Sieben Angewohnheiten für das effektive Editieren von Textdateien

Bram Moolenaar
Übersetzt von Aaron Müller

9. März 2006
Korekturen am 21. April 2008 und 2. April 2014

Abstract

Wenn Sie viel Zeit mit dem Eingeben von Text verbringen, Programmieren oder HTML-Seiten erstellen, können Sie einiges an Zeit sparen, wenn Sie einen guten Editor verwenden und diesen effektiv nutzen. Dieses Dokument zeigt Anleitungen und Hinweise, um Ihre Arbeit noch schneller und mit weniger Fehlern bewältigen zu können.

Der Open-Source Texteditor Vim (Vi IMproved) wird hier verwendet, um Ideen des effektiven Texteditierens zu zeigen, allerdings funktionieren diese Arbeitsweisen mit anderen Text-Editoren genauso gut. Den richtigen Editor zu wählen ist eigentlich der erste Schritt zum effektiven Editieren. Die Diskussion, welcher Editor der Beste für Sie ist, würde hier zu weit gehen und wurde hier somit außer Acht gelassen. Wenn Sie nicht wissen, welchen Editor Sie verwenden sollen, oder mit Ihrem jetzigen unzufrieden sind, probieren Sie Vim einmal aus; Sie werden es nicht bereuen.

Contents

1 Eine Datei bearbeiten

1.1 Schnelles Bewegen im Text

Man verbringt mehr Zeit mit lesen, dem Suchen nach Fehlern, und damit, die richtige Stelle zum Arbeiten suchen, als mit dem Eingeben von neuem Text oder editieren. Man navigiert sehr oft durch den Text, deshalb sollten Sie lernen, wie man dies so schnell wie möglich macht.

Sehr oft wollen Sie nach einer Textstelle suchen, von der Sie wissen, dass sie in der Datei zu finden ist. Oder Sie suchen alle Zeilen, die ein bestimmtes Wort oder eine bestimmte Textpassage enthält. Sie können natürlich einfach das Suchkommando texttt/pattern verwenden, um die Textstelle zu finden, aber es gibt noch intelligentere Wege:

- Wenn Sie ein bestimmtes Wort sehen und nach weiteren Vorkommen des selben Wortes suchen, benutzen Sie den Befehl `*`. Dieser Befehl verwendet das aktuelle Wort unter dem Cursor und sucht nach dem nächsten Vorkommen im Text.

- Wenn Sie die Option '`incsearch`' einstellen, zeigt Vim das Suchergebnis ,,OnTheFly" an, wärend Sie den Suchbegriff tippen. Das vermeidet schnell Tippfehler.

- Wenn Sie die Option '`hlsearch`' einstellen, färbt Vim alle gefundenen Textpassagen mit gelbem Hintergrund ein. Dies gibt einen schnellen überblick über alle gefundenen Stellen. In einem Programmcode kann man damit alle Stellen, an denen eine Variable verwendet wurde, auf einen Blick sehen. Sie müssen dazu nicht einmal den Cursor bewegen.

In strukturiertem Text gibt es noch viel mehr Möglichkeiten, sich schnell im Text zu bewegen. Vim hat spezielle Befehle für C-Programme (und ähnliche Sprachen wie C++ und Java):

- Verwenden sie `%`, um von einer geöffneten geschwungenen Klammer zur korrespondierenden geschlossenen Klammer zu springen. Oder von einem ,,#if" zum entsprechenden ,,#endif". Mit % können Sie zwischen vielen verschiedenen korrespondierenden Zeichen springen. Dies ist sehr hilfreich, um zu überprüfen, ob die Klammern bei einem `if()` bzw. { } richtig gesetzt wurden.

- Verwenden Sie `[{` um zurück zur öffnenden geschwungenen Klammer und {" um an den Anfang des momentanen Abschnitts zu springen.

- Verwenden Sie `gd` um von der Benutzung der Variablen zur Deklaration zu springen.

Es gibt natürlich noch viel mehr. Der Punkt ist, dass Sie diese Befehle kennen sollten. Sie mögen nun einwenden, dass Sie nicht alle Befehle auswendiglernen können - es gibt hunderte von verschiedenen Bewegungs-Befehlen, ein paar einfache, ein paar raffinierte - und man müsste wochenlang trainieren, um all diese Befehle zu beherrschen. Deshalb brauchen Sie diese Befehle nicht lernen; stattdessen ist es hilfreich, wenn Sie sich an eine spezielle Art und Weise des

Editieres gewöhnen und nur die Befehle lernen, mit denen Sie effektiver arbeiten können.

Hier nun *drei einfache Schritte*:

1. Achten Sie wärend des Editierens auf Abläufe, die sich wiederholen oder ziemlich viel Zeit beanspruchen.

2. Suchen Sie nach einem Editor-Befehl, der diese Arbeiten schneller erledigt. Lesen Sie die Dokumentation, fragen Sie einen Freund oder schauen Sie anderen über die Schulter.

3. Trainieren Sie die Benutzung dieses Befehls. Tun Sie dies, bis Ihre Finger automatisch agieren.

Verwenden wir ein Beispiel, um zu zeigen, wie es funktioniert:

1. Sie merken, wenn Sie ein C-Programm bearbeiten, dass Sie oft Zeit damit vergeuden, um nach der Funktionsdeklaration zu suchen. Jetzt verwenden Sie den `*` Befehl um nach anderen Stellen, an denen der Funktionsname auftritt, zu suchen, aber am Ende gehen Sie über viele Treffer, welche die Benutzung des Funktionsnamens anzeigen anstelle der Deklaration. Sie kommen auf die Idee, das es doch einen anderen Weg geben muss, um dies zu beschleunigen.

2. Beim Durchsehen der Schnellreferenz finden Sie eine Bemerkung über das Springen zu Markierungen. Die Dokumentation zeigt, wie man dies zum Springen an eine Funktionsdefinition verwenden kann, genau das was wir gesucht haben!

3. Sie experimentieren ein wenig mit dem Erstellen einer Markierungs-Datei (engl.: tags file) mit dem Programm `ctags`, welches mit Vim mitgeliefert wird. Sie lernen, wie man den `CTRL-]` Befehl verwendet, und bemerken, dass Sie damit eine Menge Zeit sparen. Um es noch einfacher zu machen, fügen Sie ein paar Zeilen in Ihr Makefile um die Markierungsdatei automatisch zu erstellen.

Ein paar Dinge, die Sie beachten sollten, wenn Sie diese drei Schritte anwenden:

- „Ich will mit meiner Arbeit fertig werden, ich habe keine Zeit um die Dokumentation nach ein paar neuen Befehlen zu durchsuchen“. Wenn Sie so denken, werden Sie in der Computersteinzeit hängen bleiben. Manche Menschen verwenden den Windows-Editor (engl. Notepad) für alles, und wundern sich dann, wenn andere nur die Hälfte der Zeit benötigen ...

- übertreiben Sie es nicht. Wenn Sie immer versuchen, den perfekten Befehl für alles was Sie tun zu finden, bleibt Ihrem Gedächtnis keine Zeit mehr über die Arbeit, die Sie momentan machen, nachzudenken. Wählen Sie nur diese Aktionen aus, die mehr Zeit als nötig verbrauchen, und trainieren Sie diese Befehle so lange, bis Sie nicht mehr über diese nachdenken müssen. Dann können Sie sich auf den Text konzentrieren.

Im folgenden Abschnitt werden Vorschläge für Aktionen aufgezeigt, mit welchen die meisten Menschen arbeiten. Sie können diese als Inspiration für die Benutzung der *drei einfachen Schritte* in ihrer eigenen Arbeit verwenden.

1.2 Tippen Sie nichts zweimal

Es gibt eine eingeschränkte Anzahl Wörter, die wir tippen. Und eine ebenso limitierte Anzahl von Textteilen und Sätzen, speziell beim Schreiben von Computer-Programmen. Offensichtlich ist, das Sie etwas nicht zweimal tippen wollen.

Es kommt oft vor, dass Sie ein Wort in ein anderes umschreiben möchten. Wenn dies über die gesammte Datei erfolgen soll, können Sie den Befehl `:s` [1]. Wenn nur an ein paar Stellen Änderungsbedarf besteht, hat man mit dem Befehl `*` die Möglichkeit, schnell das nächste Vorkommen des Wortes zu finden, und mit `cw`[2] das Wort zu ändern. Tippen Sie dann `n`, um das nächste Wort zu finden und ein `.` (Punkt), um den `cw`-Befehl zu wiederholen.

Der `.`-Befehl wiederholt die letzte Änderung. Eine Änderung in diesem Zusammenhang ist Text einfügen, löschen oder ersetzen.

Seien Sie sich über den Wiederholen-Befehl bewusst, es ist ein sehr mächtiger Mechanismus. Wenn Sie die Bearbeitung ihrer Texte darauf auslegen, werden Sie feststellen, das viele Änderungen Anlass dafür geben, einfach die .-Taste zu drücken. Achten Sie auf die Änderungen dazwischen, denn es werden die Änderungen ersetzt, die Sie wiederholen. Stattdessen könnten Sie diese Stelle mit dem `m`-Befehl[3] markieren, Ihre wiederholenden Änderungen durchführen und später wieder an diese Stelle zurückspringen.

Manche Funktions- und Variablen-Namen können sehr umständlich oder störrisch zu schreiben sein. Können Sie schnell "XpmCreatePixmapFromData" tippen, ohne einen Tippfehler zu machen und ohne ihn nachzuschlagen? Vim hat einen Vervollständigungsmechanismus, der dies viel einfacher macht. Es wird in der Datei, die Sie bearbeiten, und ebenso in den mit "include" eingebundenen Dateien nach Wörtern gesucht. Sie können "XpmCr" eintippen, gefolgt von der Tastenkombination `CTRL-N`, damit Vim die Eingabe für Sie zu "XpmCreatePixmapFromData" erweitert. Nicht nur, dass dies etwas Tipparbeit spart, auch verhindert es Tippfehler, welche Sie später, wenn der Compiler eine Fehlermeldung ausgibt, korrigieren müssen.

Wenn Sie eine Wortverbindung oder Sätze mehr als einmal tippen, gibt es auch hier einen schnelleren Weg. Vim hat einen Mechanismus, um ein Makro aufzunehmen. Tippen Sie `qa`, um ein Makro im Register "a" zu starten. Tippen Sie nun normal Ihre Befehle und Tippen Sie abschließend "q" erneut, um die Aufnahme zu stoppen. Wenn Sie die aufgenommenen Befehle wiederholen wollen, tippen Sie `@a`. Es gibt dafür 26 Register.

Mit den Aufnahmen (Makros) können Sie viele verschiedene Aktionen wiederholen, nicht nur Text einfügen. Denken Sie daran, wenn Sie wissen, dass Sie irgend etwas wiederholen wollen.

Eine Sache muss beim Aufnehmen beachtet werden: Die Befehle werden genau so abgespielt, wie sie eingegeben werden. Denken Sie beim Bewegen der Eingabemarke daran, dass der Text, den Sie vor sich haben anders sein könnte, wenn der Befehl wiederholt wird. Vier Zeichen nach links bewegen funktioniert vielleicht bei dem Text, in dem Sie gerade aufnehmen, aber es müssen vielleicht fünf Zeichen sein, wenn Sie den Befehl an einer anderen Stelle wiederholen. Es

[1] substitute, dt. ersetzen
[2] change word, dt. Wort ändern
[3] mark, dt. markieren

ist oft notwendig, Befehle zu nutzen, die über Text-Objekte (Wörter, Sätze) oder zu einem speziellen Zeichen springen.

Wenn die Befehle, die Sie für das Wiederholen benötigen, komplizierter werden, wird es noch schwerer, wenn Sie alles auf einmal eintippen. Anstatt diese aufzunehmen, sollten Sie ein Script oder ein Makro schreiben. Dies ist sehr hilfreich, um Vorlagen für Teile ihres Programmcodes zu erstellen; zum Beispiel einen Funktionskopf. Dies können Sie so intelligent lösen, wie Sie möchten.

1.3 Korrigieren Sie es, wenn es falsch ist

Tippfehler sind normal. Niemand kann sie verhindern. Der Trick ist, sie schnell zu erkennen und zu korrigieren. Der Editor sollte in der Lage sein, Ihnen dabei zu helfen. Allerdings müssen Sie ihm mitteilen, was falsch und was richtig ist.

Es kommt sehr oft vor, dass Sie immer und immer wieder den selben Fehler machen. Ihre Finger tun einfach nicht das, was ihnen aufgetragen wurde. Dies kann mit Abkürzungen behoben werden. Ein paar Beispiele:

```
:abbr Lunix Linux
:abbr accross across
:abbr hte the
```

Die Wörter werden gleich nach dem Eingeben automatisch korrigiert.

Der gleiche Mechanismus kann verwendet werden, um ein langes Wort mit nur ein paar Zeichen einzugeben. Dies ist vor allem für Wörter nützlich, die für Sie schwer zu tippen sind. Dies vermeidet, dass Sie diese falsch eingeben. Beispiele:

```
:abbr pn penguin
:abbr MS Mandrake Software
```

Wie auch immer, dies führt dazu, das kurze Eingaben zu einem ganzen Wort erweitert werden, auch wenn Sie es nicht möchten. Dies macht es schwer, wenn Sie wirklich “MS” in Ihrem Text einfügen möchten. Es ist das Beste, wenn kurze Wörter verwendet werden, die keine eigene Bedeutung haben.

Um Fehler in Ihrem Text zu finden, bietet Vim einen cleveren Hervorhebungs-Mechanismus an. Dieser wurde eigentlich dafür gemacht, um Quellcode farbig darzustellen (Syntax-Highlighting), aber es kann auch Fehler erkennen und hervorheben.

Syntax-Highlighting zeigt Kommentare in einer Farbe. Das klingt nicht nach einem wichtigen Feature, aber wenn Sie angefangen haben es zu nutzen, werden Sie feststellen, dass es sehr hilfreich ist. Sie können so schnell Text erkennen, welcher ein Kommentar sein sollte, aber nicht als ein solcher hervorgehoben ist (Sie haben wahrscheinlich das Kommentarzeichen vergessen). Oder Sie sehen eine als Kommentar markierte Code-Zeile (Sie haben ein “*/” vergessen). Das sind Fehler, die in einem Schwarz/Weiß-Editor schwer zu erkennen sind und eine Menge Zeit mit dem Finden des Fehlers verschwenden.

Das Syntax-Highlighting kann auch auf unausgeglichene Klammernpaare hinweisen. Eine ungeöffnete “)” wird mit einem hellroten Hintergrund hervorgehoben. Sie können mit dem `%`-Befehl sehen, an welcher Stelle sich das Gegenstück befindet (befinden müsste), und eine “(” oder eine “)” an der richtigen Stelle einfügen.

Andere übliche Fehler sind auch schnell erkannt, wenn man zum Beispiel statt "#include <stdio.h>", "#included <stdio.h>" geschrieben hat. Sie übersehen einfach die Fehler bei einer Schwarz/Weiß-Darstellung, aber erkennen schnell, das "include" hervorgehoben wird, aber "included" nicht.

Ein etwas komplexeres Beispiel: Für englischen Text gibt es eine Liste mit Wörtern, die benutzt werden. Jedes Wort, das nicht auf der Liste steht, könnte ein Fehler sein. Mit einer Syntax-Datei können Sie jedes Wort hervorheben, welches nicht auf der Liste steht. Mit ein paar zusätzlichen Makros können Sie weitere Wörter zur Wortliste hinzufügen, damit sie nicht länger als Fehler markiert werden. Dies funktioniert genau so, wie Sie es in einem Textverarbeitungsprogramm erwarten würden. In Vim wurde diese Funktionalität mit Skripten eingebaut und Sie können diese an ihre eigenen Bedürfnisse anpassen: Zum Beispiel, um nur die Kommentare in einem Programm auf Rechtschreibfehler zu prüfen.

2 Mehrere Dateien bearbeiten

2.1 Eine Datei kommt selten allein

Menschen arbeiten nicht nur an einer Datei. Oft gibt es viele zusammenhängende Dateien, die nacheinander, oder aber mehrere zur gleichen Zeit bearbeitet werden. Sie sollten in der Lage sein, einen Vorteil aus Ihrem Editor zu ziehen, um effektiver mit mehreren Dateien zu arbeiten.

Der oben erwähnte Tag-Mechanismus funktioniert auch, um zwischen Dateien zu springen. Der übliche Anfang ist eine "tags"-Liste für das gesamte Projekt, an dem Sie gerade arbeiten, zu generieren. Dann können Sie schnell zwischen allen Dateien im Projekt hin- und herspringen, um zu Funktionsdefinitionen, Strukturen, Typdefinitionen usw. zu gelangen. Die Zeit, die Sie sparen, verglichen mit manuellem Suchen, ist erstaunlich; eine Tags-Datei zu erstellen ist das Erste, das ich tue, wenn ich einen Programmcode lese.

Ein weiterer mächtiger Mechanismus ist das Finden aller Vorkommnisse eines Namens in einer Gruppe von Dateien. Der Befehl dazu ist `:grep`. Vim macht eine Liste mit allen Treffern und springt zum ersten. Der `:cn` Befehl bringt Sie zum jeweils nächsten Treffer. Dies ist sehr nützlich, wenn Sie die Anzahl der Argumente in einem Funktionsaufruf verändern wollen.

Include-Dateien enthalten wichtige Informationen. Aber die eine zu finden, welche die Deklaration enthält, die sie sehen wollen, kann eine Menge Zeit beanspruchen. Vim kennt Include-Dateien und kann diese nach Wörtern durchsuchen, die Sie suchen. Die meistgenutzte Aktion ist das Betrachten des Prototyps einer Funktion. Positionieren Sie den Cursor auf den Namen der Funktion in Ihrer Datei, Tippen Sie `[I:` und Vim zeigt eine Liste aller Treffer für den Funktionsnamen in den eingebundenen Dateien (Header-Dateien) an. Wenn Sie mehrere Informationen benötigen, können Sie direkt zur Deklaration springen. Ein ähnlicher Befehl kann verwendet werden, um zu überprüfen, ob Sie die richtigen Header-Dateien eingebunden haben.

In Vim können Sie den Textbereich in mehrere Abschnitte teilen, um verschiedene Dateien zu bearbeiten. Mit geteilten Fenstern können Sie den Inhalt von zwei oder mehreren Dateien vergleichen und Text zwischen den Fenstern kopieren und einfügen. Es gibt viele Befehle, um Fenster zu öffnen, sie zu

schließen, zwischen ihnen hin- und herspringen und vorübergehend Dateien zu verstecken usw. Noch einmal, Sie sollten die drei grundlegenden Schritte anwenden, um die Anzahl von Befehlen, die Sie lernen wollen, zu benutzen.

Es gibt noch mehr Anwendungsgebiete für geteilte Fenstern. Der Vorschau-Tag Mechanismus ist ein sehr gutes Beispiel. Dieser öffnet ein spezielles Vorschaufenster, wärend der Cursor in der Datei bleibt, in der Sie arbeiten. Der Text im Vorschaufenster zeigt zum Beispiel die Funktionsdeklaration des Funktionsnamens, welcher unter dem Cursor steht, an. Wenn Sie den Cursor auf einen anderen Namen bewegen und ihn dort für eine Sekunde lassen, zeigt das Vorschaufenster die Definition von diesem Namen an. Es könnte auch eine Struktur oder eine Funktion sein, welche in einer Include-Datei ihres Projekts deklariert ist.

2.2 Zusammenarbeit

Ein Editor ist für das Editieren von Texten gemacht. Ein Email-Programm, um Nachrichten zu verschicken und zu empfangen. Ein Betriebssystem ist für das Starten von Programmen gedacht. Jedes Programm hat seine eigene Aufgabe und sollte diese gut erfüllen. Die Stärke kommt durch die Zusammenarbeit der Programme.

Ein einfaches Beispiel: Markieren Sie einen Teil eines strukturierten Textes und sortieren Sie diesen: `!sort`. Der externe "sort"-Befehl wird verwendet, um den Text zu filtern. Einfach, oder? Die Sortierfunktionalität könnte in den Editor eingebaut werden. Werfen Sie einen Blick auf "man sort" und sehen Sie, wieviele Optionen zur Verfügung stehen. Ausserdem hat das Programm vermutlich einen eleganten Algorithmus, welcher die Sortierung übernimmt. Wollen Sie all dies in den Editor integrieren? Genauso andere Filter-Befehle? Der Editor würde riesig werden.

Es folgte immer dem Geist von Unix, welches getrennte Programme bietet, die ihre Arbeit sehr gut erledigen und mit anderen Programmen zusammenarbeiten, um eine größere Aufgabe zu erledigen. Unglücklicherweise arbeiten die meisten Editoren nicht so gut mit anderen Programmen zusammen - Sie können zum Beispiel den Email-Editor von Netscape nicht durch einen anderen ersetzen. Sie müssen sich mit einem schlechten Editor begnügen. Eine weitere Tendenz ist es, alle möglichen Funktionen in den Editor einzubinden; Emacs ist ein gutes Beispiel dafür, wie es enden kann. (Manche nennen es ein Betriebssystem, welches auch genutzt werden kann, um Texte zu editieren.)

Vim versucht andere Programme zu integrieren, aber dies ist ein Kampf. Gegenwärtig ist es möglich, Vim als Editor in MS-Developer Studio und Sniff zu ersetzen. Manche Email-Programme, wie beispielsweise Mutt, die einen externen Editor erlauben, können Vim als Editor nutzen. An einer Integration mit Sun Workshop wird zur Zeit gearbeitet. Generell sollte sich diese Philosophie in naher Zukunft ändern. Nur dann wird es ein System geben, das besser ist, als die Summe seiner Teilstücke.

2.3 Text ist strukturiert

Sie werden oft mit Texten arbeiten, die eine bestimmte Struktur aufweisen, die allerdings nicht für die verfügbaren Befehle ausgelegt sind. An dieser Stelle müssen Sie zurück zu den "Grundmauern" von Vim gehen und Ihre eigenen

Makros und Skripte erstellen, die mit diesem Text umgehen können. An dieser Stelle wird es etwas komplizierter.

Eine der einfacheren Dinge ist es, den Editieren-Compilieren-Fehlersuche-Zyklus zu beschleunigen. Vim hat den `:make` Befehl, welcher das Compilieren startet, die Fehler, die ausgegeben werden abfängt, und Sie zu der Stelle springen lässt, an der der Fehler aufgetreten ist, um die Probleme zu beheben. Wenn Sie einen anderen Compiler verwenden, werden die Fehlermeldungen nicht erkannt. Anstelle jetzt zurück zum alten "schreib es von Hand" System zu gehen, sollten Sie die `errorformat` Option anpassen. Diese Option sagt Vim, wie die Fehlermeldungen aussehen und wie man daraus den Dateinamen und die Zeilennummer ermittelt. Dies funktioniert mit den gcc-Fehlermeldungen, folglich sollte es auch möglich sein, es mit jedem anderen beliebigen Compiler kompatibel zu machen.

Manchmal sind Anpassungen für einen Typ von Dateien der Grund, ein paar Optionen zu setzen oder ein paar Makros zu schreiben. Zum Beispiel, wenn Sie Man-Pages lesen, können Sie ein Makro schreiben, welches das Wort unter dem Cursor einliest, den Puffer löscht und dann die Man-Page mit dem eingelesenen Wort als Titel im Puffer öffnet. Dies ist ein einfacher und effizienter Weg um Querverweise zu betrachten.

Wenn Sie die drei einfachen Schritte anwenden, können Sie effektiver mit jeder Art von strukturiertem Text arbeiten. Denken Sie nur an die Aktionen, die Sie mit der Datei machen wollen. Finden Sie die Editor-Befehle, die diese Aktionen veranlassen und fangen Sie an, diese Befehle zu nutzen. Es ist wirklich so einfach wie es sich anhört. Sie müssen es nur tun.

3 Schärfen Sie die Säge

3.1 Machen Sie es sich zur Gewohnheit

Autofahren lernen erfordert Anstrengung. Ist das ein Grund, weiterhin mit dem Fahrrad zu fahren? Nein! Sie sehen, dass Sie Zeit investieren müssen, um eine Fähigkeit zu erlernen. Dies ist beim Texteditieren nicht anders. Sie müssen neue Befehle lernen und diese zu einer Gewohnheit machen.

Auf der anderen Seite sollten Sie nicht versuchen, jeden Befehl auswendigzulernen, den ein Editor anbietet. Das wäre nur Zeitverschwendung. Die meisten Menschen müssen nur 20 bis 30 Prozent der Befehle für ihre Arbeit lernen. Allerdings sind es unterschiedliche Befehle für jeden einzelnen. Es erfordert ständiges lernen und werden sie aufmerksam, wenn Sie auf immer wiederkehrende Aufgaben stoßen. Diese lassen sich bestimmt automatisieren. Wenn Sie eine Aufgabe nur einmal erledigen müssen, und wissen, dass Sie diese später nie wieder benötigen, versuchen Sie nicht, diese zu optimieren. Aber Sie bemerken sicherlich, wenn Sie irgend etwas mehrmals innerhalb der letzten Stunde wiederholt haben. An dieser Stelle sollten Sie die Dokumentation aufsuchen, um nach einem schnelleren Weg zu suchen, oder schreiben Sie ein Makro, welches dieses für Sie erledigt. Wenn es eine größere Aufgabe ist, wie beispielsweise einen bestimmten Text mit Zeilennummern zu versehen, können Sie in Newsgroups oder im Internet suchen, ob irgend jemand vor Ihnen dieses Problem schon gelöst hat.

Der wichtigste Schritt ist der letzte. Sie können über eine wiederholende Aufgabe nachdenken, eine gute Lösung dafür finden und nach dem nächsten Wochenende haben Sie vergessen, wie Sie dieses Problem gelöst haben. So funktioniert es nicht. Sie müssen die Lösung solange wiederholen, bis Sie es ohne nachzudenken können. Nur dann erreichen Sie die Effektivität, welche Sie wünschen. Versuchen Sie nicht, zu viele Dinge auf einmal zu lernen. Aber mehrere Dinge auf einmal machen funktioniert auch gut. Für Tricks, welche Sie so selten verwenden, dass Ihre Finger dies nicht automatisch machen, sollten Sie diese aufschreiben und dort später nachsehen. Wie auch immer, wenn Sie das Ziel vor Augen behalten, finden Sie Wege, um das Editieren mehr und mehr effizient zu gestalten.

Ein letzter Hinweis zur Erinnerung, was passiert, wenn Menschen all das ignorieren: Ich sehe immer wieder Menschen, die den halben Tag hinter einer VDU[4] sitzen und auf den Bildschirm starren, dann mit zwei Fingern auf der Tastatur schreiben, dann wieder auf den Bildschirm schauen, usw. - und sich dann wundern, warum sie so müde sind... Schreiben Sie mit zehn Fingern! Es ist nicht nur schneller, sondern auch weniger ermüdend. Verwenden Sie ein Tipp-Trainer-Programm, jeden Tag eine Stunde. Nach ein paar Wochen können Sie das 10-Finger-System.

4 Epilog

Die Idee zu diesem Artikel kommt von dem erfolgreichen Buch “The 7 habits of highly effective people” von Stephen R. Covery. Ich empfehle jedem dieses Buch, der private und berufsbezogene Probleme lösen möchte (und wer will das nicht?). Manche von Ihnen behaupten, es kommt von dem Dilbert-Buch “Seven years of highly defective people” von Scott Adams (auch zu empfehlen!). Siehe http://www.iccf.nl/click1.html und gehen Sie auf “recommended books and CDs”.

5 Über den Autor

Bram Moolenaar ist der Hauptentwickler von Vim. Er schrieb den Hauptteil der Grundfunktionen von Vim und bestimmt welcher Code, der von vielen anderen Programmierern geschrieben wird, in Vim einfließt. Er promovierte an der Technischen Universität von Delft als Computer-Techniker. Jetzt arbeitet er hauptsächlich an Software, weiß aber immer noch mit dem Lötkolben umzugehen. Er ist der Gründer und Leiter von ICCF Holland, welche Waisenkindern in Uganda hilft. Er arbeitet als freier Mitarbeiter als Systemarchitekt, aber tatsächlich arbeitet er die meiste Zeit an Vim. Seine Email-Adresse ist Bram AT Moolenaar.net.

6 Anmerkung des Übersetzers

Den Orginaltext finden Sie unter http://www.moolenaar.net/habits.html. Meine Email-Adresse ist mail AT aaron-mueller.de. Korrekturlesung von Claudia Müller. Weitere Verbesserungen von Christian und Matthias Konrath.

[4] Visual Display Unit, Computerbildschirm