die wir immer und immer wieder betont haben, ist, nicht jede Funktion auswendig zu lernen, sondern sich die, für Sie nützlichsten Funktionen, zur Angewohnheit zu machen, und die Hilfe (`:help`) weidlich zu nutzen, um Funktionen zu suchen, wenn sie gebraucht werden

Also – schreiten Sie voran, und schießen Sie mit Ihren Editierfähigkeiten himmelwärts, um effizienter zu werden, als Sie es je für möglich gehalten hätten.

Glückliches *Vimmen*!

17 Resonanz

Dieses Buch ist ein *Buch-in-Entwicklung*. Ihre Resonanz ist entscheidend für die Verbesserung des Buches. Schicken Sie mir also Ihre Kommentare¹⁰⁷.

Dieses Buch steht auch auf seiner offiziellen Website als Wiki¹⁰⁸ zur Verfügung. Das bedeutet also, dass Sie Korrekturen auch selbst vornehmen können!!

Bekannte Lücken sind:

- uneinheitliche Verwendung von Beispielen.
- die gegenwärtige Stärke sind die letzten Kapitel für erfahrene Vimmer im Gegensatz zu den Anfängerkapiteln.

¹⁰³<http://www.swaroopch.com/byteofdonate>

¹⁰⁴<http://www.vim.org/maillist.php#vim>

¹⁰⁵<http://rayninfo.co.uk/vimtips.html>

¹⁰⁶http://vimdoc.sourceforge.net/html/doc/usr_toc.html

¹⁰⁷<http://www.swaroopch.com/contact/>

¹⁰⁸<http://www.swaroopch.com/notes/Vim>

18 Wohltätigkeits-Software (→charityware)

Vim ist „charityware“. Wenn Sie Vim nützlich finden, fühlen Sie sich aufgerufen der ICCF Holland-Foundation¹⁰⁹ zu helfen, in welcher Art auch immer.

Von ihrer Website: Der Süden Ugandas leidet unter der größten HIV-Infektionsrate der Welt. Eltern sterben an AIDS, gerade wenn ihre Kinder sie am dringendsten brauchen. Eine erweiterte Familie kann ihr neues Zuhause sein. Glücklicherweise gibt es in diesem Landwirtschaftsdistrikt genug Lebensmittel. Aber wer bezahlt ihre Schulgebühren, stellt medizinische Hilfe zur Verfügung, und hilft ihnen aufzuwachsen? Da hilft der ICCF Holland in der Hoffnung, dass sie sich auf lange Sicht, sich und ihren Kindern selbst helfen können.

Einzelheiten über den ICCF können im Vim gefunden werden mit `:help iccf`.

Um bei der Entwicklung des Vim selbst einen Beitrag zu leisten, können Sie die Vim-Entwicklung¹¹⁰ mit einer Geldspende fördern und ein eingetragener Vim-Anwender werden. Damit haben Sie auch Stimmrecht, über die neuen Funktionen, die dem Vim hinzugefügt werden!

19 Schlußschrift

19.1 Über dieses Buch

Dies ist eine Alpha-Ausgabe des Buches. Dies bedeutet, es könnte unvollständige Abschnitte, offensichtliche Fehler, eklatantes Fehlen von Einzelheiten, und mehr haben. Konstruktive Vorschläge sind also immer willkommen¹¹¹.

19.2 Erschaffung dieses Buches

Dieses Buch wäre ohne Vim 7 niemals geschrieben worden. Mein Leben wäre ohne Vim unvollständig. Danke Bram Moolenaar und *Vimmer* der ganzen Welt.



Der Originalinhalt wurde in Viki-Syntax geschrieben, und dann mit dem `deplate`-Befehl¹¹² zusammengeschnürt. Ganz besonderen Dank an Thomas Link¹¹³ für die geduldige Beantwortung all meiner Anfragen bezüglich *Viki/Deplate* ;-)

Das Buch wurde später in ein Wiki-Format überführt. Obwohl das Buch jetzt sogar übers Internet bearbeitet wird, nehme ich den Vim noch immer um ins MediaWiki zu schreiben und zu *kopieren/einfügen* und den Text zu speichern.

¹⁰⁹<http://iccf-holland.org>

¹¹⁰<http://www.vim.org/sponsor/faq.php>

¹¹¹<http://www.swaroopch.com/contact/>

¹¹²<http://deplate.sourceforge.net>

¹¹³http://www.vim.org/account/profile.php?user_id=4037

19.3 Inspiration

Ich habe damit angefangen dieses Buch so um 2004 zu schreiben. Es schien unausweichlich, dass ich es hintan stellen und vernachlässigen würde. Einen Monat nach der *foss.in*¹¹⁴ /2006 hab ich die Sache aber wiederbelebt, des infektiösen Geistes der Leute dort sei Dank.

Aber wieder einmal konnte ich den Schwung nicht mitnehmen. Ich wurde produktiver nachdem ich mir die *Dinge-erledigt-kriegen-Prinzipien* von David Allen reingezogen hab¹¹⁵. Das hat dann zur Wiederbelebung an der Arbeit des Buches geführt, um es letztlich in eine veröffentlichungstaugliche Form zu bringen, und am ersten Tag der *foss.in* /2008, wurde es der Welt dann vorgestellt.

19.4 Über den Autor



Swaroop C H ist 26 Jahre alt. Er hat ein Diplom in B.E. (Computerwissenschaft) aus PESIT, Bangalore, Indien. Momentan arbeitet er in seinem mit-gegründeten Unternehmen IONLAB¹¹⁶. Früher hat er bei Yahoo! und Adobe gearbeitet. Seine Leidenschaften sind Lesen/Schreiben/Laufen/Radfahren und ein bisschen programmieren.

Mehr über ihn auf <http://www.swaroopch.com/about/>

Kontakt kriegt man mit ihm übers Internet – <http://www.swaroopch.com/contact>

20 Übersetzungen

- Wenn Sie lernen wollen wie Sie eine Übersetzung erstellen, sehen Sie sich bitte die Übersetzungsanleitung aus 'A Byte of Python' an.
- Wenn Sie schon dabei sind eine Übersetzung zu schreiben, stellen Sie sich bitte mit einer Einführung vor, und tragen Sie auch die Übersetzungssprache hier auf dieser Seite ein, so ähnlich wie auf der Liste zu 'A Byte of Python'-Übersetzungen.

20.1 Schwedisch

Mikael Jacobsson (leochingwake-at-gmail-dot-com) hat sich bereit erklärt das Buch ins Schwedische zu übersetzen, und die Übersetzung macht Fortschritte auf http://leochingwake.se/wiki/index.php/Byte_of_Vim.

¹¹⁴<http://www.foss.in>

¹¹⁵<http://www.swaroopch.com/gtdbook>

¹¹⁶<http://www.ionlab.in>

20.2 Traditionelles Chinesisch

Yeh, Shin-You (oder Yesyo) hat sich bereit erklärt das Buch in traditionelles Chinesisch zu übersetzen. Die Übersetzung macht Fortschritte und beginnt mit dem Kapitel „Vim zh-tw“.

20.3 Russisch

Vitalij Naumov (oder hbvit7) hat sich bereit erklärt, das Buch ins Russische zu übersetzen. Die Übersetzung macht Fortschritte, und beginnt mit dem Kapitel „Vim.ru“.

20.4 Deutsch

Christoph Koydl hat sich selbstverständlich auch bereit erklärt, das Buch zu übersetzen, und zwar ins Deutsche. Die Rohfassung der Übersetzung ist abgeschlossen, es fehlen lediglich noch eine Überarbeitung. Als da wären..

- Besprechung von Verständnisschwierigkeiten
- Korrekturlesen / Rechtschreibprüfung
- letzte Absprachen mit Swaroop C H
- Danksagung

Abzurufen gibt es eine PDF-Version der deutschen Übersetzung auf <http://koydl.in-berlin.de>

21 Glossar

Blog	Internettagebuch (→world wide we'b' 'log')
Browser	Internetseitenanzeigeprogramm
Cursor	(Text)-Schreibmarke
Email	elektronische Post
Link	Verknüpfung
Omnicompletion	Allesvervollständigung
Plugin	Programmerweiterungsmodul
Share Alike	„gleiches Verteilen“→Verteilen unter gleichen Bedingungen
Shell	Kommandozeileninterpreter (Schnittstelle zwischen Anwender und Betriebssystemkern)
Syntax	Satzbaulehre

Tab	Registerkartenelement oder Reiter
Tag	Markierung
To twitter	zwitschern