



Kostenloses eBook

LERNEN

vim

Free unaffiliated eBook created from
Stack Overflow contributors.

#vim

Inhaltsverzeichnis

Über.....	1
Kapitel 1: Erste Schritte mit vim.....	2
Bemerkungen.....	2
Versionen.....	3
Examples.....	4
Installation.....	4
Installation unter Linux / BSD.....	4
Arch- und Arch-basierte Distributionen.....	4
Debian- und Debian-basierte Distributionen.....	4
Gentoo- und Gentoo-basierte Distributionen.....	4
RedHat- und RedHat-basierte Distributionen.....	4
Fedora.....	4
Slackware- und Slackware-basierte Distributionen.....	5
OpenBSD- und OpenBSD-basierte Distributionen.....	5
FreeBSD- und FreeBSD-basierte Distributionen.....	5
Installation unter Mac OS X.....	5
Regelmäßige Installation.....	5
Paket-Manager.....	5
Installation unter Windows.....	6
Schokoladig.....	6
Vim von der Quelle aus bauen.....	6
Vim verlassen.....	6
Erläuterung:.....	7
Interaktive Vim-Tutorials (z. B. Vimtutor).....	7
Speichern einer schreibgeschützten Datei, die in Vim bearbeitet wurde.....	8
Befehlserklärung.....	8
Vim aussetzen.....	9
Grundlagen.....	9
Was tun im Falle eines Absturzes?.....	11
Kapitel 2: : global.....	13
Syntax.....	13

Bemerkungen.....	13
Examples.....	13
Grundlegende Verwendung des Global Command.....	13
Ziehe jede Zeile, die einem Muster entspricht.....	13
Zeilen mit wichtigen Informationen verschieben / sammeln.....	14
Kapitel 3: Auswechslung.....	15
Syntax.....	15
Parameter.....	15
Bemerkungen.....	15
Beispiel.....	15
Examples.....	15
Einfacher Austausch.....	15
Refactor das Wort unter dem Cursor schnell.....	16
Ersatz mit interaktiver Genehmigung.....	16
Tastenkombination zum Ersetzen des aktuell hervorgehobenen Wortes.....	16
Kapitel 4: Autobefehle.....	17
Bemerkungen.....	17
Examples.....	17
Quelle automatisch .vimrc nach dem Speichern.....	17
Aktualisieren Sie die vimdiff-Ansichten, wenn eine Datei in einem anderen Fenster gespeichert.....	18
Kapitel 5: Bauen von vim.....	19
Examples.....	19
Build starten.....	19
Kapitel 6: Befehle im Normalmodus.....	20
Syntax.....	20
Bemerkungen.....	20
Examples.....	20
Text sortieren.....	20
Normale Sortierung.....	20
Rückwärtssortierung.....	20
Groß- und Kleinschreibung wird nicht berücksichtigt.....	20

Numerische Sortierung	20
Duplikate nach dem Sortieren entfernen	20
Optionen kombinieren	21
Kapitel 7: Befehle im Normalmodus (Bearbeitung)	22
Examples	22
Einführung - Kurzer Hinweis zum Normalmodus	22
Grundlegendes Rückgängig machen und Wiederholen	22
Rückgängig machen	22
Wiederholen	22
Wiederholen Sie die letzte Änderung	23
Kopieren, Ausschneiden und Einfügen	23
Register	24
Bewegungen	24
Kopieren und Schneiden	24
Einfügen	25
Wie führe ich ein wirklich einfaches Ausschneiden und Einfügen aus?	25
Fertigstellung	26
Kapitel 8: Befehlszeilenbereiche	28
Examples	28
Absolute Zeilennummern	28
Relative Zeilennummern	28
Zeilenverknüpfungen	28
Markierungen	28
Suche	29
Linienversätze	29
Gemischte Bereiche	29
Kapitel 9: Bewegung	30
Examples	30
Suchen	30
Zu den Charakteren springen	30
Suche nach Zeichenketten	30
Grundlegende Bewegung	31

Bemerkungen	31
Pfeile	31
Grundbewegungen	31
Muster suchen	33
Zum Anfang eines bestimmten Wortes navigieren	34
Verwenden von Markierungen zum Bewegen	36
TLDR	36
Eine Marke setzen	36
Springe zu einer Marke	36
Globale Markierungen	37
Besondere Kennzeichen	37
Springe zu einer bestimmten Zeile	38
Kapitel 10: Bewegungen und Textobjekte	39
Bemerkungen	39
Examples	39
Ändern des Inhalts einer String- oder Parameterliste	39
Kapitel 11: Bitten Sie beim Erstellen einer neuen Datei um das Erstellen nicht vorhandener	40
Einführung	40
Examples	40
Fordern Sie auf, Verzeichnisse mit: w zu erstellen, oder erstellen Sie sie mit: w!	40
Kapitel 12: Code automatisch formatieren	41
Examples	41
Im normalen Modus:	41
Kapitel 13: Dateityp-Plugins	42
Examples	42
Wo kann ich benutzerdefinierte Dateityp-Plugins einfügen?	42
Kapitel 14: Der Punktoperator	43
Examples	43
Grundlegende Verwendung	43
Einzug setzen	43
Kapitel 15: Ex von der Kommandozeile aus verwenden	45

Examples.....	45
Ersetzung von der Befehlszeile aus.....	45
Kapitel 16: Falten.....	46
Bemerkungen.....	46
Examples.....	46
Faltmethode konfigurieren.....	46
Manuelles Erstellen einer Falte.....	46
Falten öffnen, schließen und umschalten.....	46
Die Zeile mit dem Cursor anzeigen.....	47
C-Blöcke falten.....	47
Kapitel 17: Fenster teilen.....	48
Syntax.....	48
Bemerkungen.....	48
Examples.....	48
Öffnen Sie mehrere Dateien in Splits von der Befehlszeile aus.....	48
Horizontal.....	48
Vertikal.....	48
Ein neues Splitfenster öffnen.....	48
Ändern der Größe eines Split oder vsplit.....	49
Tastenkombinationen.....	49
Schließen Sie alle Splits außer dem aktuellen.....	49
Open-Split-Fenster verwalten (Tastenkombinationen).....	49
Zwischen den Spalten wechseln.....	50
Vernünftige Spaltöffnung.....	50
Kapitel 18: Get: help (mit dem integrierten Handbuch von Vim).....	51
Einführung.....	51
Syntax.....	51
Parameter.....	51
Examples.....	51
Erste Schritte / Navigieren in den Hilfedateien.....	51
Das Handbuch durchsuchen.....	52
Kapitel 19: Konvertieren von Textdateien von DOS nach UNIX mit vi.....	53

Bemerkungen.....	53
Examples.....	53
Konvertieren einer DOS-Textdatei in eine UNIX-Textdatei.....	53
Verwenden des Dateiformats von Vim.....	53
Kapitel 20: Makros.....	55
Examples.....	55
Makro aufnehmen.....	55
Bearbeiten eines Vim-Makros.....	55
Rekursive Makros.....	56
Was ist ein Makro?.....	56
Aktion aufnehmen und wiedergeben (Makros).....	58
Kapitel 21: Modi - Einfügen, normal, visuell, z.....	60
Examples.....	60
Die Grundlagen zu den Modi.....	60
Normalmodus (oder Befehlsmodus).....	60
Einfügemodus.....	60
Visueller Modus.....	60
Auswahlmodus.....	60
Ersetzen Sie den Modus.....	60
Befehlszeilenmodus.....	61
Ex-Modus.....	61
Kapitel 22: Nützliche Konfigurationen, die in .vimrc eingefügt werden können.....	62
Syntax.....	62
Examples.....	62
Verschieben Sie die angezeigten Zeilen beim Wickeln nach oben oder unten.....	62
Aktivieren Sie die Mausinteraktion.....	62
Konfigurieren Sie das Standardregister für die Verwendung als Systemzwischenablage.....	62
Kapitel 23: Ostereier.....	64
Examples.....	64
Hilfel.....	64
Wenn du dich schlecht fühlst.....	64
Die Antwort.....	64

Auf der Suche nach dem Heiligen Gral.....	64
Ceci n'est pas une Pfeife.....	64
Wenn ein Benutzer sich langweilt.....	65
Löffel.....	65
Ritter, die Ni sagen!.....	65
nunmap.....	65
Kapitel 24: Plugins.....	67
Examples.....	67
Flüchtige Vim.....	67
NERD-Baum.....	67
Kapitel 25: Puffer.....	69
Examples.....	69
Puffer verwalten.....	69
Versteckte Puffer.....	69
Puffer wechseln mit einem Teil des Dateinamens.....	69
Wechseln Sie schnell zum vorherigen Puffer oder zu einem beliebigen Puffer nach Nummer.....	70
Kapitel 26: Rechtschreibprüfung.....	71
Examples.....	71
Rechtschreibprüfung.....	71
Setze Wortliste.....	71
Kapitel 27: Reguläre Ausdrücke.....	72
Bemerkungen.....	72
Examples.....	72
Wort.....	72
Kapitel 28: Reguläre Ausdrücke im Ex-Modus.....	73
Examples.....	73
Bearbeiten Sie einen regulären Ausdruck im Ex-Modus.....	73
Kapitel 29: Schlüsselzuordnungen in Vim.....	76
Einführung.....	76
Examples.....	76
Grundlegende Zuordnung.....	76
Übersichtskarte.....	76

Kartenoperator.....	76
Befehl map.....	77
Beispiele.....	77
Map Leader-Tastenkombination.....	77
Abbildung des grundlegenden Mappings (praktische Abkürzungen).....	78
Kapitel 30: Scrollen.....	80
Examples.....	80
Nach unten scrollen.....	80
Aufwärts scrollen.....	80
Bildlauf relativ zur Cursorposition.....	80
Kapitel 31: So kompilieren Sie Vim.....	82
Examples.....	82
Kompilieren auf Ubuntu.....	82
Kapitel 32: Solarized Vim.....	83
Einführung.....	83
Examples.....	83
.vimrc.....	83
Kapitel 33: Speichern.....	84
Examples.....	84
Speichern eines Puffers in einem nicht vorhandenen Verzeichnis.....	84
Kapitel 34: Suche im aktuellen Puffer.....	85
Examples.....	85
Suche nach einem beliebigen Muster.....	85
Suche nach dem Wort unter dem Cursor.....	85
Befehl in Zeilen ausführen, die Text enthalten.....	86
Kapitel 35: Suchen und Ersetzen.....	87
Examples.....	87
Ersetzen Sie den Befehl.....	87
Ersetzen Sie mit oder ohne reguläre Ausdrücke.....	88
Kapitel 36: Text bearbeiten.....	90
Bemerkungen.....	90

Examples.....	90
Textfall konvertieren.....	90
Im normalen Modus:.....	90
Im visuellen Modus:.....	90
Inkrementieren und Dekrementieren von Zahlen und Buchstaben.....	90
Zahlen inkrementieren und dekrementieren.....	90
Inkrementieren und Dekrementieren von alphabetischen Zeichen.....	91
Zahlen inkrementieren und dekrementieren, wenn das alphabetische Inkrementieren / Dekremen.....	91
Formatierungscode.....	92
Verwendung von "Verben" und "Substantiven" zur Textbearbeitung.....	92
Kapitel 37: Text einfügen.....	94
Examples.....	94
Verlassen des Einfügemodus.....	94
Verschiedene Möglichkeiten, um in den Einfügemodus zu gelangen.....	94
Einfügemodus-Verknüpfungen.....	95
Ausführen normaler Befehle aus dem Einfügemodus.....	95
Beispiel.....	96
Fügen Sie den Text in mehreren Zeilen gleichzeitig ein.....	96
Fügen Sie Text mit dem Befehl "Einfügen" des Terminals ein.....	96
Einfügen aus einem Register im Einfügemodus.....	97
Erweiterte Einfügebefehle und Verknüpfungen.....	97
Deaktivieren Sie den automatischen Einzug, um Code einzufügen.....	98
Kapitel 38: Tipps und Tricks zur Steigerung der Produktivität.....	100
Syntax.....	100
Bemerkungen.....	100
Examples.....	100
Schnelle "Wegwerf" -Makros.....	100
Verwenden der Funktion zum Vervollständigen des Pfads in Vim.....	100
Aktivieren Sie relative Zeilennummern.....	101
Zeilennummern anzeigen.....	102
Zuordnungen zum Beenden des Einfügemodus.....	102
j k.....	102

Feststelltaste	103
Linux	103
Windows	103
Mac OS	103
Wie Sie das Verzeichnis und / oder den Pfad der Datei, die Sie bearbeiten, kennen	104
Suche innerhalb eines Funktionsblocks	104
Gefundene Zeile kopieren, verschieben oder löschen	105
Schreiben Sie eine Datei, wenn Sie `sudo` vergessen haben, bevor Sie vim starten	106
Laden Sie vimrc beim Speichern automatisch neu	106
Kommandozeilenabschluss	106
Kapitel 39: Unterschiede zwischen Neovim und Vim	108
Examples	108
Konfigurationsdateien	108
Kapitel 40: Verbessertes Rückgängigmachen und Wiederherstellen mit einem Undodir	109
Examples	109
Konfigurieren Sie Ihre Vimrc für die Verwendung eines Undodir	109
Kapitel 41: Vertiefung	110
Examples	110
Einrücken einer gesamten Datei mithilfe der eingebauten Einrückungs-Engine	110
Zeilen einrücken oder ausrücken	110
Kapitel 42: Verwenden von Python für Vim-Scripting	112
Syntax	112
Examples	112
Überprüfen Sie die Python-Version in Vim	112
Führen Sie die Vim-Befehle im Normalmodus über die Python-Anweisung aus	112
Mehrzeiligen Python-Code ausführen	112
Kapitel 43: vglobal: Führt Befehle in Zeilen aus, die nicht global übereinstimmen	114
Einführung	114
Examples	114
: v / pattern / d	114
Kapitel 44: Vim erweitern	116

Bemerkungen.....	116
Examples.....	116
Wie funktionieren Plugins?.....	116
Das Prinzip.....	116
Die manuelle Methode.....	117
Plugin für einzelne Dateien.....	117
Bündeln.....	117
VAM.....	117
Vundle.....	118
Vundle installieren.....	118
Unterstützte Plugin-Formate.....	118
Die Zukunft: Pakete.....	119
Erreger.....	119
Pathogen installieren.....	119
Pathogen verwenden.....	119
Leistungen.....	120
Kapitel 45: Vim konfigurieren.....	121
Examples.....	121
Die vimrc-Datei.....	121
Welche Optionen kann ich verwenden?.....	121
Dateien und Verzeichnisse.....	122
Optionen.....	123
Boolesche Optionen einstellen.....	123
String-Optionen einstellen.....	123
Anzahl Optionen einstellen.....	123
Einen Ausdruck als Wert verwenden.....	124
Zuordnungen.....	124
Rekursive Zuordnungen.....	124
Nicht-rekursive Zuordnungen.....	124
Befehl aus einem Mapping ausführen.....	125
Mehrere Befehle aus einem Mapping ausführen.....	125

Aufruf einer Funktion aus einem Mapping.....	125
Zuordnen einer <Plug> -Zuordnung.....	125
Variablen.....	125
Befehle.....	125
Beispiele.....	126
Funktionen.....	126
Beispiel.....	126
Skriptfunktionen.....	126
S: Funktionen aus Zuordnungen verwenden.....	127
Autobefehlsgruppen.....	127
Beispiel.....	127
Conditionals.....	127
Einstellmöglichkeiten.....	128
Satzstellung markieren.....	128
Farbschemata.....	128
Farbschemas ändern.....	128
Farbschemas installieren.....	129
Zeilenaufzählung umschalten.....	129
Plugins.....	129
Kapitel 46: Vim verlassen.....	130
Parameter.....	130
Bemerkungen.....	130
Examples.....	130
Beenden Sie mit Speichern.....	130
Beenden ohne zu speichern.....	130
Zwangsweise beenden (ohne Speichern).....	130
Zwangsweise beenden (mit Speichern).....	131
Beenden Sie das Öffnen aller Fenster (ohne Speichern).....	131
wenn mehrere Dateien geöffnet sind.....	131
Kapitel 47: Vim-Optionen.....	132
Syntax.....	132
Bemerkungen.....	132

Examples	132
einstellen	132
Vertiefung	132
Breite	132
Räume	132
Tabs	133
Automatische Einrückung	133
Anweisungsbeschreibungen	133
Unsichtbare Zeichen	133
Unsichtbare Zeichen anzeigen oder ausblenden	133
Standardsymbolzeichen	134
Anpassen von Symbolen	134
Kapitel 48: Vim-Register	135
Parameter	135
Examples	135
Löschen Sie einen Zeilenbereich in einem benannten Register	135
Fügen Sie den Dateinamen im Einfügemodus mit dem Dateinamenregister ein	136
Kopieren / Einfügen zwischen Vim und der Systemzwischenablage	136
An ein Register anhängen	136
Kapitel 49: Vim-Ressourcen	137
Bemerkungen	137
Examples	137
Vimscript auf die harte Tour lernen	137
Kapitel 50: Vimscript	138
Bemerkungen	138
Examples	138
Hallo Welt	138
Verwenden der Befehle im Normalmodus in Vimscript	138
Kapitel 51: Vim-Textobjekte	140
Examples	140
Wählen Sie ein Wort ohne umgebende Leerzeichen aus	140

Wählen Sie ein Wort mit umgebendem Leerzeichen aus.....	140
Wählen Sie Text innerhalb eines Tags aus.....	140
Kapitel 52: Vorteile von vim.....	142
Examples.....	142
Anpassung.....	142
Leicht.....	142
Kapitel 53: Whitespace.....	144
Einführung.....	144
Bemerkungen.....	144
Examples.....	144
Löschen Sie nachfolgende Leerzeichen in einer Datei.....	144
Löschen Sie leere Zeilen in einer Datei.....	144
Konvertieren Sie Tabs in Leerzeichen und Leerzeichen in Tabs.....	145
Credits.....	146



You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [vim](#)

It is an unofficial and free vim ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official vim.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit vim

Bemerkungen

Vim (oder "Vi IMproved") ist ein konsolenbasierter Multi-Mode-Texteditor (*modal*). Es ist weit verbreitet und standardmäßig auf allen Unix-, Linux- und Apple OS X-Systemen verfügbar. Vim hat eine große aktive Community und eine breite Benutzerbasis. Der Editor unterstützt alle gängigen Programmiersprachen. Viele Plugins sind verfügbar, um die Funktionen zu erweitern.

Entwickler schätzen den Editor aufgrund seiner Geschwindigkeit, seiner vielen Konfigurationsoptionen und der ausdrucksstarken Bearbeitung. Im Befehlsmodus wird der Editor durch Tastaturbefehle gesteuert, sodass der Benutzer nicht durch eine GUI oder einen Mauszeiger abgelenkt wird.

Vim basiert auf dem früheren Unix-Editor "vi", der in den siebziger Jahren erstellt wurde, und wird seit 1991 kontinuierlich weiterentwickelt. Mit Makros und Plugins bietet der Editor die meisten Funktionen einer modernen IDE. Mit seiner Skriptsprache (Vimscript) und regulären Ausdrücken ist es auch in der Lage, große Textmengen zu verarbeiten.

Hauptthemen:

- [Installation](#)
- Bearbeitungsmodi
- [Navigation](#)
- [grundlegende Bearbeitung](#)
- Erweiterte Bearbeitung
- [Aufbau](#)
- [Plugins](#)
- [vimscript](#)
- [Makros](#)
- Gemeinschaft
- Ähnliche Projekte

$\theta, \theta-1$

Ausführung	Veröffentlichungsdatum
1.14	1991-11-02

Examples

Installation

Das Vim auf Ihrem Computer ist - sofern vorhanden - sehr wahrscheinlich ein "kleiner" Build, dem nützliche Funktionen wie Unterstützung der Zwischenablage, Syntaxhervorhebung oder sogar die Verwendung von Plugins fehlen.

Dies ist kein Problem, wenn Sie nur eine schnelle Methode zum Bearbeiten von Konfigurationsdateien benötigen. Wenn Sie jedoch Vim zu Ihrem Haupteditor machen möchten, werden Sie bald eine Reihe von Wänden treffen.

Es wird daher generell empfohlen, einen vollständigen Build zu installieren.

Installation unter Linux / BSD

Auf diesen Systemen besteht der Trick darin, die GUI-Version zu installieren, die sowohl einen `gvim` Befehl zum Starten der GUI als auch einen `vim` Befehl zum Starten der TUI enthält.

Arch- und Arch-basierte Distributionen

```
$ sudo pacman -R vim
$ sudo pacman -S gvim
```

Debian- und Debian-basierte Distributionen

```
$ sudo apt-get update
$ sudo apt-get install vim-gtk
```

Gentoo- und Gentoo-basierte Distributionen

```
$ sudo emerge --sync
$ sudo emerge app-editors/gvim
```

RedHat- und RedHat-basierte Distributionen

```
$ sudo yum check-update
$ sudo yum install vim-X11
```

Fedora

```
$ sudo dnf check-update
$ sudo dnf install vim-X11
```

Slackware- und Slackware-basierte Distributionen

```
$ sudo slackpkg update
$ sudo slackpkg install-new vim-gvim
```

OpenBSD- und OpenBSD-basierte Distributionen

```
$ sudo pkg_add vim-x11
```

FreeBSD- und FreeBSD-basierte Distributionen

```
$ sudo pkg install editors/vim
```

Installation unter Mac OS X

Die Strategie ist ähnlich wie bei Mac OS X: Wir installieren die GUI-Version, um sowohl die GUI als auch die TUI zu erhalten. Am Ende sollten wir in der Lage sein:

- Doppelklicken Sie im Finder auf das MacVim-Symbol.
- Klicken Sie im Dock auf das MacVim-Symbol.
- geben Sie `$ mvim` in der Shell ein, um die MacVim- `$ mvim` zu öffnen.
- `$ mvim -v` in der Shell `$ mvim -v` ein, um die MacVim-TUI zu öffnen.

Regelmäßige Installation

Laden Sie [einen offiziellen Snapshot](#) herunter und installieren Sie ihn wie mit jeder anderen Mac OS X-Anwendung.

Platzieren Sie das mit MacVim mitgelieferte `mvim` Skript irgendwo in Ihrem `$PATH`.

Paket-Manager

MacPorts:

```
$ sudo port selfupdate
$ sudo port install macvim
```

Homebrew:

```
$ brew install macvim
```

So machen Sie MacVim zur Standardkonsole Vim:

```
$ brew install macvim --with-override-system-vim
```

Installation unter Windows

Auf Windows-Systemen ist standardmäßig kein Vim vorhanden. Sie können Vim von der [Tuxproject-Site](#) herunterladen und installieren, um aktuellere und vollständige Builds zu erhalten, oder Sie können Vim von der offiziellen [Vim-Site](#) herunterladen und installieren.

Schokoladig

```
> choco install vim
```

Vim von der Quelle aus bauen

Wenn die oben genannten Methoden nicht Ihre Bedürfnisse ist es noch möglich , Vim selbst zu bauen, *nur* die Optionen , die Sie benötigen.

Dieses Thema wird in einem eigenen Abschnitt (derzeit im Entwurf) erörtert.

Vim verlassen

Um Vim zu beenden, zuerst sicherstellen , dass Sie im *Normal* - Modus sind durch Drücken der Taste `Esc`.

- `:q` Enter (verhindert das Beenden, wenn Sie nicht gespeicherte Änderungen haben - kurz für: quit)

Änderungen *verwerfen* und Vim beenden:

- `:q!` Geben Sie ein , um die Änderungen zu `:quit!` und zu verwerfen (kurz für `:quit!` Nicht zu verwechseln mit `:!q`),
- `ZQ` ist eine Abkürzung, die dasselbe tut wie `:q!` .
- `:cq` Geben Sie quit und return error ein (verwerfen Sie alle Änderungen, damit der Compiler diese Datei nicht neu kompiliert).

Änderungen *speichern* und Vim beenden:

- `:wq` Enter (Abkürzung für `:write` und `:quit`),
- `:x` Eingabe (wie bei `:wq` , `:wq` aber nicht, wenn die Datei nicht geändert wurde),
- `ZZ` ist eine Verknüpfung, die das Gleiche tut wie `:x` (Arbeitsbereich speichern und Editor beenden),
- `:[range]wq!` Eingabe (schreibe die Zeilen in [Bereich])

Um mehrere Puffer gleichzeitig zu schließen (auch in mehreren Fenstern und / oder Registerkarten), hängen Sie den Buchstaben `a` an einen der *Befehle* oben (die mit `:`). Zum Beispiel zum Schreiben und Beenden aller Fenster, die Sie verwenden können:

- `:wqa` Geben Sie **oder ein**
- `:xa Enter` - Schreiben Sie alle geänderten Puffer und beenden Sie Vim. Wenn Puffer ohne Dateinamen vorhanden sind, die nur lesbar sind oder aus einem anderen Grund nicht geschrieben werden können, wird Vim nicht beendet
- `:xa! Enter` - Schreiben Sie alle geänderten Puffer, auch die schreibgeschützten, und beenden Sie Vim. Wenn es Puffer gibt, die keinen Dateinamen haben oder aus einem anderen Grund nicht geschrieben werden können, wird Vim nicht beendet
- `:qa Enter` - versuchen Sie den Vorgang zu beenden, aber stoppen Sie nicht gespeicherte Dateien.
- `:qa! Enter` - *ohne Speichern* abbrechen (Änderungen in *nicht* gespeicherten Dateien verwerfen)

Wenn Sie Vim geöffnet haben, ohne eine Datei anzugeben, und diese Datei vor dem Beenden speichern möchten, erhalten Sie die Nachricht `E32: No file name`. Sie können Ihre Datei speichern und beenden mit:

- `:wq filename Enter` **oder**;
- `:x filename` eingeben

Erläuterung:

Die `:` Tastendruck öffnet sich tatsächlich *Befehlsmodus*. Der Befehl `q` ist eine Abkürzung für `quit`, `w`, von `write` und `x`, `exit` (Sie können auch Folgendes `:quit` `:write` und `:exit` wenn Sie möchten). Verknüpfungen beginnend *nicht* `:` wie `zz` und `zQ` in den *Normalmodus* Tastenzuordnungen beziehen. Sie können sich als Abkürzungen vorstellen.

Die `!` Der Tastenanschlag wird manchmal am Ende eines Befehls verwendet, um die Ausführung zu erzwingen, wodurch Änderungen verworfen werden können `:q!`. Platzieren der `!` am Anfang des Befehls hat eine andere Bedeutung. Zum Beispiel kann man falsch eingeben: `!!q` statt `:q!` und vim würde mit einem Fehler von 127 enden.

Eine einfache Möglichkeit, sich daran zu erinnern, ist, daran zu denken `!` als eine Möglichkeit, darauf zu bestehen, etwas auszuführen. Genau wie beim Schreiben: "Ich möchte aufhören!"

Interaktive Vim-Tutorials (z. B. Vimtutor)

`vimtutor` ist ein interaktives Tutorial, das die grundlegendsten Aspekte der Textbearbeitung behandelt.

Auf einem UNIX-ähnlichen System können Sie das Lernprogramm mit folgendem Befehl starten:

```
$ vimtutor
```

Unter Windows befindet sich „Vim Tutor“ im Verzeichnis „Vim 7.x“ unter „All Programs“ im Windows-Menü.

Weitere Informationen finden Sie unter `:help vimtutor`.

Andere interaktive Tutorials umfassen diese browserbasierten:

- [Vim Adventures](#) - Eine interaktive Spielversion von vimtutor, die im Internet verfügbar ist. Nur die ersten Stufen sind frei.
- [Open Vim](#) - Ein interaktives Terminal, das Ihnen die grundlegenden Befehle mit Feedback vermittelt.
- [Vim Genius](#) - Ein weiteres interaktives Terminal, das auch fortgeschrittene und fortgeschrittene Lektionen einschließlich Makros und Arglist beinhaltet.

Speichern einer schreibgeschützten Datei, die in Vim bearbeitet wurde

Manchmal öffnen wir möglicherweise eine Datei, für die wir keine Berechtigung zum Schreiben in Vim haben, ohne `sudo`.

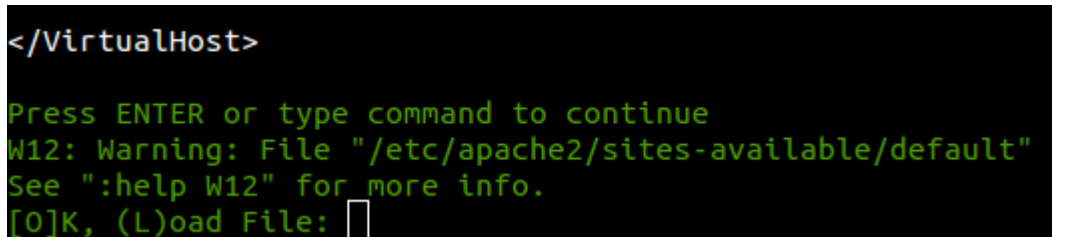
Verwenden Sie diesen Befehl, um eine schreibgeschützte Datei zu speichern, die in Vim bearbeitet wurde.

```
:w !sudo tee > /dev/null %
```

Welches Sie zuordnen könnten `:w!!` in deiner `.vimrc`:

```
cmap w!! w !sudo tee > /dev/null %
```

Es wird eine Aufforderung angezeigt, wie im Bild gezeigt.



Drücken Sie `o` und die Datei wird gespeichert. In `vi` / `vim` bleibt es für weitere Bearbeitungen oder zum Lesen geöffnet. Sie können das Programm normalerweise beenden, indem Sie Folgendes eingeben `:q!` da die Datei noch schreibgeschützt ist.

Befehlserklärung

```
:w ..... isn't modifying your file in this case,
..... but sends the current buffer contents to
..... a substituted shell command
!sudo ..... call the shell 'sudo' command
tee ..... the output of the vi/vim write command is redirected
              using the 'tee' command
      > /dev/null ..... throws away the standard output, since we don't need
                          to pass it to other commands
              % .... expands to the path of the current file
```

Quellen:

- [Adam Culps Tech-Blog](#)
- [Stackoverflow, Wie funktioniert der vim-Trick "Schreiben mit Sudo"?](#)

Vim aussetzen

Wenn Sie `vim` von der Befehlszeile aus verwenden, können Sie `vim` aussetzen und zu Ihrer Eingabeaufforderung zurückkehren, ohne `vim` tatsächlich zu beenden. Daher können Sie Ihre `vim` Sitzung später von derselben Eingabeaufforderung abrufen.

Im *Normalmodus* (wenn nicht, drücken Sie `esc`, um dorthin zu gelangen), geben Sie einen der folgenden Befehle aus:

```
:st eingeben

:sus eintreten

:stop enter

:suspend Enter :suspend
```

Alternativ auf einigen Systemen, wenn im *normalen* oder *visuellen* Modus, Ausgabe `Strg + z` die gleiche Wirkung hat.

Hinweis: Wenn `autowrite` eingestellt ist, werden Puffer mit Änderungen und Dateinamen ausgeschrieben. Fügen Sie ein `!` vor `eingeben`, um zu vermeiden, z. `:st! eintreten`

Wenn Sie später zu Ihrer `vim` Sitzung zurückkehren möchten, wenn Sie keine anderen Jobs angehalten haben, wird durch das Ausgeben der folgenden Informationen vim als Vordergrundjob wiederhergestellt.

```
fg eingeben
```

Andernfalls müssen Sie die Job-ID Ihrer `vim` Sessions finden, indem Sie `jobs` eingeben und dann die entsprechenden Jobs in den Vordergrund legen. `fg %[job ID] eingeben`, z. `fg %1 eingeben`.

Grundlagen

Führen Sie interaktive [vim-Tutorials](#) so oft aus, wie Sie es benötigen, um sich mit den Grundlagen vertraut zu machen.

Vim bietet verschiedene Modi, z. B. **Normalmodus**, **Einfügemodus** und **Befehlszeilenmodus**.

Im Normalmodus können Sie Text bearbeiten und navigieren. In diesem Modus entsprechen `h`, `j`, `k` und `l` den Cursortasten `←`, `↓`, `↑` und `→`. Den meisten Befehlen im Normalmodus kann ein "count" vorangestellt werden, z. B. `3j` fährt 3 Zeilen nach unten.

Der Einfügemodus dient zum direkten Einfügen des Textes. In diesem Modus ähnelt vim anderen einfacheren Texteditoren. Um den Einfügemodus zu aktivieren, drücken Sie `i` im normalen Modus. Zum Verlassen drücken Sie `<ESC>` (Escape-Taste).

Im Befehlszeilenmodus können komplexere Befehle ausgeführt werden, beispielsweise das Speichern der Datei und das Beenden von vim. Drücken Sie `:` um den Befehlszeilenmodus zu starten. Um diesen Modus zu verlassen, können Sie auch `<ESC>` drücken. Um die Änderungen an der Datei zu speichern, verwenden Sie `:w` (oder `:write`). Um vim zu verlassen, ohne die Änderungen zu speichern, verwenden Sie `:q!` (oder `:quit!`).

Dies sind einige der nützlicheren Befehle in vim:

Befehl	Beschreibung
i	(Einfügen) wechselt in den Einfügemodus <i>vor</i> der aktuellen Cursorposition
I	wechselt in den Einfügemodus <i>vor</i> dem ersten druckbaren Zeichen der aktuellen Zeile
a	(Anfügen) wechselt <i>nach</i> der aktuellen Cursorposition in den Einfügemodus
A	wechselt in den Einfügemodus <i>nach</i> dem letzten druckbaren Zeichen der aktuellen Zeile
x	Zeichen an der aktuellen Cursorposition löschen
X	Zeichen links an der aktuellen Cursorposition löschen
w	zum nächsten Wort wechseln
b	zum vorherigen Wort wechseln
0	zum Zeilenanfang gehen
\$	zum Ende der Zeile gehen
r	Ersetzen - Ruft den Ersetzungsmodus für ein Zeichen auf. Das nächste Zeichen, das Sie eingeben, wird das Zeichen unter dem Cursor ersetzen.
R	wechselt unbegrenzt in den Ersetzungsmodus. Jedes Zeichen, das Sie eingeben, wird das Zeichen unter dem Cursor ersetzen und den Cursor um eins vorrücken.
s	Ersatz - löscht das Zeichen an der aktuellen Cursorposition und wechselt dann in den Einfügemodus
S	Löschen Sie die aktuelle Zeile, in der sich der Cursor gerade befindet, und wechseln Sie in den Einfügemodus
<Esc> , <Cc>	Beenden Sie den Einfügemodus und kehren Sie zum Normalmodus zurück
u	rückgängig machen

Befehl	Beschreibung
<Cr>	Wiederholen
dd , dw , dl , d\$	Schneiden Sie die aktuelle Zeile vom Cursor zum nächsten Wort oder das Zeichen, die aktuelle Position bis zum Ende der aktuellen Zeile. Hinweis: D ist das Äquivalent von d\$
cc , cw , cl	wechselt die aktuelle Zeile vom Cursor zum nächsten Wort bzw. zum Zeichen
yy , yw , yl , y\$	yank ("copy") die aktuelle Zeile, vom Cursor zum nächsten Wort oder das Zeichen, die aktuelle Position bis zum Ende der aktuellen Zeile
p , P	Put ("Einfügen") nach bzw. vor der aktuellen Position
o , O	um eine neue leere Zeile nach oder vor der aktuellen zu erstellen und den Einfügemodus zu aktivieren
:w	schreibe den aktuellen Puffer auf die Festplatte
:q! , ZQ	Beenden ohne zu schreiben
:x :wq , ZZ	schreiben und beenden
:help	Öffnet ein Fenster mit Hilfedatei
:help {subject}	Hilfe zu einem bestimmten Thema anzeigen
qz	Beginnen Sie mit den Aufzeichnungsaktionen, um z zu registrieren, q , um die Aufnahme zu beenden, und @z , um die Aktionen @z . z kann ein beliebiger Buchstabe sein: q wird häufig aus Gründen der Übersichtlichkeit verwendet. Lesen Sie mehr: Makros

Was tun im Falle eines Absturzes?

Vim speichert alle nicht gespeicherten Änderungen in einer *Auslagerungsdatei* , einer zusätzlichen Datei, die gelöscht wird, sobald die Änderungen durch Speichern übernommen werden. Der Name der Auslagerungsdatei ist normalerweise der Name der zu bearbeitenden Datei, vorangestellt von einem . und mit einem .swp Suffix (Sie können es mit :sw).

Falls Ihr vim-Prozess beendet wird, bevor Sie die Änderungen speichern konnten, können Sie Ihre Arbeit wiederherstellen, indem Sie die in der Auslagerungsdatei enthaltenen Änderungen auf Ihre aktuelle Datei anwenden, indem Sie die Befehlszeilenoption -r . Wenn beispielsweise myFile die Datei ist, die Sie bearbeitet haben, verwenden Sie myFile :

```
$ vi -r myFile
```

um die nicht festgeschriebenen Änderungen wiederherzustellen.

Wenn eine Auslagerungsdatei vorhanden ist, sollten Sie von vim trotzdem zur Eingabe von Wiederherstellungsoptionen aufgefordert werden

```
$ vi myFile
E325: ATTENTION
Found a swap file by the name ".myFile.swp"
...
Swap file ".myFile.swp" already exists!
[O]pen Read-Only, (E)dit anyway, (R)ecover, (D)elele it, (Q)uit, (A)bort:
```

Wenn Sie (R) ecover wählen, werden die Änderungen aus der `swp` Datei übernommen, die Auslagerungsdatei wird jedoch nicht gelöscht. Vergessen Sie nicht, die Auslagerungsdatei anschließend zu löschen, wenn Sie mit der Wiederherstellung zufrieden sind.

Erste Schritte mit vim online lesen: <https://riptutorial.com/de/vim/topic/879/erste-schritte-mit-vim>

Kapitel 2: : global

Syntax

- : [<range>]g[lobal]/{<pattern>}/{<command>}
- : [<range>]g[lobal]!/{<pattern>}/{<command>} (invertiert)
- : [<range>]v[lobal]/{<pattern>}/{<command>} (invertiert)

Bemerkungen

Der "globale" Befehl von Vim wird verwendet, um einen Ex-Befehl auf jede Zeile anzuwenden, in der ein Regex übereinstimmt.

Examples

Grundlegende Verwendung des Global Command

```
:g/Hello/d
```

Löscht jede Zeile, die den Text "Hallo" enthält. **Wichtiger Hinweis** : Dies ist nicht der normale Modusbefehl `d`, sondern der ex-Befehl `:d`.

Sie können den globalen Befehl verwenden, um Tastatureingaben im Normalmodus anstelle von Ex-Befehlen anzuwenden, indem Sie dem Befehl `normal` oder `norm` voranstellen. Zum Beispiel:

```
:g/Hello/norm dw
```

Löscht das erste Wort aus jeder Zeile, die den Text "Hallo" enthält.

Der globale Befehl unterstützt auch den visuellen Modus [und die Bereiche](#).

Ziehe jede Zeile, die einem Muster entspricht

Der Befehl

```
:g/apples/y A
```

werden alle Zeilen zerren, die das Wort *Äpfel* in die `a` Register, das mit eingefügt werden kann `"ap`. Jeder Ausdruck regelmäßig verwendet werden kann.

Beachten Sie das Leerzeichen vor dem `A` und die Schreibweise des Registers. Wenn ein Großbuchstabe als Ruckregister verwendet wird, werden Übereinstimmungen an dieses Register *angehängt*. Wenn ein Kleinbuchstabe verwendet wird, wird nur die *letzte* Übereinstimmung in diesem Register abgelegt.

Zeilen mit wichtigen Informationen verschieben / sammeln

ein einfacher, aber sehr nützlicher Befehl:

```
:g/ending/m$
```

verschiebt Zeilen mit dem `ending` an das Ende des Puffers.

`m` bedeutet sich bewegen

`$` bedeutet Pufferende, während `0` Pufferanfang bedeutet.

: global online lesen: <https://riptutorial.com/de/vim/topic/3861/--global>

Kapitel 3: Auswechslung

Syntax

- `s/<pattern>/<pattern>/optional-flags`
- `<pattern>` ist ein Regex

Parameter

Flagge	Bedeutung
<code>&</code>	Behalten Sie die Flaggen vom vorherigen Ersatz.
<code>c</code>	Aufforderung zur Bestätigung jeder Auswechslung.
<code>e</code>	Melden Sie keine Fehler.
<code>G</code>	Ersetzen Sie alle Vorkommen in der Zeile.
<code>ich</code>	Groß- und Kleinschreibung wird nicht berücksichtigt.
<code>ich</code>	Groß- und Kleinschreibung wird berücksichtigt.
<code>n</code>	Melden Sie die Anzahl der Übereinstimmungen, ersetzen Sie sie nicht.

Bemerkungen

Verwenden Sie `set gdefault`, um zu vermeiden, dass bei jedem Stellvertreter das 'g' Flag angegeben werden muss.

Beispiel

Wenn `gdefault` eingestellt ist, führt das Ausführen von `:s/foo/bar` in der Zeile `foo baz foo` anstelle von `bar baz foo bar baz bar bar baz foo`.

Examples

Einfacher Austausch

`:s/foo/bar` Ersetzen Sie die **erste** Instanz von `foo` durch einen *Balken* in der aktuellen Zeile.

`:s/foo/bar/g` Ersetzen Sie jede Instanz von `foo` durch einen *Balken* in der aktuellen Zeile.

`:%s/foo/bar/g` Ersetzen Sie *foo* durch einen *Balken* in der gesamten Datei.

Refactor das Wort unter dem Cursor schnell

1. * auf dem Wort, das Sie ersetzen möchten.
2. `:%s//replacement/g` , leert den *Fund* Muster zu verlassen.

Ersatz mit interaktiver Genehmigung

`:s/foo/bar/c` Markiert die erste Instanz von *foo* in der Zeile und fordert zur Bestätigung der Ersetzung mit *bar auf*

`:s/foo/bar/gc` nacheinander jedes Match von *foo* in der Datei und fordert zur Bestätigung der Ersetzung mit *bar auf*

Tastenkombination zum Ersetzen des aktuell hervorgehobenen Wortes

Zum Beispiel mit folgender `nmap` :

```
nmap <expr> <S-F6> ':%s/' . @/ . '//gc<LEFT><LEFT><LEFT>'
```

Wählen Sie mit * ein Wort aus, geben Sie `Shift - F6` ein , geben Sie einen Ersatz ein und drücken Sie die `Eingabetaste` , um alle Vorkommen interaktiv umzubenennen.

Auswechslung online lesen: <https://riptutorial.com/de/vim/topic/3384/auswechslung>

Kapitel 4: Autobefehle

Bemerkungen

`autocmd` Befehle

`autocmd` ist ein additiver Befehl, und Sie möchten dieses Verhalten möglicherweise standardmäßig nicht.

Wenn Sie zum Beispiel während der Bearbeitung Ihre `.vimrc` - `.vimrc` erneut `.vimrc`, kann sich vim verlangsamen.

Hier ist der Beweis:

```
:autocmd BufWritePost * if &diff | diffupdate | endif " update diff after save
:autocmd BufWritePost * if &diff | diffupdate | endif " update diff after save
```

Wenn Sie jetzt `:autocmd BufWritePost *`, werden in der Ausgabe **beide** Zeilen `:autocmd BufWritePost *`, nicht nur eine. Beide werden hingerichtet.

Um dieses Verhalten zu vermeiden, umgeben Sie alle `autocmd` wie folgt:

```
if has ('autocmd')          " Remain compatible with vi which doesn't have autocmd
  augroup MyDiffUpdate      " A unique name for the group. DO NOT use the same name twice!
    autocmd!                " Clears the old autocommands for this group name
    autocmd BufWritePost * if &diff | diffupdate | endif " Update diff after save
    " ... etc ...
  augroup END
endif
```

Examples

Quelle automatisch `.vimrc` nach dem Speichern

Fügen Sie dies Ihrem `$MYVIMRC`:

```
" Source vim configuration file whenever it is saved
if has ('autocmd')          " Remain compatible with earlier versions
  augroup Reload_Vimrc      " Group name. Always use a unique name!
    autocmd!                " Clear any preexisting autocommands from this group
    autocmd! BufWritePost $MYVIMRC source % | echom "Reloaded " . $MYVIMRC | redraw
    autocmd! BufWritePost $MYGVIMRC if has('gui_running') | so % | echom "Reloaded " .
$MYGVIMRC | endif | redraw
  augroup END
endif " has autocmd
```

Eigenschaften:

- `echom` teilt dem Benutzer mit, was passiert ist (und meldet sich auch an `:messages`).

- `$MYVIMRC` und `$MYGVIMRC` behandeln plattformspezifische Namen für die Konfigurationsdateien.
- und stimmen nur mit den tatsächlichen Konfigurationsdateien überein (ignoriert Kopien in anderen Verzeichnissen oder ein `fugitive:// diff`)
- `has()` verhindert einen Fehler, wenn inkompatible Versionen wie `vim-tiny` .
- `autocmd!` vermeidet den Aufbau mehrerer identischer Autobefehle, wenn diese Datei erneut beschafft wird. (Es löscht alle Befehle in der genannten Gruppe, daher ist der Gruppenname wichtig.)

Aktualisieren Sie die vimdiff-Ansichten, wenn eine Datei in einem anderen Fenster gespeichert wird

```
:autocmd BufWritePost * if &diff | diffupdate | endif
```

Autobefehle online lesen: <https://riptutorial.com/de/vim/topic/4887/autobefehle>

Kapitel 5: Bauen von vim

Examples

Build starten

`:mak[e] [!] [arguments]` startet das Programm, auf das die Option `makeprg` . Standardmäßig ist "`makeprg`" auf "make" gesetzt, kann aber so konfiguriert werden, dass jedes geeignete Programm `makeprg` wird.

Alle `[arguments]` (können mehrere sein) werden an `makeprg` als ob es mit `:{makeprg} [arguments]` aufgerufen worden wäre.

Die Ausgabe des aufgerufenen Programms wird gemäß der Option '`errorformat`' auf Fehler analysiert. Wenn Fehler gefunden werden, wird das Quickfix-Fenster geöffnet, um sie anzuzeigen.

`:cnext` `:cprev` kann verwendet werden, um zwischen Fehlern zu wechseln, die im Quickfix-Fenster angezeigt werden. `:cc` springt zum Fehler unter dem Cursor.

Es ist zu beachten, dass auf Systemen, auf denen `gnumake` installiert und ordnungsgemäß konfiguriert ist, es in der Regel nicht erforderlich ist, `&makeprg` außer dem Standardwert zu definieren, um `&makeprg` -Projekte zu kompilieren. Geben Sie also in C, C ++, Fortran ... Folgendes ein `:make %<` , um die aktuelle Datei zu kompilieren. `:!./%<` die Quelldatei im aktuellen Verzeichnis befindet, wird sie von `:!./%<` ausgeführt. Kompilierungsoptionen können über `$CFLAGS` , `$CXXFLAGS` , `$LDFLAGS` usw. gesteuert werden. `$CFLAGS` zu *impliziten Regeln* finden Sie in der Dokumentation zu `make` .

Bauen von vim online lesen: <https://riptutorial.com/de/vim/topic/3321/bauen-von-vim>

Kapitel 6: Befehle im Normalmodus

Syntax

- : [Bereich] sor [t] [!] [b] [f] [i] [n] [o] [r] [u] [x] [/ {Muster} /]
- Hinweis: Die Optionen [n] [f] [x] [o] [b] schließen sich gegenseitig aus.

Bemerkungen

Siehe [Sortierung](#) im vim Handbuch für die kanonische Erklärung

Examples

Text sortieren

Normale Sortierung

Markieren Sie den zu sortierenden Text und den Typ:

```
:sort
```

Wenn Sie keinen Text markieren oder einen Bereich angeben, wird der gesamte Puffer sortiert.

Rückwärtssortierung

```
:sort!
```

Groß- und Kleinschreibung wird nicht berücksichtigt

```
:sort i
```

Numerische Sortierung

Sortieren Sie nach der ersten Nummer, die in jeder Zeile angezeigt wird:

```
:sort n
```

Duplikate nach dem Sortieren entfernen

`:sort u` (u steht für unique)

Optionen kombinieren

Um eine umgekehrte Sortierung zu erhalten, ohne Duplikate zu entfernen:

`:sort! iu`

Befehle im Normalmodus online lesen: <https://riptutorial.com/de/vim/topic/6005/befehle-im-normalmodus>

Kapitel 7: Befehle im Normalmodus (Bearbeitung)

Examples

Einführung - Kurzer Hinweis zum Normalmodus

Im Normalmodus können Befehle durch direkte Tastenkombinationen eingegeben werden (z. B. `u`, um die letzte Änderung rückgängig zu machen). Diese Befehle haben häufig Äquivalente im 'ex'-Modus, auf die Sie durch Eingabe eines Doppelpunkts `:` zugreifen können, der Sie in einem einzeiligen Puffer am unteren Rand des Vim-Fensters ablegt.

Im 'Ex'-Modus geben Sie nach der Eingabe des Doppelpunkts einen Befehlsnamen oder seine Abkürzung gefolgt von `Enter` ein, um den Befehl auszuführen. Also `: undo Enter` bewirkt dasselbe wie das direkte Tippen von `u` im Normalmodus.

Sie sehen, dass die direkten Befehle oft schneller sind (einmal gelernt) als die 'ex'-Befehle für die einfache Bearbeitung. Der Vollständigkeit halber werden sie jedoch in der folgenden Dokumentation, soweit möglich, angezeigt, wenn beide verfügbar sind.

Den meisten dieser Befehle kann auch eine *Zählung* vorangestellt werden, indem eine Zahl vorangestellt oder eingefügt wird. Durch Eingabe von `3dd` im Normalmodus werden beispielsweise drei Zeilen (beginnend an der aktuellen Cursorposition) gelöscht.

Grundlegendes Rückgängig machen und Wiederholen

Rückgängig machen

Befehl	:	Beschreibung
<code>u</code>	<code>u</code> <code>undo</code>	Machen Sie die letzte Änderung rückgängig
<code>5u</code>		Die letzten <i>fünf</i> Änderungen rückgängig machen (beliebige Anzahl verwenden)

Bitte beachten Sie, dass in Vim die 'letzte Änderung' je nach Modus variiert wird. Wenn Sie den Einfügemodus (`i`) aufrufen und einen ganzen Absatz eingeben, bevor Sie zum Normalmodus (`Esc`) zurückkehren, ist *dieser gesamte Absatz* der Fall als letzte Änderung betrachtet.

Wiederholen

Befehl	:	Beschreibung
Ctrl-R	red redo	Wiederholt die letzte rückgängig gemachte Änderung
2Ctrl-R		Wiederholen Sie die <i>beiden</i> letzten rückgängig gemachten Änderungen (verwenden Sie eine beliebige Zahl).

Es gibt eine andere Möglichkeit, Änderungen in Vim rückgängig zu machen und wiederherzustellen, die etwas anders behandelt werden. Wenn Sie eine Änderung mit `u` rückgängig machen, durchlaufen Sie die Knoten in einem 'Baum' Ihrer Änderungen, und drücken Sie die Tastenkombination `strg-R`, um die Knoten in der Reihenfolge nach unten zu bewegen. (Der Undo-Baum ist ein separates Thema und zu komplex, um hier behandelt zu werden.)

Sie können *auch* `U` (also Großbuchstaben) verwenden, um alle letzten Änderungen in einer einzelnen Zeile (der Zeile, in der Ihre letzten Änderungen vorgenommen wurden) zu entfernen. Dies *schließt nicht* die Knoten des Baumes in der gleichen Weise wie `u` durchlaufen. Mit `U` zählt tatsächlich als eine Änderung selbst - ein anderen Knoten *vorwärts* auf dem Baum - so dass, wenn Sie `U` ein zweites Mal unmittelbar nach dem ersten Drücken sie als Redo - Befehl handeln.

Jeder hat seine Verwendung, aber `u` / `: undo` sollte die meisten einfachen Fälle abdecken.

Wiederholen Sie die letzte Änderung

Der Befehl Wiederholen, der mit der Punkt- oder Punkttaste (`.`) Ausgeführt wird, ist nützlicher als er zuerst erscheint. Einmal erlernt, werden Sie es oft verwenden.

Befehl	:	Beschreibung
.		Wiederholen Sie die letzte Änderung
10		Wiederholen Sie die letzte Änderung 10 Mal

Also dann, für ein sehr einfaches Beispiel, wenn Sie eine Änderung an Zeile macht 1 durch `i` eingeben `i` Esc, mit folgendem Ergebnis:

```
1 I made a mistake
2  made a mistake
3  made a mistake
```

Der Cursor befindet sich an Position 1 der Zeile 1, und Sie müssen nur noch `j` drücken, um die nächsten beiden Zeilen zu fixieren. `.` zweimal - das heißt, `j` bewegt sich eine Zeile nach unten und `.`. Um die letzte Änderung zu wiederholen, war das Hinzufügen des `i`. Sie müssen nicht zweimal in den Einfügemodus zurückspringen, um diese Zeilen zu korrigieren.

Es wird viel leistungsfähiger, wenn es verwendet wird, um [Makros](#) zu wiederholen.

Kopieren, Ausschneiden und Einfügen

In Vim werden diese Operationen anders gehandhabt als in fast jedem anderen modernen Editor oder Textverarbeitungsprogramm (`Ctrl-C` , `Ctrl-X` , `Ctrl-V`). Um dies zu verstehen, müssen Sie etwas über Register und Bewegungen wissen.

Hinweis: In diesem Abschnitt wird das Kopieren und Ausschneiden von Bildern oder das Riechen des Bereichs nicht behandelt, da dies nicht in den normalen Modus und die grundlegende Bearbeitung fällt.

Register

Vim verwendet das Konzept von *Registern* , um mit dem Verschieben von Text innerhalb des Programms selbst umzugehen. Windows verfügt zu diesem Zweck über eine einzige Zwischenablage, die einem einzelnen Register in Vim entspricht. Beim Kopieren, Ausschneiden und Einfügen in Vim gibt es Möglichkeiten, einen ähnlich einfachen Bearbeitungsworkflow zu verwenden (bei dem Sie nicht über Register nachdenken müssen), es gibt jedoch auch viel komplexere Möglichkeiten.

Ein Register dient der Eingabe / Ausgabe eines Befehls, indem dem Befehl `"` und einem Kleinbuchstaben vorangestellt wird.

Bewegungen

Eine Bewegung in Vim ist ein Befehl, der die Cursorposition an eine andere Stelle verschiebt. Beim Kopieren, Ausschneiden und Einfügen im Normalmodus sind die Möglichkeiten der Textauswahl für die Bewegung nur durch Ihr Bewegungswissen eingeschränkt. Einige werden unten veranschaulicht.

Kopieren und Schneiden

Die Grundbefehle zum Kopieren und Ausschneiden basieren auf `y` ('yank' zum Kopieren) und `d` ('delete', zum Ausschneiden). Die Ähnlichkeiten werden in der folgenden Tabelle angezeigt.

Befehl	:	Beschreibung
<code>y</code> {Bewegung}		Kopieren Sie den durch die Bewegung angegebenen Text in das Standardregister
<code>yy</code>		Kopieren Sie die aktuelle Zeile in das Standardregister, <i>linewise</i>
<code>Y</code>		Kopieren Sie die aktuelle Zeile in das Standardregister (Synonym für <code>yy</code>).
<code>"ayiW</code>		Kopieren Sie das Wort, auf dem sich der Cursor befindet, in das Register 'a'.
<code>20 "byy</code>		Kopiere zwanzig Zeilen, beginnend mit dem Cursor, in das Register 'b'.
<code>d</code> {Bewegung}		Schneiden Sie den durch die Bewegung angezeigten Text in das

Befehl	:	Beschreibung
		Standardregister
dd		Schneiden Sie die aktuelle Zeile in das Standardregister, <i>linewise</i>
D		Schneiden Sie vom Cursor bis zum Zeilenende in das Standardregister (KEIN Synonym für dd)
"adiw		Schneiden Sie das Wort, auf dem sich der Cursor befindet, in das Register 'a'.
20 "bdd		Schneiden Sie zwanzig Zeilen ab dem Cursor in das Register 'b'.

Hinweis: Wenn etwas kopiert oder *zeilenweise ausgeschnitten wird*, wird durch das unten dargestellte *Einfügeverhalten* Text vor oder hinter der aktuellen *Zeile* (anstelle des Cursors) *eingefügt*. Beispiele folgen zur Verdeutlichung.

Einfügen

Es gibt verschiedene Möglichkeiten, in Vim einzufügen, je nachdem, was Sie erreichen möchten.

Befehl	:	Beschreibung
p		Fügen Sie <i>nach</i> dem Cursor das ein, was sich im Standardregister befindet
P		Fügen Sie das, was sich im Standardregister befindet, <i>vor</i> dem Cursor ein
"ap		Fügen Sie den Inhalt des Registers 'a' nach dem Cursor ein
"cP		Fügen Sie den Inhalt des Registers 'c' vor dem Cursor ein

Wie führe ich ein wirklich einfaches Ausschneiden und Einfügen aus?

Wenn ich folgenden Text habe:

```
1 This line should be second
2 This line should be first
```

Ich kann das einfachste Ausschneiden und Einfügen durchführen, indem ich den Cursor irgendwo in Zeile 1 platziere und `ddp` eingebe. Hier sind die Ergebnisse:

```
1 This line should be first
2 This line should be second
```

Was ist passiert? `dd` ‚Cuts‘ die erste Zeile (linewise) in das Standardverzeichnis - die nur eine

Sache zu einer Zeit, wie die Windows - Zwischenablage enthalten - und `p` Pasten der Linie nach dem aktuellen, die aufgrund des Befehls `dd` gerade geändert hat.

Hier ist ein nicht ganz einfaches Beispiel. Ich muss ein paar Worte bewegen. (Dies ist nicht notwendig, aber Sie können dieses Prinzip auf größere Codeabschnitte anwenden.)

```
1 These words order out are of
```

Ich kann `w` wiederholen, um zum 'o' am Anfang von 'order' zu gelangen (oder `b`, wenn ich es einfach getippt und meinen Fehler erkannt habe).

Dann `"bitte"`, um "Ordnung" in Register "a" zu setzen.

Dann `w` auf den ,a' in ,sind' zu bekommen.

Im Anschluss würde ich `"bdaw"` eingeben, um "are" in das Register "b" zu setzen. Nun habe ich folgendes angezeigt:

```
1 These words out of
```

Um klar zu sein, jetzt ist 'order' im Register 'a' und 'are' ist im Register 'b' wie zwei separate Zwischenablagen.

Um die Wörter richtig zu ordnen, tippe ich `b`, um zum 'o' in 'out' zu gelangen, und dann `"bP to put"` sind "from register" b "vor" out ":

```
1 These words are out of
```

Jetzt gebe ich `A ein`, um zum Ende der Zeile zu gelangen, gefolgt von `Space Esc` (vorausgesetzt, es gibt kein Leerzeichen nach 'of') und `"ap"`, um 'order' dort zu platzieren, wo es hingehört.

```
1 These words are out of order
```

Fertigstellung

Mit der Vervollständigung können Sie die in einem Dokument verwendeten Wörter abgleichen. Wenn Sie ein Wort eingeben, stimmen Sie mit `Strg p` oder `Strg n` mit vorherigen oder nächsten ähnlichen Wörtern im Dokument überein.

Dies kann sogar mit dem `Strg-X-` Modus kombiniert werden, um ganze Zeilen zu vervollständigen. Geben Sie beispielsweise Folgendes ein:

```
This is an example sentence.
```

Gehen Sie dann zur nächsten Zeile und geben Sie den gleichen Satz ein:

```
Thi
```

und dann `Strg p` drücken, was zu folgenden Ergebnissen führt:

```
This
```

Wenn Sie sich noch im Einfügemodus befinden, drücken Sie `Strg x Strg p`, und das nächste Wort wird beendet.

```
This is
```

Fahren Sie mit `Strg x Strg p` fort, bis die gesamte Zeile abgeschlossen ist.

Wenn Sie wissen, dass Sie eine komplette Zeile abschließen möchten, geben Sie Folgendes ein:

```
This is an example sentence.
```

dann in der nächsten Zeile:

```
Thi
```

und drücken Sie `x Ctrl l`, um die Zeile zu vervollständigen.

Wenn der Abschluss eines Dateinamens erfolgt, kann mit `Strg x Strg f` dieses Verzeichnis vervollständigt werden. Art:

```
~/Deskt
```

dann drücke `Strg x Strg f` und:

```
~/Desktop
```

wird abgeschlossen (wenn an diesem Ort). `Ctrl x Ctrl f` kann dann wiederholt zum Auflisten der Dateien auf dem Desktop verwendet werden.

Befehle im Normalmodus (Bearbeitung) online lesen:

<https://riptutorial.com/de/vim/topic/5250/befehle-im-normalmodus--bearbeitung->

Kapitel 8: Befehlszeilenbereiche

Examples

Absolute Zeilennummern

Der folgende Befehl wird ausgeführt `:command` in den Zeilen 23 bis 56 :

```
:23,56command
```

Hinweis: Die Bereiche sind standardmäßig *inklusive* .

Relative Zeilennummern

Im folgenden Befehl beginnt der Bereich 6 Zeilen oberhalb der aktuellen Zeile und endet 3 Zeilen darunter:

```
:-6,+3command
```

Zeilenverknüpfungen

- `.` stellt *die aktuelle Zeile* dar , kann aber auch ganz weggelassen werden.
- `$` für *die letzte Zeile* .
- `%` für *den gesamten Puffer* , es ist eine Abkürzung für `1,$` .

Die beiden folgenden Befehle führen aus `:command` für jede Datei von der aktuellen Zeile bis zur letzten Zeile:

```
:.,$command  
:,$command
```

Der folgende Befehl führt Folgendes aus `:command` für den gesamten Puffer:

```
:%command
```

Markierungen

Der folgende Befehl führt Folgendes aus `:command` in jeder Zeile von der Zeile mit der Markierung `f` Manual bis zur Zeile mit der Markierung `t` Manual

```
:'f,'tcommand
```

Automatische Marken können ebenfalls verwendet werden:

```
:'<,'>command      " covers the visual selection
```

```
:{'','}command      " covers the current paragraph
:['',']command      " covers the last changed text
```

Siehe `:help mark-motions`.

Suche

Die folgenden Befehle führen Folgendes aus `:command` in jeder Zeile von der ersten Übereinstimmung `from` der ersten Übereinstimmung `to`:

```
:/from/,/to/command    " from next 'from' to next 'to'
:?from?/,to/command    " from previous 'from' to next 'to'
:?from?,?to?command    " from previous 'from' to previous 'to'
```

Siehe `:help search-commands`.

Linienversätze

Zeilenversätze können verwendet werden, um die Start- und Endlinie anzupassen:

```
:/foo/-,/bar/+4command  " from the line above next 'foo' to 4 lines below next 'bar'
```

Siehe `:help search-offset`.

Gemischte Bereiche

Es ist möglich, alle oben genannten Bereiche zu Ausdrucksbereichen zu kombinieren:

```
:1267,/foo/-2command
:{,command
:'f,$command
```

Seien Sie kreativ und vergessen Sie nicht zu lesen `:help cmdline-ranges`.

Befehlszeilenbereiche online lesen: <https://riptutorial.com/de/vim/topic/3383/befehlszeilenbereiche>

Kapitel 9: Bewegung

Examples

Suchen

Zu den Charakteren springen

`f {char}` - zum nächsten Vorkommen von `{char}` rechts vom Cursor in derselben Zeile springen

`F {char}` - zum nächsten Vorkommen von `{char}` links vom Cursor in derselben Zeile springen

`t {char}` - Bewegen Sie sich nach dem nächsten Vorkommen von `{char}` rechts neben dem Cursor in derselben Zeile

`T {char}` - Bewegt sich nach dem nächsten Vorkommen von `{char}` links vom Cursor in derselben Zeile

Zwischen den 'Ergebnissen' vorwärts / rückwärts springen über `;` und `.`

Weiterhin können Sie mit `/<searchterm> Enter` nach ganzen Wörtern suchen.

Suche nach Zeichenketten

`*` - zum nächsten Vorkommen des Wortes unter dem Cursor wechseln

`#` - Bewegen Sie sich zum vorherigen Vorkommen des Wortes unter dem Cursor

`/ searchterm Enter` bringt Sie zum nächsten Treffer (Vorwärtssuche). Wenn du verwendest `?` Statt `/` geht die Suche rückwärts.

Springe zwischen den Matches über `n` (weiter) und `N` (vorher).

Um Ihre vorherigen Suchen anzuzeigen / zu bearbeiten, geben Sie `/ ein` und drücken Sie die Aufwärtspfeiltaste `.`

Hilfreich sind auch diese Einstellungen: (Anmerkung `:se` ist gleich `:set`)

- `:se hls` hlls HighLightSearch, hebt alle Suchtreffer hervor; use `:noh` zum vorübergehenden Ausschalten der Suche / Markierung von Markierungen (`:set noh` oder `:set nohls` schaltet aus.)
- `:se is` oder `:set incs` schaltet die inkrementelle Suche ein, der Cursor springt automatisch zum nächsten Treffer. (`:se nois` schaltet sich aus.)
- `:se ic IgnoreCase`", schaltet die Groß- / Kleinschreibung aus. (`:se noic` schaltet sich wieder ein.)

- `:se scs SmartCaSe`, kann verwendet werden, wenn `IgnoreCase` eingestellt ist; macht Fall (in) Empfindlichkeit **smart!** zB `/the` für die Suche `the` , `The` , `ThE` usw. während `/The` nur für aussehen `The` .

Grundlegende Bewegung

Bemerkungen

- Jede Bewegung kann nach einem Bedienerbefehl verwendet werden, sodass der Befehl den Text bearbeitet, der in der Reichweite der Bewegung enthalten ist.
- Bewegungen können, genau wie Bedienerbefehle, eine Zählung enthalten, so dass Sie sich beispielsweise um `zwei` Reihen bewegen können.

Pfeile

In Vim funktionieren die normalen Pfeiltasten (`←` `↓` `↑` `→`) wie erwartet. Doch für die Touch-typers, dann ist es einfacher , die `h j k l` alternativen Schlüssel zu verwenden. Bei einer typischen Tastatur befinden sie sich nebeneinander in derselben Reihe und sind mit der rechten Hand leicht zugänglich. Die Mnemonic-Technik, um sich zu erinnern, welche davon welche ist, geht folgendermaßen

- `h` / `l` - diese befinden sich "am weitesten links / rechts" unter den vier Buchstaben der Tastatur, sie entsprechen also "links / rechts gehen";
- `j` - Das Kleinbuchstabe "j" wird wie typische Pfeile unter den typischen Buchstaben "nach unten" gesetzt - es entspricht also "nach unten".
- `k` - Umgekehrt wird bei Kleinbuchstaben "k" der "Aufstieg" über typische Buchstaben wie ein kleiner Zeiger "angehoben" - es entspricht also "Aufsteigen".

Grundbewegungen

Alle folgenden Befehle sollten im **normalen Modus** ausgeführt werden .

Befehl	Beschreibung
<code>h</code> oder <code>links</code>	<code>[count]</code> Zeichen nach links gehen
<code>j</code> oder <code>runter</code>	<code>go [count]</code> Zeichen unten
<code>k</code> oder <code>höher</code>	<code>go [count]</code> Zeichen oben
<code>l</code> oder <code>richtig</code>	<code>[count]</code> Zeichen nach rechts gehen
<code>gg</code>	gehe in die erste Zeile oder <code>[count]</code> 'th Zeile, falls gegeben

Befehl	Beschreibung
H	Gehen Sie zur ersten Zeile im sichtbaren Bildschirm
M	Gehen Sie zur mittleren Zeile im sichtbaren Bildschirm
L	Gehen Sie zur letzten Zeile im sichtbaren Bildschirm
G	gehe in die letzte Zeile oder <code>[count]</code> 'th Zeile, falls gegeben
Zuhause oder 0	gehe zum ersten Zeichen der Zeile
^	gehe zum ersten nicht leeren Zeichen der Zeile
+	Gehen Sie eine Zeile nach unten zum ersten nicht leeren Zeichen
-	gehe eine Zeile hoch zum ersten nicht leeren Zeichen
\$ oder Ende	gehe zum Ende der Zeile (wenn <code>[count]</code> angegeben ist, gehe <code>[count - 1]</code> Zeilen nach unten)
	gehe zum <code>[count]</code> -ten Zeichen oder zum Anfang der Zeile, wenn <code>count</code> nicht angegeben ist
f {char}	gehe zu <code>[count]</code> th Vorkommen von {char} nach rechts <i>inklusive</i>
F {char}	gehe zu <code>[count]</code> th Vorkommen von {char} nach links <i>inklusive</i>
t {char}	Gehen Sie zu <code>[count]</code> 'th Vorkommen von {char} zum rechten <i>Exklusivwert</i>
T {char}	Gehen Sie zu <code>[count]</code> 'th Vorkommen von {char} zum <i>exklusiven Link</i>
;	Wiederholen Sie die letzten f , t , F oder T <code>[count]</code> -Zeiten
.	Wiederholen Sie den letzten f , t , F oder T in der entgegengesetzten Richtung <code>[count]</code> mal
w	gehe zum Anfang des nächsten Wortes
b	gehe zum Anfang des vorherigen Wortes
e	gehe zum Ende des nächsten Wortes
ge	zum Ende des vorherigen Wortes gehen
%	zu passenden Paaren gehen, zB <code>()</code> , <code>[]</code> , <code>{}</code> , <code>/* */</code> oder <code>#if</code> , <code>#ifdef</code> , <code>#else</code> , <code>#elif</code> , <code>#endif</code>
{ }	vorheriger / nächster Absatz
[{ }]	Anfang / Ende des Blocks

Befehl	Beschreibung
'{verkohlen}	Gehe zur Markierung (Markiere mit <code>m {char}</code>)
<CB> <CF>	vorherige / nächste Seite
<CO> <CI>	Gehe zurück oder vorwärts in der "Sprungliste" (erfordert <code>jumplist</code> Funktion, siehe <code>:help jumps</code>)

Hinweis: `b` , `e` und `w` betrachten ein Wort standardmäßig als Buchstaben, Zahlen und Unterstriche (dies kann mit der Einstellung `iskeyword` konfiguriert werden). Jede dieser Optionen kann auch groß geschrieben werden, sodass sie alles überspringen, das nicht auch Leerzeichen ist.

Hinweis: Vim erkennt zwei Arten von Bewegung: Operator Bewegung (`:help movement`) und Sprünge (`:help jumplist`). Bewegungen, wie sie mit `g` (`gg` , `G` , `g,`) ausgeführt werden, gelten als Sprünge, ebenso wie Änderungen. Änderungen erhalten ihre eigene Jumplist, die wie oben erwähnt über `g`, und `g;` navigierbar ist `g;` (Siehe `:help changelist`). Sprünge werden von Vim nicht als Bewegungsbefehle behandelt

Wenn Sie sich über Zeilen nach oben oder unten bewegen, behält der Cursor seine Spalte wie erwartet. Wenn die neue Zeile zu kurz ist, springt der Cursor zum Ende der neuen Zeile. Wenn sich die Spalte außerhalb des Zeilenendes befindet, wird der Cursor am Zeilenende angezeigt. Die anfängliche Spaltennummer bleibt erhalten, bis eine Aktion zum Ändern (z. B. Bearbeiten von Text oder explizites Verschieben von Spalten) ausgeführt wird.

Wenn die Länge einer Zeile die Breite des Bildschirms überschreitet, wird der Text umbrochen (in den Standardeinstellungen kann dieses Verhalten konfiguriert werden). Fügen Sie `g` vor dem üblichen Befehl ein, um sich durch die auf dem Bildschirm angezeigten Zeilen und nicht durch Zeilen in der Datei zu bewegen. Zum Beispiel bewegt `gj` den Cursor an die Position, die eine Zeile unter seiner aktuellen Position angezeigt wird, auch wenn sich diese in derselben Zeile der Datei befindet.

Muster suchen

Vim unterstützt die Verwendung regulärer Ausdrücke beim Durchsuchen einer Datei.

Das Zeichen, das angibt, dass Sie eine Suche durchführen möchten, ist `/` .

Die einfachste Suche, die Sie durchführen können, ist die folgende

```
/if
```

Dadurch wird die gesamte Datei nach allen Instanzen von `if` durchsucht. Bei unserer Suche `if` handelt es sich jedoch eigentlich um einen regulären Ausdruck, der mit jedem Vorkommen des Wortes übereinstimmt, `if` die Wörter innerhalb anderer Wörter handelt.

Zum Beispiel würde unsere Suche sagen, dass die folgenden Wörter zu unserer Suche passen: `if` , `spiffy` , `endif` usw.

Wir können kompliziertere Suchen durchführen, indem Sie kompliziertere reguläre Ausdrücke verwenden.

Wenn unsere Suche war:

```
/\<if\>
```

dann würde unsere Suche nur dann exakte Übereinstimmungen mit dem vollständigen Wort zurückgeben, `if`. Das oben genannte "`spiffy`" und "`endif`" würde von der Suche nur `if`, `if`.

Wir können auch Bereiche verwenden. Gegeben eine Datei:

```
hello1
hello2
hello3
hello4
```

Wenn Sie nach den Zeilen suchen möchten, die "Hallo" gefolgt von einer Ziffer zwischen 1 und 3 enthalten, würden wir sagen:

```
/hello[1-3]
```

Ein anderes Beispiel:

```
/(?:\d*\.)?\d+
```

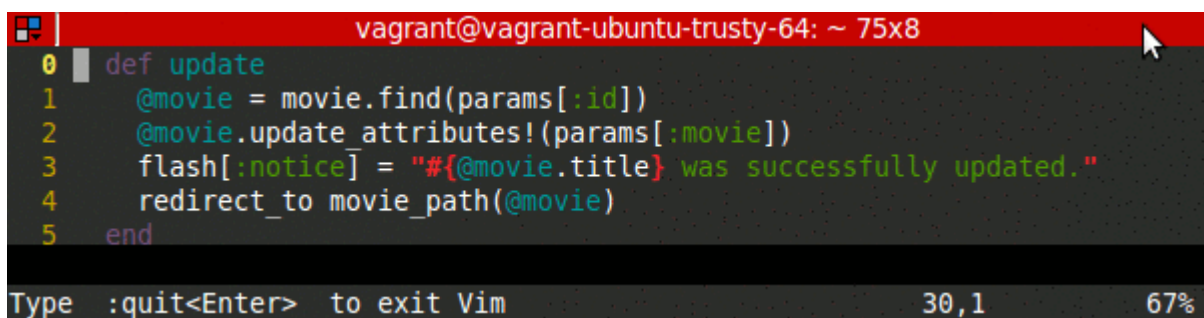
würde alle Integer- und Dezimalzahlen in der Datei finden.

Zum Anfang eines bestimmten Wortes navigieren

Beim Bearbeiten von Text ist es üblich, zu einem bestimmten Wort auf dem Bildschirm zu navigieren. In diesen Beispielen untersuchen wir, wie wir zu dem `updated` Wort navigieren können. Aus Gründen der Beständigkeit in allen Beispielen wollen wir auf dem ersten Buchstaben des Wortes landen.

Mid-Screen-Sprung

M \$ B



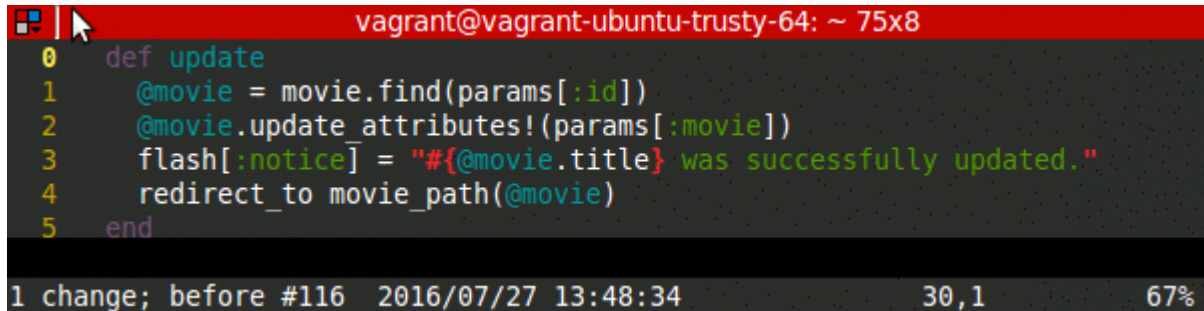
```
vagrant@vagrant-ubuntu-trusty-64: ~ 75x8
0 | def update
1 |   @movie = movie.find(params[:id])
2 |   @movie.update_attributes!(params[:movie])
3 |   flash[:notice] = "#{@movie.title} was successfully updated."
4 |   redirect_to movie_path(@movie)
5 | end
Type :quit<Enter> to exit Vim 30,1 67%
```

Dieser Ansatz ist schnell und verwendet nur 3 Tastenanschläge. Der Nachteil ist jedoch, dass es

nicht sehr allgemein ist, da unsere Ziellinie normalerweise nicht in der Mitte des Bildschirms liegt. Trotzdem ist es eine nützliche Bewegung, wenn Sie weniger granulare Bewegungen ausführen.

Mit einer Zählung

3j f u ; ;

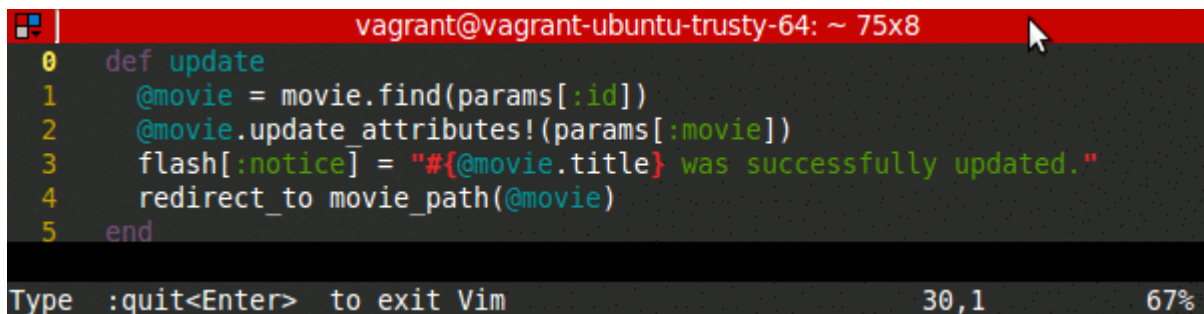


```
vagrant@vagrant-ubuntu-trusty-64: ~ 75x8
0 def update
1   @movie = movie.find(params[:id])
2   @movie.update_attributes!(params[:movie])
3   flash[:notice] = "#{@movie.title} was successfully updated."
4   redirect_to movie_path(@movie)
5 end
1 change; before #116 2016/07/27 13:48:34 30,1 67%
```

Auf den ersten Blick erscheint dies aufgrund der Anzahl der Tastatureingaben als Rückschritt vom ersten Ansatz. Da wir hier jedoch eine Zählung anstelle von `M` verwenden, ist dies flexibler. Wir können schnell die richtige Anzahl ermitteln, die verwendet werden soll, wenn die `relative` number aktiviert ist. Um zum Zielwort zu gelangen, verwenden Sie `f` in Kombination mit `;`; kann überraschend effektiv sein - und sicherlich besser als wiederholt `w` drücken. Wenn Sie Ihr Ziel mit überschießen `;`, Können Sie gehen nach hinten mit.

Explizite Suche

/ up Geben Sie n n ein



```
vagrant@vagrant-ubuntu-trusty-64: ~ 75x8
0 def update
1   @movie = movie.find(params[:id])
2   @movie.update_attributes!(params[:movie])
3   flash[:notice] = "#{@movie.title} was successfully updated."
4   redirect_to movie_path(@movie)
5 end
Type :quit<Enter> to exit Vim 30,1 67%
```

Die Navigation über `/` kann sehr leistungsfähig sein. Wir können oft direkt zu unserem Zielwort springen, indem wir es eingeben. Hier tippen wir nur die ersten beiden Zeichen ein, in der Hoffnung, dass es eindeutig zu unserem Wort passt. Leider gibt es mehrere Matches, aber wir können mit `n` schnell zum nächsten Match springen.

Implizite Suche

/ Y Raum Geben Sie w

```
vagrant@vagrant-ubuntu-trusty-64: ~ 75x8
0 def update
1   @movie = movie.find(params[:id])
2   @movie.update attributes!(params[:movie])
3   flash[:notice] = "#{@movie.title} was successfully updated."
4   redirect_to movie_path(@movie)
5 end

/up 30,1 67%
```

In einigen Fällen ist es möglicherweise effizienter, in die *Nähe* unseres Ziels zu springen, als direkt darauf zu zielen. Hier stellen wir fest, dass sich neben dem Ziel ein selten vorkommender Buchstabe `y` befindet. Wir können unserem Suchbegriff ein Leerzeichen hinzufügen, um die Wahrscheinlichkeit zu verringern, dass wir einen anderen `y` Charakter treffen. Dies kann auch mit `f {char}` wie im Beispiel *Using a count verwendet werden*.

Verwenden von Markierungen zum Bewegen

Marken sind wie Lesezeichen; Sie helfen Ihnen, Orte zu finden, an denen Sie bereits waren.

TLDR

Setzen Sie sie mit `m{a-zA-Z}` in den Normalmodus und springen Sie mit `'{a-zA-Z}` (einfaches Anführungszeichen) oder ``{a-zA-Z}` (Backtick) im Normal- oder visuellen Modus. Kleinbuchstaben stehen für Marken innerhalb eines Puffers, Großbuchstaben und Ziffern sind global. Sehen Sie Ihre aktuell gesetzten Marken mit `:marks` und weitere Informationen finden Sie unter `:help mark`.

Eine Marke setzen

Vims integrierte Hilfe sagt:

```
m{a-zA-Z}          Set mark {a-zA-Z} at cursor position (does not move
                    the cursor, this is not a motion command).
```

Die Markierung verfolgt, an welcher Zeile und Spalte sie platziert wurde. Es gibt keine visuelle Bestätigung, dass eine Marke gesetzt wurde oder ob eine Marke einen vorherigen Wert hatte und überschrieben wurde.

Springe zu einer Marke

Vims integrierte Hilfe sagt:

```
Jumping to a mark can be done in two ways:
1. With ` (backtick):    The cursor is positioned at the specified location
                        and the motion is exclusive.
2. With ' (single quote): The cursor is positioned on the first non-blank
```

```
character in the line of the specified location and
the motion is linewise.
```

Backtick verwendet die Spaltenposition, nicht jedoch das einfache Anführungszeichen. Der Unterschied zwischen ihnen erlaubt es Ihnen einfach, die Spaltenposition Ihrer Marke zu ignorieren, wenn Sie möchten.

Sie können im visuellen Modus zusätzlich zum normalen Modus zwischen nicht-globalen Marken wechseln, um Text basierend auf Marken auswählen zu können.

Globale Markierungen

Globale Markierungen (Großbuchstaben) ermöglichen das Springen zwischen Dateien. Das bedeutet, wenn zum Beispiel die Markierung `A` in `foo.txt`, dann von `bar.txt` (irgendwo in meinem Dateisystem), wenn ich zur Markierung `A` springe, wird mein aktueller Puffer durch `foo.txt`. Vim fordert Sie zum Speichern der Änderungen auf.

Ein Sprung zu einer Marke in einer anderen Datei wird **nicht** als Bewegung betrachtet, und visuelle Auswahlen (unter anderem) funktionieren nicht wie ein Sprung zu Marken innerhalb eines Puffers.

Um zur vorherigen Datei (in diesem Fall `bar.txt`) zurückzukehren, verwenden Sie `:b[uffer] #` (d. `:b[uffer] # :b#` oder `:buffer#`).

Hinweis:

Besondere Kennzeichen

Es gibt bestimmte Markierungen, die Vim automatisch setzt (die Sie selbst überschreiben können, aber wahrscheinlich nicht müssen).

Zum Beispiel (umschrieben von Vims Hilfe):

```
`[` and `]`: jump to the first or last character of the previously changed or
yanked text. {not in Vi}

`<` and `>`: jump to the first or last line (with ```) or character (with
<code>`</code>) of the last selected Visual area in the current
buffer. For block mode it may also be the last character in the
first line (to be able to define the block). {not in Vi}.
```

Mehr von Vims eingebauter Hilfe:

```
`` ``
To the position before the latest jump, or where the
last "m'" or "m`" command was given. Not set when the
:keepjumps command modifier was used.
Also see restore-position.
```

'"`"	<p>To the cursor position when last exiting the current buffer. Defaults to the first character of the first line. See last-position-jump for how to use this for each opened file.</p> <p>Only one position is remembered per buffer, not one for each window. As long as the buffer is visible in a window the position won't be changed.</p> <p>{not in Vi}.</p>
'.`.	<p>To the position where the last change was made. The position is at or near where the change started. Sometimes a command is executed as several changes, then the position can be near the end of what the command changed. For example when inserting a word, the position will be on the last character.</p> <p>{not in Vi}</p>
'"`"	<p>To the cursor position when last exiting the current buffer. Defaults to the first character of the first line. See last-position-jump for how to use this for each opened file.</p> <p>Only one position is remembered per buffer, not one for each window. As long as the buffer is visible in a window the position won't be changed.</p> <p>{not in Vi}.</p>
'^`^	<p>To the position where the cursor was the last time when Insert mode was stopped. This is used by the gi command. Not set when the :keepjumps command modifier was used. {not in Vi}</p>

Darüber hinaus sind die Zeichen (,) , { und } Marken, die an die gleiche Stelle springen wie ihre Normalmodusbefehle - das heißt, '}' führt im Normalmodus dasselbe aus wie } .

Springe zu einer bestimmten Zeile

Um zu einer bestimmten Zeile mit Doppelpunktnummer zu springen. Um zur ersten Zeile einer Datei zu springen, verwenden Sie

```
:1
```

Um zu Zeile 23 zu springen

```
:23
```

Bewegung online lesen: <https://riptutorial.com/de/vim/topic/1117/bewegung>

Kapitel 10: Bewegungen und Textobjekte

Bemerkungen

Ein Textobjekt in Vim ist eine weitere Möglichkeit, einen Textabschnitt anzugeben, der bearbeitet werden soll. Sie können mit Bedienern oder im visuellen Modus anstelle von Bewegungen verwendet werden.

Examples

Ändern des Inhalts einer String- oder Parameterliste

Nehmen wir an, Sie haben diese Codezeile:

```
printf("Hello, world!\n");
```

Sagen Sie nun, Sie möchten den Text in "Programm beenden" ändern.

Befehl	Puffer	Gedächtnis
ci"	printf(" ");	c hange i n den ".
Program exiting.\n<esc>	printf("Program exiting.\n");	

Bewegungen und Textobjekte online lesen: <https://riptutorial.com/de/vim/topic/4107/bewegungen-und-textobjekte>

Kapitel 11: Bitten Sie beim Erstellen einer neuen Datei um das Erstellen nicht vorhandener Verzeichnisse

Einführung

Wenn Sie eine neue Datei bearbeiten: `vim these/directories/dont/exist/newfile` können Sie die Datei nicht speichern, da das Verzeichnis, in dem `vim` versucht, zu speichern, nicht vorhanden ist.

Examples

Fordern Sie auf, Verzeichnisse mit: `w` zu erstellen, oder erstellen Sie sie mit: `w!`

Dieser Code fordert Sie auf, das Verzeichnis mit `:w` zu erstellen, oder tun Sie es einfach mit `:w!` :

```
augroup vimrc-auto-mkdir
  autocmd!
  autocmd BufWritePre * call s:auto_mkdir(expand('<afile>:p:h'), v:cmdbang)
  function! s:auto_mkdir(dir, force)
    if !isdirectory(a:dir)
      \   && (a:force
      \       || input("'" . a:dir . "' does not exist. Create? [y/N]") =~? '^y\[es]$')
      call mkdir(iconv(a:dir, &encoding, &termencoding), 'p')
    endif
  endfunction
augroup END
```

Bitten Sie beim Erstellen einer neuen Datei um das Erstellen nicht vorhandener Verzeichnisse
online lesen: <https://riptutorial.com/de/vim/topic/9470/bitten-sie-beim-erstellen-einer-neuen-datei-um-das-erstellen-nicht-vorhandener-verzeichnisse>

Kapitel 12: Code automatisch formatieren

Examples

Im normalen Modus:

Im normalen Modus:

gg *nach oben* = dann G

Code automatisch formatieren online lesen: <https://riptutorial.com/de/vim/topic/7931/code-automatisch-formatieren>

Kapitel 13: Dateityp-Plugins

Examples

Wo kann ich benutzerdefinierte Dateityp-Plugins einfügen?

Dateityp-Plugins für `foo` Dateitypen werden in dieser Reihenfolge beschafft:

1. `$HOME/.vim/ftplugin/foo.vim` . Seien Sie vorsichtig mit dem, was Sie in diese Datei `$VIMRUNTIME/ftplugin/foo.vim` , da sie möglicherweise von `$VIMRUNTIME/ftplugin/foo.vim` - unter Windows wird es stattdessen `$HOME/vimfiles/ftplugin/foo.vim`
2. `$VIMRUNTIME/ftplugin/foo.vim` . Wie alles unter `$VIMRUNTIME` sollte diese Datei nicht geändert werden.
3. `$HOME/.vim/after/ftplugin/foo.vim` . Diese Datei ist der letzte, also der ideale Ort für *Ihre* dateitypspezifischen Einstellungen.

Dateityp-Plugins online lesen: <https://riptutorial.com/de/vim/topic/7734/dateityp-plugins>

Kapitel 14: Der Punktoperator

Examples

Grundlegende Verwendung

Der Punktoperator wiederholt die zuletzt ausgeführte Aktion, zum Beispiel:

Datei `test.tx`

```
helo, World!  
helo, David!
```

(Cursor bei [1] [1])

`cwHello<Esc>` nun ein `cwHello<Esc>` (Word- `helo` in `Hello helo`)

Nun sieht der Puffer so aus:

```
Hello, World!  
helo, David!
```

(Cursor bei [1] [5])

Nach der Eingabe von `j_` der Cursor auf [2] [1].

Nun geben Sie das ein `.` und die letzte Aktion wird erneut ausgeführt:

```
Hello, World!  
Hello, David!
```

(Cursor bei [2] [5])

Einzug setzen

Dies ist sehr nützlich, wenn Sie den Code einziehen möchten

```
if condition1  
if condition2  
# some commands here  
endif  
endif
```

Bewegen Sie den Cursor in die 2. Zeile, dann `>>` , der Code wird nach rechts eingerückt.

Jetzt können Sie Ihre Aktion wiederholen, indem Sie mit der 3. Zeile fortfahren und dann drücken `.` zweimal wird das Ergebnis sein

```
if condition1  
    if condition2
```

```
    # some commands here  
endif  
endif
```

Der Punktoperator online lesen: <https://riptutorial.com/de/vim/topic/3665/der-punktoperator>

Kapitel 15: Ex von der Kommandozeile aus verwenden

Examples

Ersetzung von der Befehlszeile aus

Wenn Sie vim auf ähnliche Weise wie sed möchten, können Sie das Flag `-c` verwenden, um einen Ex-Befehl von der Befehlszeile aus auszuführen. Dieser Befehl wird automatisch ausgeführt, bevor Ihnen die Datei angezeigt wird. Zum Beispiel, um `foo` durch einen `bar` zu ersetzen:

```
vim file.txt -c "s/foo/bar"
```

Dadurch wird die Datei geöffnet, wobei alle Instanzen von `foo` durch `bar` . Wenn Sie Änderungen an der Datei vornehmen möchten, *ohne* manuell speichern zu müssen, können Sie mehrere ex-Befehle ausführen und den letzten Befehl schreiben und beenden. Zum Beispiel:

```
vim file.txt -c "s/foo/bar" -c "wq"
```

Wichtige Notiz:

Sie können *nicht* mehr Ex - Befehle durch eine Bar getrennt laufen | . Zum Beispiel

```
vim file.txt -c "s/foobar | wq"
```

Ist *nicht* richtig; Es kann jedoch gemacht werden, wenn Sie `ex` .

```
ex -c ":%s/this/that/g | wq" file.txt
```

Ex von der Kommandozeile aus verwenden online lesen:

<https://riptutorial.com/de/vim/topic/6819/ex-von-der-kommandozeile-aus-verwenden>

Kapitel 16: Falten

Bemerkungen

Durch das Falten werden mehrere Textzeilen reduziert und als einzelne Zeile angezeigt. Es ist nützlich, um Teile eines Dokuments auszublenden, die für die aktuelle Aufgabe als unwichtig angesehen werden. Falten ist eine reine Änderung des Dokuments: Die gefalteten Linien sind unverändert vorhanden.

Eine Falte ist hartnäckig. Einmal erstellt, kann eine Falte geöffnet und geschlossen werden, ohne dass sie neu erstellt werden muss. Wenn sie geschlossen sind, können Falten über eine Zeile verschoben oder gezogen werden, als ob sie eine einzelne Zeile wären, auch wenn die darunter liegende Operation den gesamten Text unterhalb der Falte bearbeitet

Examples

Faltmethode konfigurieren

`:set foldmethod={method}` legt die fold-Methode für das aktuelle Fenster fest. Dies bestimmt, wie Falten innerhalb dieses Fensters bearbeitet werden. Gültige Optionen für "Methode" sind:

- `manual` (Falten werden vom Benutzer manuell erstellt und zerstört)
- `indent` (Falten werden für Zeilen mit gleichem Einzug erstellt)
- `marker` (Teilzeichenfolgenmarkierungen werden verwendet, um den Anfang und das Ende einer Falte anzuzeigen)
- `syntax` (Syntax-Hervorhebungselemente definieren Falten)
- `expr` (ein Vimscript-Ausdruck wird pro Zeile ausgewertet, um die `expr` festzulegen)
- `diff` (Textänderung wird nicht geändert, wenn eine Diff-Ansicht gefaltet ist)

Die Standardeinstellung ist `manual`.

Manuelles Erstellen einer Falte

- `zf{motion}` erstellt eine Falte, die den Text abdeckt, den "Bewegung" abdecken würde.
- `{count}zF` erstellt eine Falzung, die "Count" `{count}zF` abdeckt.
- `{range}fo[ld]` erstellt eine Falte für die Linien im angegebenen Bereich.

Alle drei Befehle sind nur gültig, wenn `foldmethod` auf `manual` oder `marker`. Bei der bisherigen Falzmethode werden die neu erstellten Falze sofort geschlossen.

Falten öffnen, schließen und umschalten

- `zo` öffnet eine Falte unter dem Cursor.
- `zO` öffnet rekursiv alle Falten unter dem Cursor.
- `zc` schließt eine Falte unter dem Cursor.

- `zC` schließt alle Falten unter dem Cursor rekursiv.
- `za` schaltet eine Falte unter dem Cursor um (eine geschlossene Falte wird geöffnet, eine offene Falte wird geschlossen).
- `zM` schließt alle Falten im Puffer.
- `zR` öffnet alle Falten im Puffer.
- `zm` schließt eine Falzstufe im Puffer.
- `zr` öffnet eine Faltebene im Puffer.

Die Zeile mit dem Cursor anzeigen

`zv` sicher, dass die Zeile mit dem Cursor nicht gefaltet wird. Die minimale Anzahl von Falten, die zum Freilegen der Cursorlinie erforderlich ist, wird geöffnet.

C-Blöcke falten

Dies ist unser Puffer:

```
void write_buffer(size_t size, char ** buffer)
{
    char * buf = *buffer;
    size_t iter;
    for(iter = 0; iter < size; iter++)
    {
        putchar(*(buf + iter));
    }
}
```

Der Cursor steht auf [1] [1] ([Zeile] [Spalte]). Bewegen Sie den Cursor zur Curl-Klammer der for-Schleife:

`/for<Enter>j` Cursor [6] [2].

`zf%` nun `zf%` (Faltung erstellen, zur entsprechenden Klammer wechseln). Sie haben erfolgreich die erste Faltung erstellt.

`:2<Enter>_` nun ein `:2<Enter>_`, der Cursor steht jetzt auf [2] [1] und `zf%`: Der gesamte Funktionskörper wird gefaltet.

Sie sind in der Lage, alle Faltungen öffnen Sie gerade erstellt mit `zo` und wieder schließen sie mit `zC`.

Falten online lesen: <https://riptutorial.com/de/vim/topic/3791/falten>

Kapitel 17: Fenster teilen

Syntax

- `:split <file>`
- `:vsplit <file>`
- `:sp <-` Abkürzung für Split
- `:vsp <-` Abkürzung für vsplit

Bemerkungen

Beim Aufruf über die Befehlszeile können mehrere Dateien im Argument angegeben werden, und vim erstellt für jede Datei einen Split. Beim Aufruf aus dem Ex-Modus kann pro Aufruf des Befehls nur eine Datei geöffnet werden.

Examples

Öffnen Sie mehrere Dateien in Splits von der Befehlszeile aus

Horizontal

```
vim -o file1.txt file2.txt
```

Vertikal

```
vim -O file1.txt file2.txt
```

Sie können optional die Anzahl der zu öffnenden Splits angeben. Das folgende Beispiel öffnet zwei horizontale `file3.txt` und lädt `file3.txt` in einen Puffer:

```
vim -o2 file1.txt file2.txt file3.txt
```

Ein neues Splitfenster öffnen

Sie können einen neuen Split in Vim im *Normalmodus* mit den folgenden Befehlen öffnen:

Horizontal:

```
:split <file name>  
:new
```

Vertikal:

```
:vsplit <file name>
```

```
:vnew
```

split öffnet die Datei in einem neuen Split am oberen oder linken Bildschirmrand (oder dem aktuellen Split) `:sp` und `:vs` sind praktische Verknüpfungen.

new öffnet einen leeren Split

Ändern der Größe eines Split oder vsplit

Manchmal möchten Sie vielleicht die Größe eines Splits oder eines vsplit ändern.

Um die Größe der aktuell aktiven Aufteilung zu `:resize <new size>`, verwenden Sie `:resize <new size>`. `:resize 30` beispielsweise die `:resize 30`, wird der Split 30 Zeilen groß.

Um die Größe des aktuell aktiven vsplit zu `:vertical resize <new size>`, verwenden Sie `:vertical resize <new size>`. `:vertical resize 80` würde das vsplit 80 Zeichen breit machen.

Tastenkombinationen

- `Strg + w` und `+` vergrößern die Größe des geteilten Fensters
- `Strg + w` und `-` verkleinern das geteilte Fenster
- `Strg + w` und `=` setzen die geteilten Fenster gleich groß

Schließen Sie alle Splits außer dem aktuellen

Normaler Modus

```
Strg-w o
```

Ex-Modus

```
:only
```

oder kurz

```
:on
```

Open-Split-Fenster verwalten (Tastenkombinationen)

Nachdem Sie ein geteiltes Fenster in vim geöffnet haben (wie viele Beispiele unter diesem Tag zeigen), möchten Sie wahrscheinlich Fenster schnell steuern. So steuern Sie geteilte Fenster mithilfe von Tastenkombinationen.

Bewegen Sie sich zum Teilen von oben / unten:

- `Strg + w` und `k`
- `Strg + w` und `j`

Zum Teilen nach links / rechts wechseln:

- Strg + w und h
- Strg + w und l

Verschiebe, um oben / unten zu teilen (wrap)

- Strg + w und w

Neues leeres Fenster erstellen:

- Strg + w und n -oder-: neu

Neuen Split horizontal / vertikal erstellen:

- Strg + W , s (Großbuchstabe)
- Strg + W , v (Kleinschreibung)

Machen Sie die aktuell aktive Aufteilung auf dem Bildschirm:

- Strg + w und o -oder-: ein

Zwischen den Spalten wechseln

Um sich nach links zu teilen, verwenden Sie <Cw><Ch>

Verwenden Sie <Cw><Cj> um zum Split zu <Cw><Cj>

Um nach rechts zu teilen, verwenden Sie <Cw><Ck>

Um zum Split zu gelangen, verwenden Sie <Cw><Cl>

Vernünftige Spaltöffnung

Es ist eine bessere Erfahrung, die Aufteilung unten und rechts zu öffnen

setze es mit

```
set splitbelow  
set splitright
```

Fenster teilen online lesen: <https://riptutorial.com/de/vim/topic/1705/fenster-teilen>

Kapitel 18: Get: help (mit dem integrierten Handbuch von Vim)

Einführung

Das integrierte Handbuch von Vim ist die maßgebliche Informationsquelle und Dokumentation zu allen Vim-Funktionen, einschließlich Konfigurationen, integrierten Funktionen und sogar Vimscript. Wenn Sie nicht wissen, wie man es durchschaut, sind Sie zwar nicht das Anfängerfreundlichste, können aber finden, was Sie brauchen.

Starten Sie die Suche, indem Sie Folgendes ausführen `:help`, `:help [subject]` oder `:help :help`.

Syntax

- `:h[elp] [keyword]`

Parameter

Parameter	Einzelheiten
keyword	Konfiguration, Funktionsname oder ein beliebiges anderes Schlüsselwort mit Bedeutung für Vim. Schlüsselwörter mit einem führenden Doppelpunkt : Suche nach Vim-Befehlen; Beispiel <code>:help :split</code> liefert den Befehl zum Teilen von Fenstern, und <code>:help split</code> liefert die Vimscript-Funktion <code>split()</code> .

Examples

Erste Schritte / Navigieren in den Hilfedateien

Führen Sie von überall in Vim aus `:help :help`. Dies öffnet ein horizontal geteiltes Fenster mit der Manpage zum Befehl `:help`. `:help` führt Sie zum Inhaltsverzeichnis des Handbuchs.

Die Hilfedateien von Vim können wie normale Dateien navigiert werden (Sie können wie in normalen Dateien mit / nach Schlüsselwörtern suchen) und zusätzlich mit Tags verknüpft werden. Springen Sie mit `CTRL-J` zum Ziel eines Tags.

Tags sind Wörter von Rohr umgeben | Zeichen. Die Versionen 7.3 und höher "verdecken" diese Pipe-Zeichen (`:help conceal`) und heben sie hervor.

Die Seite Inhaltsverzeichnis zeigt beispielsweise Folgendes. Alle blau hervorgehobenen Wörter sind Tags und werden von Pipe-Zeichen umgeben. `CTRL-J` mit dem Cursor auf `quickref`, gelangen Sie zu einer nützlichen Seite mit einer Liste von Tags zu nützlichen Vim-Funktionen.

```
----- doc-file-list Q_ct
BASIC:
quickref      Overview of the most common commands you will use
tutor         30 minutes training course for beginners
copying       About copyrights
iccf          Helping poor children in Uganda
sponsor       Sponsor Vim development, become a registered Vim user
www           Vim on the World Wide Web
bugs          Where to send bug reports

USER MANUAL: These files explain how to accomplish an editing task.

usr_toc.txt   Table Of Contents
```

Das Handbuch durchsuchen

`:help [subject]` versucht, die beste Übereinstimmung für das von Ihnen angegebene Argument zu finden. Das Argument "kann Platzhalterzeichen wie `*` `?` Und `[az]` (beliebiger Buchstabe) enthalten.

Sie können die Befehlszeilenvervollständigung von Vim zusätzlich mit `CTRL+D` :

`:help spli<Ctrl-D>` zeigt eine Liste von `spli` , die mit dem Musterspli `spli` , einschließlich `split()` und `:split` .

Um nach `Ctrl` basierten Befehlen wie `Ctrl-V` zu suchen, geben Sie `Ctrl-V` :

`:help ^v` mit einem Buchstabierzeichen oder, genauer gesagt,

`:help i_^V` , um im Einfügemodus Hilfe zu `Ctrl-V` zu erhalten.

Wie Sie sehen, hat vim eine Nomenklatur für seine Hilfethemen. Beispielsweise werden Optionen in Anführungszeichen gesetzt (siehe `:h 'sw'`), Befehle beginnen mit einem Doppelpunkt (siehe `:h :split`), Funktionen enden mit leeren Klammern (siehe `:h split()`), Einfügemoduszuordnungen beginnen mit `i_` , Befehl Moduszuordnungen beginnen mit `c_` usw., mit Ausnahme von normalen Moduszuordnungen, die kein Präfix haben.

Suchbegriff	Hilfeseite
<code>:help textwidth</code>	Konfiguration für Zeilenlänge / Textbreite
<code>:help normal</code>	<code>:normal</code> Befehl, um Befehle im normalen Modus von der Befehlszeile aus auszuführen
<code>:help cursor</code>	Vimscript-Befehl zum Bewegen des Cursors
<code>:help buffers</code>	Arbeiten mit Puffern Gleiches wie <code>:help windows</code>
<code>:help :buffer</code>	Der <code>:buffer</code>
<code>:help :vs</code>	Vertikale Aufteilung

Get: [help](https://riptutorial.com/de/vim/topic/8837/get--help--mit-dem-integrierten-handbuch-von-vim-) (mit dem integrierten Handbuch von Vim) online lesen:

<https://riptutorial.com/de/vim/topic/8837/get--help--mit-dem-integrierten-handbuch-von-vim->

Kapitel 19: Konvertieren von Textdateien von DOS nach UNIX mit vi

Bemerkungen

Das Zeichen `^M` steht für einen Wagenrücklauf in Vim (`<cm>` oder nur `<CR>`). Vim zeigt dieses Zeichen an, wenn mindestens eine Zeile in der Datei `LF` Zeilenenden verwendet. Mit anderen Worten, wenn Vim eine Datei als `fileformat=unix` aber in einigen Zeilen Wagenrücklauf (`CR`) enthalten, werden die Wagenrückläufe als `^M` angezeigt.

Eine Datei mit einer einzelnen Zeile mit `LF` Zeilenende und mehreren Zeilen mit `CRLF` Zeilenenden wird meistens durch falsches Bearbeiten einer auf einem MSDOS-System erstellten Datei erstellt. Zum Beispiel, indem Sie eine Datei unter einem MSDOS-Betriebssystem erstellen, auf ein UNIX-basiertes System kopieren und dann einen Hash-Bang-Sting (z. B. `#!/bin/sh`) mit Tools unter dem UNIX-basierten Betriebssystem voranstellen.

Examples

Konvertieren einer DOS-Textdatei in eine UNIX-Textdatei

Häufig haben Sie eine Datei, die in DOS oder Windows bearbeitet wurde und die Sie unter UNIX anzeigen. Dies kann wie folgt aussehen, wenn Sie die Datei mit vi anzeigen.

```
First line of file^M
Next Line^M
And another^M
```

Wenn Sie das `^M` entfernen möchten, können Sie jedes `^M` von Hand löschen. Alternativ können Sie in vi nach dem Drücken von `Esc` an der Eingabeaufforderung des Befehlsmodus Folgendes eingeben:

```
:1,$s/^M//g
```

Wobei `^M` zusammen mit `Ctrl` und `v` eingegeben wird und dann `Ctrl` und `m` zusammen.

Dies führt den Befehl von der ersten Zeile '1' bis zur letzten Zeile '\$' aus, der Befehl besteht darin, " das 'M' durch nichts zu ersetzen 'und' global " g '.

Verwenden des Dateiformats von Vim

Wenn Vim öffnet eine Datei mit `<CR><NL>` Zeilenenden (gemeinsam auf MS - DOS - basierte Betriebssysteme, die auch als `CRLF`) es eingestellt wird `fileformat` zu `dos` , können Sie , was Sie sich an:

```
:set fileformat?  
fileformat=dos
```

Oder nur

```
:set ff?  
fileformat=dos
```

Um es in `<NL>` -Zeilenenden umzuwandeln (üblich bei den meisten UNIX-basierten Betriebssystemen, auch als `LF`), können Sie die `fileformat` ändern.

```
:set ff=unix
```

Konvertieren von Textdateien von DOS nach UNIX mit vi online lesen:

<https://riptutorial.com/de/vim/topic/3827/konvertieren-von-textdateien-von-dos-nach-unix-mit-vi>

Kapitel 20: Makros

Examples

Makro aufnehmen

Eine Möglichkeit, ein Makro zu erstellen, besteht darin, es *aufzuzeichnen*.

Starten Sie die Aufnahme eines Makros und speichern Sie es in einem Register (in diesem Beispiel verwenden wir `a`, es kann jedoch ein beliebiges Register sein, zu dem Sie normalerweise Text hinzureißen könnten)

```
qa
```

Führen Sie dann die Befehle aus, die Sie im Makro aufnehmen möchten (hier umgeben wir den Inhalt einer Zeile mit `` -Tags):

```
I<li><ESC>A</li>
```

Wenn Sie mit den Befehlen fertig sind, die Sie im Makro aufnehmen möchten, stoppen Sie die Aufzeichnung:

```
q
```

Wenn Sie nun die in `a` gespeicherte aufgezeichnete Befehlsfolge ausführen möchten, verwenden Sie:

```
@a
```

und vim wiederholt die aufgezeichnete Sequenz.

Nächstes Mal, wenn Sie das zuletzt verwendete Makro wiederholen möchten, können Sie den Typ `@` doppelt `@`:

```
@@
```

Und als zusätzlichen Bonus ist es gut, sich daran zu erinnern, dass eine Zahl vor einem Befehl so oft wiederholt wird. So wiederholen Sie das Makro in dem Register gespeichert `a` 20 - mal mit:

```
20@a
```

Bearbeiten eines Vim-Makros

Manchmal machen Sie einen Fehler mit einem längeren Makro, aber bearbeiten Sie es lieber, als es vollständig neu aufzuzeichnen. Sie können dies mit dem folgenden Verfahren tun:

1. Setzen Sie das Makro in eine leere Zeile mit "`<register>p`".

Wenn Ihr Makro in Register `a` gespeichert ist, lautet der Befehl "`ap`".

2. Bearbeiten Sie das Makro nach Bedarf.

3. Ziehen Sie das Makro in das richtige Register, indem Sie den Cursor an den Anfang der Zeile bewegen und "`<register>y$`".

Sie können das Originalregister erneut verwenden oder ein anderes verwenden. Wenn Sie Register `b` möchten, lautet der Befehl "`by$`". Oder mit "`<register>d$`" (löscht die nicht verwendete Zeile)

Rekursive Makros

Vim-Makros können auch rekursiv sein. Dies ist nützlich, wenn Sie bis zum Ende der Datei auf jede Zeile (oder jedes andere Textobjekt) zugreifen müssen.

Beginnen Sie mit einem leeren Register, um ein rekursives Makro aufzuzeichnen. (Ein Register kann mit `q<register>q` geleert werden.)

Wählen Sie einen konsistenten Startpunkt für jede Linie, um zu beginnen und zu beenden.

Rufen Sie vor dem Beenden der Aufzeichnung das Makro selbst als letzten Befehl auf. (Deshalb muss das Register leer sein: Es wird also nichts unternommen, da das Makro noch nicht vorhanden ist).

Beispiel, gegeben den Text:

```
line 1
line 2
line 3
foo bar
more random text
.
.
.
line ???
```

Im Normalmodus kann man mit dem Cursor in der ersten Zeile und einem leeren Register `a` dieses Makro aufzeichnen:

```
qaI"<Esc>A"<Esc>j@aq
```

Bei einem einzigen Aufruf von `@a` würden sich nun alle Zeilen der Datei in doppelte Anführungszeichen setzen.

Was ist ein Makro?

Ein Makro ist eine Reihe von Tastenanschlägen, die von Vim ohne Verzögerung "wiedergegeben"

werden sollen. Makros können in Registern oder Variablen gespeichert, an Tasten gebunden oder in der Befehlszeile ausgeführt werden.

Hier ist ein einfaches Makro, das das dritte `word` in einer Zeile abdeckt:

```
0wwgUiw
```

Dieses Makro könnte in Register `q` *aufgezeichnet* werden:

```
qq      start recording into register q
0wwgUiw
q        stop recording
```

oder direkt im Register gespeichert `q` :

```
:let @q = '0wwgUiw'
```

abgespielt werden mit:

```
@q
```

Es könnte aber auch direkt in die Befehlszeile eingegeben werden:

```
:normal 0wwgUiw
```

für sofortige Wiedergabe über den Befehl `:normal .`

Oder in eine Variable schreiben:

```
:let myvar = '0wwgUiw'
```

abgespielt werden mit:

```
@=myvar
```

Oder als Mapping gespeichert:

```
nnoremap <key> 0wwgUiw
```

Wiedergabe durch Drücken der `<key>` .

Wenn Sie ein Makro zur späteren Wiederverwendung speichern möchten, können Sie den Einfügemodus eingeben:

```
<C-r>q
```

Dadurch wird das Makro in Register `q` `0wwgUiw` (in diesem Beispiel: `0wwgUiw`). Sie können diese

Ausgabe verwenden, um z. B. das Makro in Ihrer `vimrc` zu definieren:

```
let @q='0wwgUiw'
```

Auf diese Weise wird das Register `q` bei jedem Start von vim mit diesem Makro initialisiert.

Aktion aufnehmen und wiedergeben (Makros)

Mit dem Befehl `q` wir eine Menge langwieriger Arbeit in vim vereinfachen.

Beispiel 1. Erzeugen Sie eine Array-Sequenz (1 bis 20).

SCHRITT 1. Drücken Sie `i` , um den Einfügemodus zu aktivieren. Geben Sie `1`

```
1
```

SCHRITT 2. Nehmen Sie folgende Aktion auf: "Hängen Sie die letzte Nummer an die nächste Zeile an und erhöhen Sie die Nummer"

1. `esc` , um den Eingabemodus zu verlassen
2. `qa` um in den Aufzeichnungsmodus zu wechseln, und verwenden Sie dabei den Puffer `a`
3. `yy` und `p` um die aktuelle Zeile zu kopieren und als nächste Zeile einzufügen
4. `ctrl + A a` um die Nummer zu `ctrl`
5. `q` erneut `q` ein, um die Aufnahme zu beenden

```
1
2
```

SCHRITT 3. Aktion 18 Mal **wiederholen** .

Geben Sie `18@a` um Aktion 3 und Aktion 4 in Schritt 2 wiederzugeben.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Makros online lesen: <https://riptutorial.com/de/vim/topic/1447/makros>

Kapitel 21: Modi - Einfügen, normal, visuell, z

Examples

Die Grundlagen zu den Modi

`vim` ist ein modaler Editor. Dies bedeutet, dass sich der Benutzer zu jeder Zeit in einer `vim` Sitzung in einer der Betriebsarten befindet. Jeder von ihnen bietet verschiedene Befehle, Operationen, Tastenkombinationen ...

Normalmodus (oder Befehlsmodus)

- Der Modus `vim` beginnt in.
- Aus anderen Modi, normalerweise über `Esc` .
- **Hat die meisten Navigations- und Textbearbeitungsbefehle.**

Siehe `:help normal-mode` .

Einfügemodus

- Häufig wird aufgerufen durch: `a` , `i` , `A` , `I` , `c` , `s` .
- **Zum Einfügen von Text** .

Siehe `:help insert-mode` .

Visueller Modus

- Häufig wird aufgerufen durch: `v` (zeichenweise), `V` (linienweise), `<Cv>` (blockweise).
- Grundsätzlich zur **Textauswahl** ; Die meisten normalen Befehle stehen zur Verfügung, plus zusätzliche für den ausgewählten Text.

Siehe `:help visual-mode` .

Auswahlmodus

- Über den Einfügemodus mit `<Cg>` .
- Ähnlich wie der visuelle Modus, jedoch mit viel weniger verfügbaren Befehlen.
- Im Gegensatz zum Einfügemodus ist das Tippen sofort möglich.
- Kaum benutzt.

Siehe `:help select-mode` .

Ersetzen Sie den Modus

- Über den Normalmodus mit `R` .
- Erlaubt, vorhandenen Text zu überschreiben.

Siehe `:help replace-mode` .

Befehlszeilenmodus

Siehe `:help command-line-mode` .

Ex-Modus

Siehe `:help Ex-mode` .

Modi - Einfügen, normal, visuell, z online lesen: <https://riptutorial.com/de/vim/topic/2231/modi---einfugen--normal--visuell--z>

Kapitel 22: Nützliche Konfigurationen, die in `.vimrc` eingefügt werden können

Syntax

- `set maus = a`
- `Set Wrap`
- `nmap j gj`
- `nmap k gk`

Examples

Verschieben Sie die angezeigten Zeilen beim Wickeln nach oben oder unten

Normalerweise bewegen sich `J` und `K` die Zeilen der Dateien nach oben und unten. Wenn Sie die Umrandung jedoch aktiviert haben, können Sie stattdessen die **angezeigten** Zeilen auf und ab bewegen.

```
set wrap " if you haven't already set it
nmap j gj
nmap k gk
```

Aktivieren Sie die Mausinteraktion

```
set mouse=a
```

Dies ermöglicht die Mausinteraktion im `vim` Editor. Die Maus kann

- die aktuelle Cursorposition ändern
- Text auswählen

Konfigurieren Sie das Standardregister für die Verwendung als Systemzwischenablage

```
set clipboard=unnamed
```

Dadurch ist es möglich, zwischen Vim und der Systemzwischenablage zu kopieren / einzufügen, ohne ein spezielles Register anzugeben.

`yy` fügt die aktuelle Zeile in die Systemzwischenablage ein

`p` fügt den Inhalt der Systemzwischenablage in Vim ein

Dies funktioniert nur, wenn Ihre Vim-Installation die Unterstützung der Zwischenablage unterstützt.

Führen Sie den folgenden Befehl im Terminal aus, um zu prüfen, ob die Zwischenablageoption verfügbar ist: `vim --version | grep clipboard`

Nützliche Konfigurationen, die in `.vimrc` eingefügt werden können online lesen:

<https://riptutorial.com/de/vim/topic/6560/nutzliche-konfigurationen--die-in--vimrc-eingefugt-werden-können>

Kapitel 23: Ostereier

Examples

Hilfe!

Für den notleidenden Benutzer liefert vim Weisheit.

```
:help!
```

Wenn du dich schlecht fühlst

Problem: Vim-Benutzer sind nicht immer glücklich.

Lösung: Machen Sie sie glücklich.

7.4

```
:smile
```

Hinweis: Erfordert eine Patch-Version \geq [7.4.1005](#)

Die Antwort

Vim weiß die Antwort:

```
:help 42
```

Vim öffnet das Dokument `usr_42.txt` aus dem Handbuch und zeigt den folgenden Text an:

Was bedeutet Leben, Universum und alles? **42**

Douglas Adams, die einzige Person, die wusste, worum es in dieser Frage wirklich ging, ist leider leider tot. Nun fragst du dich vielleicht, was der Tod bedeutet ...

Auf der Suche nach dem Heiligen Gral

Überprüfen Sie dies heraus:

```
:help holy-grail
```

Ceci n'est pas une Pfeife

Wenn Sie den Hilfebereich von | suchen oder `bar :: :h bar` siehst du:

```
|                                     bar
                                     To screen column [count] in the current line.
```

exclusive motion. Ceci n'est pas une pipe.

Dies ist ein Hinweis auf das Gemälde *La trahison des images* von René Magritte.



Wenn ein Benutzer sich langweilt

Suchen Sie nach `:h UserGettingBored`

```
UserGettingBored      *UserGettingBored*
                       When the user presses the same key 42 times.
                       Just kidding! :-)
```

Löffel

Anstatt nach der `fork` suchen, können Sie nach der `spoon` suchen:

```
:h spoon

fork spoon

For executing external commands fork()/exec() is used when possible, otherwise
system() is used, which is a bit slower. The output of ":version" includes ...
```

Ritter, die Ni sagen!

Überprüfen Sie dies heraus:

```
:Ni!
```

Monty Python und der Heilige Gral

nunmap

```
:help map-modes
```

```
:nunmap can also be used outside of a monastery.
```


Ostereier online lesen: <https://riptutorial.com/de/vim/topic/4656/ostereier>

Kapitel 24: Plugins

Examples

Flüchtige Vim

Fugitive Vim ist ein Plugin von Tim Pope, das Zugriff auf git-Befehle bietet, die Sie ausführen können, ohne vim zu verlassen.

Einige gebräuchliche Befehle umfassen:

:Gedit - Bearbeiten Sie eine Datei im Index und schreiben Sie sie, um die Änderungen in Szene zu setzen

:Gstatus - entspricht dem `git status`

:Gblame - vertikale Aufteilung der Ausgabe von `git blame`

:Gmove - für `git mv`

:Gremove - für `git rm`

:Git - führe einen beliebigen Befehl aus

Sie fügt der `statusline` außerdem Elemente hinzu, `statusline` den aktuellen Zweig.

Bitte [lesen](#) Sie deren [GitHub](#) für weitere Details und Installationsanweisungen.

NERD-Baum

NERD TREE ist ein Plugin von scrooloose, mit dem Sie das Dateisystem mit vim erkunden können. Sie können Dateien und Verzeichnisse über ein Baumsystem öffnen, das Sie mit der Tastatur oder der Maus bearbeiten können.

Fügen Sie dies Ihrem `.vimrc` hinzu, um NERDTree beim Start von vim automatisch zu starten:

```
autocmd vimenter * NERDTree
```

Um NERDTree automatisch zu schließen, wenn dies das einzige verbleibende Fenster ist, fügen Sie dies zu Ihrer `.vimrc` hinzu:

```
autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTree") && b:NERDTree.isTabTree()) | q  
| endif
```

Es wird empfohlen, dem NERDTreeToggle-Befehl eine Tastenkombination zuzuordnen. Fügen Sie dies Ihrer `.vimrc` hinzu (in diesem Beispiel wird Strg + N verwendet).

```
map <C-n> :NERDTreeToggle<CR>
```

Vollständige Details und Installationsanweisungen können auf ihrem [Github](#) angezeigt werden .

Plugins online lesen: <https://riptutorial.com/de/vim/topic/9976/plugins>

Kapitel 25: Puffer

Examples

Puffer verwalten

Sie können Puffer verwenden, um mit mehreren Dateien zu arbeiten. Wenn Sie eine Datei mit öffnen

```
:e path/to/file
```

es wird in einem neuen Puffer geöffnet (der Befehl bedeutet, die Datei zu bearbeiten). Neuer Puffer, der eine temporäre Kopie der Datei enthält.

Sie können zum vorherigen Puffer mit `:bp[rev]` und zum nächsten Puffer mit `:bn[ext]` .

Mit `b{n}` können Sie zu einem bestimmten Puffer `b{n}` , um zum n-ten Puffer zu gelangen. `b2` geht zum zweiten Puffer.

Verwenden Sie `:ls` oder `:buffers` um alle Puffer aufzulisten

Versteckte Puffer

Wenn Sie sich mit ungespeicherten Änderungen von einem Puffer entfernen, wird dieser Fehler angezeigt:

```
E37: No write since last change (add ! to override)
```

Sie können dies deaktivieren, indem Sie der VIMRC-Datei den Befehl `set hidden` hinzufügen. Mit dieser Option bleiben Ihre Änderungen im Puffer erhalten, werden jedoch nicht auf der Festplatte gespeichert.

Puffer wechseln mit einem Teil des Dateinamens

Zur einfachen Auswahl eines Puffers nach Dateiname können Sie Folgendes verwenden:

```
:b [part_of_filename]<Tab><Tab><Tab>...<Enter>
```

Im ersten `Tab` wird das Wort zu einem vollständigen Dateinamen erweitert, und nachfolgende `Tabulator`tasten durchlaufen die Liste der möglichen Übereinstimmungen.

Wenn mehrere Übereinstimmungen verfügbar sind, können Sie *vor* der Worterweiterung eine Liste mit Übereinstimmungen anzeigen, indem Sie diese Option festlegen:

```
:set wildmode=longest:full:list,full
```

Auf diese Weise können Sie Ihr Wort verfeinern, wenn die Liste der Übereinstimmungen zu lang ist, aber für die Erweiterung eine zusätzliche `Tabulatortaste` erforderlich ist. Fügen Sie die Einstellung Ihrem `$MYVIMRC` wenn Sie ihn `$MYVIMRC` möchten.

Einige Leute starten diesen Prozess gerne mit einer Keymap, die zuerst die Puffer auflistet:

```
:nnoremap <Leader>b :set nomore <Bar> :ls <Bar> :set more <CR>:b<Space>
```

Das macht es einfach, einen Puffer anhand seiner Nummer auszuwählen:

```
:b [buffer_num]
```

Wechseln Sie schnell zum vorherigen Puffer oder zu einem beliebigen Puffer nach Nummer

`<C-^>` wechselt zur vorherigen bearbeiteten Datei. Auf den meisten Tastaturen ist `<C-^>` STRG-6.

`3<C-^>` wechselt zu Puffer Nummer 3. Dies ist sehr schnell, aber nur, wenn Sie die Puffer-Nummer kennen.

Sie können die [Puffernummern](#) aus `:ls` oder aus einem Plugin wie [MiniBufExplorer](#) sehen .

Puffer online lesen: <https://riptutorial.com/de/vim/topic/2317/puffer>

Kapitel 26: Rechtschreibprüfung

Examples

Rechtschreibprüfung

So aktivieren Sie den Vim-Rechtschreibprüfungslauf `:set spell`. Um es auszuschalten, run `:set nospell`. Wenn die Rechtschreibprüfung immer aktiviert sein soll, fügen Sie dem vimrc einen `set spell` hinzu. Sie können die Rechtschreibung nur für bestimmte Dateitypen mithilfe eines automatischen Befehls aktivieren.

Sobald die Rechtschreibprüfung aktiviert ist, werden falsch geschriebene Wörter hervorgehoben. Tippen Sie `]s`, um zum nächsten falsch geschriebenen Wort zu gelangen, und `[s`, um zum vorherigen Wort zu gelangen. Um eine Liste der korrigierten Schreibweisen anzuzeigen, setzen Sie den Cursor auf ein falsch geschriebenes Wort und geben Sie `z=`. Sie können die Nummer des Wortes `<number>` mit dem Sie das falsch geschriebene Wort ersetzen möchten, und die `<number>`, um es zu ersetzen. Sie können auch die Eingabetaste drücken, um das Wort unverändert zu lassen.

Wenn sich der Cursor auf einem falsch geschriebenen Wort befindet, können Sie auch `<number>z=` eingeben, um zur `<number>`-ten Korrektur zu wechseln, ohne die Liste anzuzeigen. Normalerweise verwenden Sie `1z=` wenn Sie der Meinung sind, dass die erste Vermutung von vim wahrscheinlich das richtige Wort ist.

Setze Wortliste

Um die Wortliste `spelllang`, die vim für die Rechtschreibprüfung verwendet, setzen Sie die Option für die `spelllang`. Zum Beispiel

```
:set spelllang=en      # set to English, usually this is the default
:set spelllang=en_us   # set to U.S. English
:set spelllang=en_uk   # set to U.K. English spellings
:set spelllang=es      # set to Spanish
```

Wenn Sie die `spelllang` und die Rechtschreibprüfung in einem Befehl `spelllang` möchten, können Sie `spelllang` tun:

```
:setlocal spell spelllang=en
```

Rechtschreibprüfung online lesen: <https://riptutorial.com/de/vim/topic/3653/rechtschreibprüfung>

Kapitel 27: Reguläre Ausdrücke

Bemerkungen

ausführen `:h pattern` , um eine Vielzahl von Informationen zum regulären Ausdruck anzuzeigen

Examples

Wort

Vim verfügt über spezielle Operatoren, um Wortanfang, Wort, Ende usw. abzugleichen. `\<` für den Anfang eines Wortes und `\>` für das Ende eines Wortes.

`/\<foo\>` im folgenden Text nach `/\<foo\>` wird nur das letzte foo zurückgegeben.

Fußball ist kein dummes `foo`

Reguläre Ausdrücke online lesen: <https://riptutorial.com/de/vim/topic/6533/regulare-ausdrucke>

Kapitel 28: Reguläre Ausdrücke im Ex-Modus

Examples

Bearbeiten Sie einen regulären Ausdruck im Ex-Modus

Angenommen, Sie suchen nach einem `Title Case` in einer großen Textdatei und möchten einen falschen regulären Ausdruck bearbeiten:

1. Gehen Sie zuerst in den `Ex` Modus, indem Sie `q:` eingeben `q:`
2. Sie sehen jetzt alle Befehle, die Sie im `commandline` eingegeben haben. Drücken Sie `j` , um den regulären Ausdruck zu öffnen, den Sie bearbeiten möchten (`/\v[AZ]\w+\s[AZ]\w+`)
3. Wenn Sie fertig sind, drücken Sie `ESC` , um in den Normalmodus zu wechseln
4. Drücken `Enter` dann die `Enter` , um das Suchmuster auszuführen

Hier ist ein Screenshot, der die Suche nach `Title Case`


```
1 Lorem Ipsum is simply dummy text  
0 Lorem Ipsum has been the industry  
>\galley of type and scrambled  
1 It has survived not only five  
>\unchanged.  
2 It was popularised in the 1960s  
>\recently with desktop publishing
```

<https://riptutorial.com/de/vim/topic/6472/regulare-ausdrucke-im-ex-modus>

Kapitel 29: Schlüsselzuordnungen in Vim

Einführung

Durch die Aktualisierung von Vim-Tastenzuordnungen können Sie zwei Arten von Problemen lösen: Erneutes Zuweisen von Tastenbefehlen zu Buchstaben, die einprägsamer oder zugänglicher sind, und Erstellen von Tastenbefehlen für Funktionen, die keine haben. Hier erfahren Sie mehr über die verschiedenen Arten, wie Sie Tastenbefehle zuordnen und den Kontext, auf den sie sich beziehen (z. B. *Vim-Modi*).

Examples

Grundlegende Zuordnung

Übersichtskarte

Eine Schlüsselsequenz kann in einer anderen Tastenfolge erneut abgebildet werden, eine der mit den `map`

Beispielsweise verlässt die folgende typische `map` den *Einfügemodus*, wenn Sie `j k` in schneller Reihenfolge drücken:

```
:inoremap jk <Esc>
```

Kartenoperator

Es gibt mehrere Varianten von `:map` für verschiedene Modi.

Befehle	Modi
<code>:map :noremap :unmap :noremap :unmap</code>	Normaler, visueller und vom Bediener anstehender Modus
<code>:map! , :noremap! , :unmap!</code>	Einfügen und Befehlszeilenmodus
<code>:nmap :nnoremap :nunmap :nnoremap :nunmap</code>	Normaler Modus
<code>:imap :inoremap :iunmap :inoremap :iunmap</code>	Einfüge- und Ersetzungsmodus
<code>:vmap :vnoremap :vunmap :vnoremap :vunmap</code>	Sicht- und Auswahlmodus
<code>:xmap :xnoremap :xunmap :xnoremap :xunmap</code>	Visueller Modus

Befehle	Modi
<code>:smap :snoremap :sunmap :snoremap :sunmap</code>	Auswahlmodus
<code>:cmap :cnoremap :cunmap :cnoremap :cunmap</code>	Befehlszeilenmodus
<code>:omap :onoremap :ounmap :onoremap :ounmap</code>	Betreiber ausstehender Modus

Normalerweise sollten Sie die [:noremap Varianten verwenden](#) ; Dies macht das Mapping gegen Remapping und Rekursion immun.

Befehl map

- Sie können alle Zuordnungen anzeigen mit `:map` (oder einer der oben genannten Variationen).
- Um die aktuelle Zuordnung für eine bestimmte Tastenfolge anzuzeigen, verwenden Sie `:map <key>` wobei `<key>` eine Tastenfolge ist
- Sondertasten wie `Esc` werden mit der speziellen Notation `<>` wie `<Esc>` . Die vollständige Liste der Schlüsselcodes finden Sie unter <http://vimdoc.sourceforge.net/html/doc/intro.html#keycodes>
- `:nmapclear` - Alle Karten im `:nmapclear`
- `:nunmap` - Hebt eine normale Moduszuordnung auf
- Sie können durch eine Änderung der die maximale Zeit zwischen den Tasten einer Sequenz konfigurieren `timeout` und `tttimeout` Variablen

Beispiele

- `imap jk <Esc> : jk` im Einfügemodus `jk` eingeben, `imap jk <Esc>` Sie zum normalen Modus zurück
- `nnoremap tt :tabnew<CR> tt` im normalen Modus eingeben, wird eine neue Registerkarte geöffnet
- `nnoremap <Cj> <Cw>j : <Cj>` im normalen Modus `<Cj>` , springen Sie zum Fenster darunter und nach links
- `vmap <Cc> \cc : vmap <Cc> \cc` im visuellen Modus `<Cc>` eingeben, wird `\cc` (NERDCommenter-Befehl zum Kommentieren der Zeile). Da dies auf einem Plugin-Mapping beruht, können Sie `:vnoremap` hier nicht verwenden!

Lesen Sie [hier weiter](#)

Map Leader-Tastenkombination

Der Leiterschlüssel kann zum Erstellen eines Mappings mit einer Schlüsselbindung verwendet werden, die vom Endbenutzer überschrieben werden kann.

Der Leader ist standardmäßig der `\` key. Um es zu überschreiben, muss der Endbenutzer `:let`

`g:mapleader='somekey(s) ' ausführen :let g:mapleader='somekey(s) ' bevor Sie das Mapping definieren.`

In einem typischen Szenario wird der Mapleader in `.vimrc`, und Plugins verwenden `<Leader>` im Keybinding-Teil ihrer Zuordnungen, um sie anpassbar zu machen.

Im Plugin würden wir Zuordnungen definieren mit:

```
:nnoremap <Leader>a somecomplexaction
```

Dies würde die `Somecomplexaction`- Aktion der Tastenkombination `\ + a` zuordnen.

Die `eine` Aktion ohne Führer nicht ändert.

Es ist auch möglich, `<Plug>Mappings` zu verwenden, um mehr Spielraum für die Anpassung der Plugins-Keybindings zu lassen.

Abbildung des grundlegenden Mappings (praktische Abkürzungen).

In den meisten Texteditoren lautet die Standardverknüpfung zum Speichern des aktuellen Dokuments `Strg + s` (oder `Cmd + s` unter macOS).

Vim verfügt nicht standardmäßig über diese Funktion, dies kann jedoch zur Vereinfachung der Zuordnung zugeordnet werden. Die folgenden Zeilen werden in der `.vimrc` Datei `.vimrc`.

```
nnoremap <c-s> :w<CR>
inoremap <c-s> <c-o>:w<CR>
```

Der Befehl `nnoremap` ordnet `Ctrl + s` zu `:w` (aktuellen Inhalt in Datei schreiben) zu, während der `inoremap` Befehl den Befehl `Strg + s` zu `:w` ordnet und zum Einfügemodus zurückkehrt (`<co>` für einen Befehl und in den normalen Modus kehrt anschließend in den Einfügemodus zurück, ohne die Cursorposition zu ändern, was andere Lösungen wie `<esc>:w<cr>a` nicht gewährleisten können).

Ähnlich,

```
" This is commented, as Ctrl+Z is used in terminal emulators to suspend the ongoing
program/process.
" nnoremap <c-z> :u<CR>

" Thus, Ctrl+Z can be used in Insert mode
inoremap <c-z> <c-o>:u<CR>

" Enable Ctrl+C for copying selected text in Visual mode
vnoremap <c-c> <c-o>:y<CR>
```

PS: Es muss jedoch beachtet werden, dass `Ctrl + s` während der Verwendung von `ssh` (oder `PuTTY`) möglicherweise nicht wie erwartet funktioniert. Die Lösung dieses Problems ist nicht in den Anwendungsbereich dieses Dokuments, aber gefunden werden kann [hier](#).

Schlüsselzuordnungen in Vim online lesen:

<https://riptutorial.com/de/vim/topic/3535/schlüsselzuordnungen-in-vim>

Kapitel 30: Scrollen

Examples

Nach unten scrollen

Befehl	Beschreibung
Strg + E	Eine Zeile nach unten scrollen.
Strg + D	Einen halben Bildschirm nach unten <code>scroll</code> (konfigurierbar mit der <code>scroll</code> Option)
Strg + F	Einen ganzen Bildschirm nach unten scrollen.
z +	Zeichnen Sie die erste Linie unter dem Fenster am oberen Rand des Fensters.

Aufwärts scrollen

Befehl	Beschreibung
Strg + Y	Eine Zeile nach oben scrollen
Strg + U	Blättern Sie einen halben Bildschirm (konfigurierbar , um die Verwendung von <code>scroll</code> - Option).
Strg + B	Einen ganzen Bildschirm nach oben scrollen.
z ^	Zeichnen Sie die erste Linie über dem Fenster am unteren Rand des Fensters.

Bildlauf relativ zur Cursorposition

Befehl	Beschreibung
z	Zeichne die aktuelle Zeile am oberen Rand des Fensters neu und setze den Cursor auf das erste nicht leere Zeichen in der Zeile.
zt	Wie <code>z</code> , aber lassen Sie den Cursor in derselben Spalte.
z.	Zeichne die aktuelle Zeile in der Mitte des Fensters neu und setze den Cursor auf das erste nicht leere Zeichen in der Zeile.
zz	Wie <code>z.</code> Lassen Sie den Cursor jedoch in derselben Spalte.
z-	Zeichne die aktuelle Zeile am unteren Rand des Fensters neu und setze den Cursor auf das erste nicht leere Zeichen in der Zeile.

Befehl	Beschreibung
Zum Beispiel	Wie <code>z-</code> aber lassen Sie den Cursor in der gleichen Spalte.

Scrollen online lesen: <https://riptutorial.com/de/vim/topic/3000/scrollen>

Kapitel 31: So kompilieren Sie Vim

Examples

Kompilieren auf Ubuntu

So erstellen Sie vim aus dem Quellcode auf Ubuntu:

1. Laden Sie eine Kopie des Quellcodes herunter, indem Sie sie aus dem [offiziellen Vim-Repository auf GitHub](#) herunterladen.
2. Holen Sie sich die Abhängigkeiten, indem Sie `$ sudo apt-get build-dep vim-gnome` oder ähnliches `$ sudo apt-get build-dep vim-gnome`.
3. Wechseln Sie in das Verzeichnis des Vim-Quellcodes: `cd vim/src`
4. Führen Sie `$./configure`. Sie können den Build anpassen (und die Sprachintegrationen von Perl, Python usw. aktivieren), indem Sie Konfigurationsoptionen übergeben. Eine Übersicht finden Sie unter `src/INSTALL`.
5. Führen Sie `$ make`.
6. Beenden Sie die Installation, indem Sie `$ sudo make install`. Da Ihr selbst kompilierter Vim nicht vom Paketmanager verwaltet wird, wird er in `/usr/local/bin/vim` anstelle von `/usr/bin/vim`. Um es auszuführen, müssen Sie entweder den vollständigen Pfad angeben oder sicherstellen, dass `/usr/local/bin` vor `/usr/bin` in Ihrem `PATH` (normalerweise).
7. (Optional) Entfernen Sie die verteilungsgestützte Version von Vim, falls Sie sie bereits installiert haben: `$ sudo apt-get remove vim vim-runtime gvim`.

Um die Installation zu überprüfen, können Sie `$ which vim` ausführen, das bei erfolgreicher Installation etwas wie `/usr/local/bin/vim`.

So kompilieren Sie Vim online lesen: <https://riptutorial.com/de/vim/topic/3737/so-kompilieren-sie-vim>

Kapitel 32: Solarized Vim

Einführung

Die meiste Zeit am Terminal zu verbringen, kann für die Augen eine große Sache sein. Die richtige Farbwahl kann Ihren Augen in vielerlei Hinsicht zugute kommen. Vor kurzem bin ich auf [Solarized ColorScheme for Vim](#) . Das Hinzufügen dieses kleinen Plugins kann großen Einfluss auf den Textauftritt auf dem Terminal haben. Vielen Dank an Ethan Schoonover für die Entwicklung dieses Pakets. Die `howtos` werden [hier](#) ziemlich gut erklärt. Genießen!

Examples

`.vimrc`

Solarized bietet zwei Optionen - den hellen und den dunklen Modus.

Lichtmodus :

```
syntax enable
set background=light
colorscheme solarized
```

Dunkler Modus :

```
syntax enable
set background=dark
colorscheme solarized
```

Solarized Vim online lesen: <https://riptutorial.com/de/vim/topic/9500/solarized-vim>

Kapitel 33: Speichern

Examples

Speichern eines Puffers in einem nicht vorhandenen Verzeichnis

```
:!mkdir -p %:h
```

um die fehlenden Verzeichnisse zu erstellen, dann

```
:w
```

Speichern online lesen: <https://riptutorial.com/de/vim/topic/6440/speichern>

Kapitel 34: Suche im aktuellen Puffer

Examples

Suche nach einem beliebigen Muster

Die Standardsuchbefehle von Vim sind `/` für die Vorwärtssuche und `?` für die Rückwärtssuche.

So starten Sie eine Suche im normalen Modus:

1. drücken Sie `/` ,
2. Schreibe dein Muster,
3. Drücken Sie `<CR>` , um die Suche durchzuführen.

Beispiele:

```
/foobar<CR>      search forward for foobar
?foo\bar<CR>     search backward for foo/bar
```

`n` und `N` können Sie zum nächsten und vorherigen Ereignis springen:

- Durch Drücken von `n` nach einer Vorwärtssuche wird der Cursor beim nächsten Vorkommen *vorwärts angezeigt* .
- Durch Drücken von `N` nach einer Vorwärtssuche wird der Cursor beim nächsten Vorkommen *rückwärts* positioniert.
- Durch Drücken von `n` nach einer Rückwärtssuche wird der Cursor beim nächsten Vorkommen *rückwärts* positioniert.
- Durch Drücken von `N` nach einer Rückwärtssuche wird der Cursor beim nächsten Vorkommen *vorwärts angezeigt* .

Suche nach dem Wort unter dem Cursor

Bewegen Sie den Cursor im normalen Modus zu einem beliebigen Wort und drücken Sie `*` , um das nächste Vorkommen des Wortes unter dem Cursor vorwärts zu suchen, oder drücken Sie `#` , um rückwärts zu suchen.

`*` oder `#` Suche nach dem exakten Wort unter dem Cursor: Die Suche nach `big` würde nur `big` und nicht `bigger` .

Unter der Haube verwendet Vim eine einfache Suche mit *Wortgrenzen* Atom:

- `/\<big\>` für `*` ,
- `?\<big\>` für `#` .

`g*` oder `g#` suchen nicht nach dem genauen Wort unter dem Cursor: Suchen nach `big` würde `bigger` .

Unter der Haube verwendet Vim eine einfache Suche ohne Atome mit *Wortgrenzen* :

- `/\<big\>` für `*` ,
- `?\<big\>` für `#` .

Befehl in Zeilen ausführen, die Text enthalten

Der Befehl `:global` bereits ein eigenes Thema: [Der globale Befehl](#)

Suche im aktuellen Puffer online lesen: <https://riptutorial.com/de/vim/topic/3269/suche-im-aktuellen-puffer>

Kapitel 35: Suchen und Ersetzen

Examples

Ersetzen Sie den Befehl

Dieser Befehl:

```
:s/foo/bar/g
```

ersetzt jedes Vorkommen von `foo` durch einen `bar` in der aktuellen Zeile.

```
fool around with a foodie
```

wird

```
barl around with a bardie
```

Wenn Sie das letzte `/g` weglassen, wird nur das erste Vorkommnis in der Zeile ersetzt. Zum Beispiel,

```
:s/foo/bar
```

Auf der vorherigen Zeile würde es werden

```
barl around with a foodie
```

Dieser Befehl:

```
:5,10s/foo/bar/g
```

führt die gleiche Substitution in den Zeilen 5 bis 10 aus.

Dieser Befehl

```
:5,$s/foo/bar/g
```

führt die gleiche Ersetzung von Zeile 5 bis zum Ende der Datei aus.

Dieser Befehl:

```
:%s/foo/bar/g
```

führt dieselbe Ersetzung für den gesamten Puffer durch.

Wenn Sie sich im visuellen Modus befinden und den Doppelpunkt drücken, erscheint das Symbol '<', '>'. Sie können das dann tun

```
: '<', '>' s/foo/bar/g
```

und die Ersetzung innerhalb Ihrer visuellen Modusauswahl erfolgen.

Dieser Befehl:

```
:%s/foo/bar/gc
```

ist äquivalent zu dem obigen Befehl, fragt jedoch bei jedem Vorkommnis dank der Markierung `/c` (für "Bestätigung") eine Bestätigung ab.

Siehe `:help :s` und `:help :s_flags`.

Siehe auch [diesen Abschnitt zu Befehlszeilenbereichen](#).

Ersetzen Sie mit oder ohne reguläre Ausdrücke

Dieser Ersetzungsbefehl kann [reguläre Ausdrücke verwenden](#) und entspricht jeder Instanz von `foo` gefolgt von einem (einem) Zeichen seit dem Zeitraum. In regulären Ausdrücken entspricht jedes Zeichen einem der folgenden Zeichen. Daher werden alle Instanzen von `foo` gefolgt von einem beliebigen Zeichen in der aktuellen Zeile gefunden.

```
:s/foo./bar/g
```

```
1 fooing fooes fool foobar foosup
```

wird werden

```
1 barng bars bar barar barup
```

Wenn Sie das Literal anpassen möchten. Zeitraum können Sie es im Suchfeld mit einem Backslash `\` entziehen.

```
:s/foo\./bar/g
```

```
1 fooing fooes foo.l foo.bar foosup
```

wird werden

```
1 fooing fooes barl barbar foosup
```

Oder deaktivieren Sie alle Musteranpassungen, indem Sie dem Befehl `s` mit `no` folgen.

```
:sno/foo./bar/g
```

```
1 fooing fooes foo.l foo.bar foosup
```

wird einen Fehler auslösen

```
E486: Pattern not found
```

Suchen und Ersetzen online lesen: <https://riptutorial.com/de/vim/topic/3533/suchen-und-ersetzen>

Kapitel 36: Text bearbeiten

Bemerkungen

Um Dinge wie `11:59AM`, `3rd` und `XVIII` zu erhöhen und zu dekrementieren, verwenden Sie das Plugin [Vim-Speeddating](#)

Examples

Textfall konvertieren

Im normalen Modus:

- `~` invertiert den Fall des Zeichens unter dem Cursor,
- `gu{motion}` verkleinert den von `{motion}` abgedeckten Text,
- `gU{motion}` den von `{motion}` abgedeckten Text ein

Beispiel (`^` markiert die Cursorposition):

```

Lorem ipsum dolor sit amet.
      ^
Lorem ipSum dolor sit amet.      ~
Lorem IPSUM DOLOR sit amet.      gU2w
Lorem IPsum DOLOR sit amet.      gue
```

Im visuellen Modus:

- `~` invertiert den Fall des ausgewählten Textes,
- `u` verkleinert den ausgewählten Text,
- `U` den ausgewählten Text dar

Beispiel (`^^^` markiert die visuelle Auswahl):

```

Lorem ipsum dolor sit amet.
      ^^^^^^^^^^^^^^^
Lorem ipSUM DOLOR SIT amet.      ~
Lorem ipSUM DOLOR SIT amet.      U
Lorem ipsum dolor sit amet.      u
```

Inkrementieren und Dekrementieren von Zahlen und Buchstaben

Im Normalmodus können wir die nächste Zahl in der Zeile am oder nach dem Cursor mit `<Ca>` erhöhen und mit `<Cx>` dekrementieren. In den folgenden Beispielen ist die Cursorposition mit `^`.

Zahlen inkrementieren und dekrementieren

```
for i in range(11):  
    ^
```

<Cx> verringert die Anzahl:

```
for i in range(10):  
    ^
```

10<Ca> erhöht es um 10 :

```
for i in range(20):  
    ^
```

Inkrementieren und Dekrementieren von alphabetischen Zeichen

Um das Inkrementieren und Dekrementieren auch für Buchstaben zu ermöglichen, verwenden Sie entweder den Befehl `ex :set nrformats+=alpha` oder fügen Sie `set nrformats+=alpha` zu Ihrer `.vimrc`.

Inkrement-Beispiel:

```
AAD  
  ^
```

<Ca> erhöht es zu B :

```
ABD  
  ^
```

Dekrement Beispiel:

```
ABD  
  ^
```

<Cx> dekrementiert D bis C :

```
ABC  
  ^
```

Zahlen inkrementieren und dekrementieren, wenn das alphabetische Inkrementieren / Dekrementieren aktiviert ist

Wenn Sie das Inkrementieren / Dekrement für die Arbeit mit alphabetischen Zeichen aktivieren, müssen Sie darauf achten, diese nicht zu ändern, wenn Sie wirklich nur Zahlen ändern möchten. Sie können das alphabetische Inkrementieren / Dekrementieren entweder mit dem Befehl `ex :set nrformats-=alpha` oder Sie können es einfach wissen und sicher sein, vor dem Inkrementieren oder Dekrementieren zur Zahl **zu wechseln**. Hier ist das " `for i in range(11):` " - Beispiel von oben, um die Arbeit zu wiederholen, während das alphabetische Inkrement / Dekrement eingestellt ist:

Angenommen, Sie möchten 11 bis 10 verringern und das alphabetische Inkrement / Dekrement ist aktiv.

```
for i in range(11):  
    ^
```

Da alphabetisches Inkrement / Dekrement aktiv ist, zu vermeiden, dass die Zeichen unter dem Cursor zu modifizieren, zuerst vorwärts zum ersten Schritt 1 unter Verwendung des *normalen Betriebsart* Bewegungsbefehl `f1` (die Kleinbuchstabe `f` gefolgt von der Nummer 1, nicht mit einer Funktionstaste verwechseln):

```
for i in range(11):  
    ^
```

Da sich der Cursor nun auf der Zahl befindet, können Sie sie mit `<Cx>` dekrementieren. Nach dem Dekrementieren wird der Cursor auf die letzte Ziffer der Ziffer gesetzt:

```
for i in range(10):  
    ^
```

Formatierungscode

Im normalen Modus:

`gg` nach oben gehen

= dann `G`

Verwendung von "Verben" und "Substantiven" zur Textbearbeitung

Um über die Befehle nachzudenken, die ausgeführt werden sollen, um einen Text auf bestimmte Weise zu bearbeiten, können Sie ganze Sätze darstellen.

Ein Befehl ist eine Aktion, die für ein Objekt ausgeführt wird. Deshalb hat es ein Verb:

```
:normal i    " insert  
:normal a    " append  
:normal c    " overwrite  
:normal y    " yank (copy)  
:normal d    " delete
```

Einige dieser Wörter funktionieren mit einem Objekt wie `d`, `c`, `y`. Solche Objekte können **Wort**, **Zeile**, **Satz**, **Absatz**, **Tag sein**. Man kann diese in Kombination verwenden:

```
:normal dw    " deletes the text from the position of the cursor to the end of the next word  
:normal cw    " deletes the text from the cursor to the end of the next word and  
               " enters insert mode
```

Man könnte auch einen **Modifikator verwenden**, um genau anzugeben, wo die Aktion ausgeführt

werden soll:

```
:normal diw      " delete inside word. I.e. delete the word in which is the cursor.
:normal ciw      " removes the word, the cursor points at and enters insert mode
:normal ci"      " removes everything between the opening and closing quotes and
                  " enters insert mode
:normal cap      " change the current paragraph
:normal ct8      " remove everything until the next number 8 and enter insert mode
:normal cf8      " like above but remove also the number
:normal c/goal   " remove everything until the word 'goal' and enter insert mode
:normal ci{      " change everything inside the curly braces
```

Mehr Ressourcen:

[Lernen Sie, vim - Verben, Substantive und Modifikatoren zu sprechen!](#)

[Vim lernen im Jahr 2014: Vim als Sprache](#)

[VimSpeak-Bearbeitung mithilfe der Sprachgrammatik](#)

[Text bearbeiten online lesen: https://riptutorial.com/de/vim/topic/1707/text-bearbeiten](https://riptutorial.com/de/vim/topic/1707/text-bearbeiten)

Kapitel 37: Text einfügen

Examples

Verlassen des Einfügemodus

Befehl	Beschreibung
<Esc>	Verlässt den Einfügemodus, löst automatische Befehle und Abkürzungen aus
<C-[>	Genau auch von <Esc>
<Cc>	Verlässt den Einfügemodus, löst keine automatischen Befehle aus

Einige Leute verwenden gerne ein relativ ungewöhnliches Zeichenpaar wie `jk` als Abkürzung für `<Esc>` oder `<C-[>`, die auf manchen Tastaturen nur schwer zu erreichen sind:

```
inoremap jk <Esc>l
```

Verschiedene Möglichkeiten, um in den Einfügemodus zu gelangen

Befehl	Beschreibung
<code>a</code>	Text an die aktuelle Cursorposition anhängen
<code>A</code>	Text am Ende der aktuellen Zeile anhängen
<code>i</code>	Fügen Sie Text vor der aktuellen Cursorposition ein
<code>I</code>	Fügen Sie Text vor dem ersten nicht leeren Zeichen der aktuellen Zeile ein
<code>gI</code>	Fügen Sie Text in die erste Spalte der Cursorzeile ein
<code>gi</code>	Fügen Sie den Text an der Stelle ein, an der er zuletzt im Einfügemodus verlassen wurde
<code>O</code>	Eröffne eine neue Zeile über der aktuellen Zeile und füge dort Text hinzu (CAPITAL <code>O</code>)
<code>o</code>	Öffne eine neue Zeile unterhalb der aktuellen Zeile und füge dort Text hinzu (Kleinbuchstabe <code>o</code>)
<code>s</code> oder <code>cI</code>	Zeichen unter dem Cursor löschen und in den Einfügemodus wechseln
<code>S</code> oder <code>cc</code>	Löschen Sie die gesamte Zeile und wechseln Sie in den Einfügemodus

Befehl	Beschreibung
<code>c</code>	Löschen Sie von der Cursorposition bis zum Ende der Zeile und starten Sie den Einfügemodus
<code>c{motion}</code>	<code>{motion}</code> löschen und Einfügemodus starten (siehe Basic Motion)

Einfügemodus-Verknüpfungen

Befehl	Beschreibung
<code><Cw></code>	Wort vor dem Cursor löschen
<code><Ct></code>	Aktuelle Zeile mit einer <code>shiftwidth</code>
<code><Cd></code>	Entrollte aktuelle Zeile um eine <code>shiftwidth</code>
<code><Cf></code>	Linie neu einblenden (Cursor auf Position für automatischen Einzug bewegen)
<code><Ca></code>	Fügen Sie zuvor eingefügten Text ein
<code><Ce></code>	Geben Sie das Zeichen unten ein
<code><Ch></code>	Ein Zeichen rückwärts löschen
<code><Cy></code>	Fügen Sie das Zeichen oben ein
<code><Cr>{register}</code>	Füge den Inhalt von <code>{register}</code>
<code><Co>{normal mode command}</code>	Führen Sie <code>{normal mode command}</code> ohne den Einfügemodus zu verlassen
<code><Cn></code>	Nächste Autovervollständigungsoption für das aktuelle Wort
<code><Cp></code>	Vorherige Autovervollständigungsoption für das aktuelle Wort
<code><Cv></code>	Fügen Sie ein Zeichen mit seinem ASCII-Wert in Dezimalzahl ein
<code><Cv>x</code>	Fügen Sie ein Zeichen mit dem ASCII-Wert in Hexadezimal ein
<code><Cv>u</code>	Fügen Sie ein Zeichen nach dem Unicode-Wert in Hexadezimal ein
<code><Ck></code>	Geben Sie einen Digraphen ein

Ausführen normaler Befehle aus dem Einfügemodus

Drücken Sie im Einfügemodus `<Co>` , um den Einfügemodus vorübergehend zu verlassen und einen einmaligen normalen Befehl auszuführen.

Beispiel

`<Co>2w` springt zum zweiten Wort nach links und kehrt zum Einfügemodus zurück.

Hinweis: Wiederholen mit . werden nur die Aktionen vom Zurückkehren in den Einfügemodus wiederholen

Dies ermöglicht einige nützliche Mappings, z

```
inoremap <C-f> <Right>
inoremap <C-b> <Left>
inoremap <C-a> <C-o>^
inoremap <C-e> <C-o>$
```

Strg + a setzt jetzt den Cursor an den Anfang der Zeile und Strg + e - an das Ende der Zeile. Diese Zuordnungen werden standardmäßig in `readline` können daher für Personen hilfreich sein, die Konsistenz wünschen.

Fügen Sie den Text in mehreren Zeilen gleichzeitig ein

Drücken Sie `Strg + v`, um den visuellen Blockierungsmodus aufzurufen.

Verwenden Sie `↑ / ↓ / j / k`, um mehrere Zeilen auszuwählen.

Drücken Sie die Umschalttaste + i und geben Sie ein, was Sie möchten.

Nachdem Sie die `Esc`-Taste gedrückt haben, wird der Text in alle ausgewählten Zeilen eingefügt.

Denken Sie daran, dass `Ctrl + c` nicht zu 100% mit `Esc` identisch ist und in dieser Situation nicht funktioniert!

Es gibt leichte Variationen von `Shift + i`, die Sie im visuellen Blockmodus drücken können:

Schlüssel	Beschreibung
<code>c</code> oder <code>s</code>	Löschen Sie den ausgewählten Block und wechseln Sie in den Einfügemodus
<code>C</code>	Löschen Sie ausgewählte Zeilen (vom Cursor bis zum Ende) und wechseln Sie in den Einfügemodus
<code>R</code>	Löschen Sie ausgewählte Zeilen und wechseln Sie in den Einfügemodus
<code>EIN</code>	An ausgewählte Zeilen anhängen, wobei sich die Spalte am Ende der ersten Zeile befindet

Beachten Sie auch das Drücken von `.`. Nach einem visuellen Blockiervorgang wird der Vorgang wiederholt, an dem sich der Cursor befindet!

Fügen Sie Text mit dem Befehl "Einfügen" des Terminals ein

Wenn Sie den Befehl Einfügen aus Ihrem Terminalemulatorprogramm verwenden, interpretiert Vim den Zeichenstrom so, als ob er eingegeben worden wäre. Dies führt zu allen möglichen unerwünschten Wirkungen, insbesondere zu ungünstigen Hinweisen.

Um dies zu beheben, im Befehlsmodus:

```
:set paste
```

Gehen Sie dann in den Einfügemodus, beispielsweise mit `i`. Beachten Sie, dass der Modus jetzt `- INSERT (paste) --`. Fügen Sie nun den Befehl mit dem Terminalemulator oder mit der Maus ein. Wenn Sie fertig sind, gehen Sie in den Befehlsmodus, mit `Esc` und führen Sie Folgendes aus:

```
:set nopaste
```

Es gibt einen einfacheren Weg, wenn Sie nur einmal einfügen möchten. *Fügen Sie dies in Ihre `.vimrc` ein* (oder verwenden Sie das Plugin [unimpaired.vim](#)):

```
function! s:setup_paste() abort
  set paste
  augroup unimpaired_paste
    autocmd!
    autocmd InsertLeave *
      \ set nopaste |
      \ autocmd! unimpaired_paste
  augroup end
endfunction

nnoremap <silent> yo :call <SID>setup_paste()<CR>o
nnoremap <silent> yO :call <SID>setup_paste()<CR>O
```

Jetzt können Sie einfach `yo` drücken, um den Code unter dem Cursor einzufügen, und dann `<Esc>`, um zum normalen / nopaste-Modus zurückzukehren.

Einfügen aus einem Register im Einfügemodus

Im Einfügemodus können Sie mit `<Cr>` aus einem Register einfügen, das durch den nächsten Tastendruck festgelegt wird. `<Cr>` zum Beispiel Pasten aus dem unbenannten (`"`) - Register.

Siehe `:help registers`.

Erweiterte Einfügebefehle und Verknüpfungen

Hier finden Sie eine Kurzanleitung für erweiterte Befehle zum Einfügen, Formatieren und Filtern.

Befehl / Verknüpfung	Ergebnis
<code>g +? + m</code>	Rot13-Kodierung bei Bewegung <i>m</i> durchführen
<code>n + ctrl + a</code>	+ <i>n</i> zur Nummer unter dem Cursor

Befehl / Verknüpfung	Ergebnis
<i>n</i> + Strg + x	- <i>n</i> bis Nummer unter dem Cursor
g + q + <i>m</i>	Bewegungslinien <i>m</i> auf feste Breite formatieren
: r ce w	Mittellinien im Bereich <i>r</i> bis Breite <i>w</i>
: r le i	Zeilen links im Bereich <i>r</i> mit Einzug ausrichten <i>i</i>
: r ri w	Richten Sie die Zeilen im Bereich <i>r</i> an der Breite <i>w</i> aus
! <i>mc</i>	Bewegungslinien <i>m</i> durch Befehl <i>c</i> filtern
<i>n</i> !! <i>c</i>	<i>N</i> Zeilen durch Befehl <i>c</i> filtern
: r ! <i>c</i>	Filterbereich <i>r</i> Zeilen durch Befehl <i>c</i>

Deaktivieren Sie den automatischen Einzug, um Code einzufügen

Wenn der Text über einen Terminal - Emulator einfügen, die auto-indent - Funktion *kann* die Vertiefung des eingefügten Text zerstören.

Zum Beispiel:

```
function () {
  echo 'foo'
  echo 'bar'
  echo 'baz'
}
```

wird eingefügt als:

```
function () {
  echo 'foo'
    echo 'bar'
      echo 'baz'
}
```

Verwenden Sie in diesen Fällen die Option `paste / nopaste`, um die Funktion zum automatischen `nopaste` zu aktivieren / deaktivieren:

```
:set paste
:set nopaste
```

Hinzu kommt ein einfacherer Ansatz für das Problem: Fügen Sie die folgende Zeile in Ihre `.vimrc` ein:

```
set pastetoggle=<F3>
```

Und wenn Sie einfügen möchten, wie es ist, aus der Zwischenablage. Drücken Sie einfach `F3` im `insert` und fügen Sie ein. `F3` erneut `F3` , um den `paste` .

Text einfügen online lesen: <https://riptutorial.com/de/vim/topic/953/text-einfugen>

Kapitel 38: Tipps und Tricks zur Steigerung der Produktivität

Syntax

- : Relative Nummer setzen
- :Nummer setzen
- : set nonumber /: set nonu
- : pwd

Bemerkungen

Diese automatische Nachladen geschieht nur, wenn Sie bearbeiten `vimrc` in einer Version Vollversion von vim , die unterstützt `autocmd` .

Examples

Schnelle "Wegwerf" -Makros

Fügen Sie dies zu Ihrem vimrc hinzu:

```
nnoremap Q @q
```

Um mit der Aufnahme des Makros "Wegwerfen" zu beginnen, verwenden Sie `qq` . Um die Aufnahme zu beenden, drücken Sie `q` (im Normalmodus für beide).

Um das aufgezeichnete Makro auszuführen, verwenden Sie `Q`

Dies ist nützlich für Makros, die Sie viele Male hintereinander wiederholen müssen, danach aber nicht mehr verwendet werden.

Verwenden der Funktion zum Vervollständigen des Pfads in Vim

Dies ist sehr üblich. Sie speichern einen Pfad zu einer Datei oder einem Ordner, Sie öffnen Vim und versuchen, das, was Sie gerade gespeichert haben, zu schreiben. Sie sind jedoch nicht zu 100% sicher, dass sie korrekt ist.

Wenn Sie die Funktion zum Vervollständigen des Pfads wünschen und eine Datei `/home/ubuntu/my_folder/my_file` und eine andere Datei bearbeiten, die auf den Pfad der vorherigen Datei verweist:

Rufen Sie den Einfügemodus auf: `Einfügen` oder wie gewünscht. Als nächstes schreiben Sie `/h` . Wenn sich der Cursor unter `h` , drücken Sie `Strg x` und dann `Strg f` , damit der Editor ihn nach `/home/` vervollständigt, da das Muster `/h` eindeutig ist

Angenommen, Sie haben zwei Ordner in `/home/ubuntu/` namens `my_folder_1` `my_folder_2`

und Sie möchten den Pfad `/home/ubuntu/my_folder_2`

wie gewöhnlich:

Aktivieren Sie den Einfügemodus

Schreiben Sie `/h` und drücken Sie `Strg x` und dann `Strg f`. Jetzt haben Sie `/home/` Next und fügen Sie nach `/home/` hinzu und drücken Sie `Strg x` und dann `Strg f`. Jetzt haben Sie `/home/ubuntu/` weil dieser Pfad eindeutig ist. Nun schreibe `my` after `/home/ubuntu/` und drücke `Strg x` und dann `Strg f`. Der Editor vervollständigt Ihr Wort bis `my_folder_` und der Verzeichnisbaum wird `my_folder_` Verwenden Sie die Pfeiltasten, um den gewünschten auszuwählen.

Aktivieren Sie relative Zeilennummern

Um einige Textzeilen zu löschen, wenn Sie die genaue Anzahl der zu `10dd` Zeilen nicht kennen, versuchen Sie es mit `10dd`, `5dd` und `3dd` bis Sie alle Zeilen entfernen.

Relative Zeilennummern lösen dieses Problem. Angenommen, wir haben eine Datei, die Folgendes enthält:

```
sometimes, you see a block of
text. You want to remove
it but you
cannot directly get the
exact number of
lines to delete
so you try
10d , 5d
3d until
you
remove all the block.
```

Wechseln Sie in den NORMAL-Modus: `Esc`

Nun führe aus `:set relativenumber`. Wenn Sie fertig sind, wird die Datei wie folgt angezeigt:

```
3 sometimes, you see a block of
2 text. You want to remove
1 it but you
0 cannot directly get the
1 exact number of
2 lines to delete
3 so you try
4 10d , 5d
5 3d until
6 you
7 remove all the block.
```

Dabei ist `0` die Zeilennummer für die aktuelle Zeile und sie zeigt auch die reale Zeilennummer vor der relativen Nummer. Sie müssen also nicht die Anzahl der Zeilen aus der aktuellen Zeile schätzen, um sie zu schneiden oder zu löschen, oder sie zu zählen einzeln.

Sie können jetzt Ihren üblichen Befehl wie `6dd` und sind sich der Anzahl der Zeilen sicher.

Sie können auch die Kurzform desselben Befehls verwenden `:set rnu`, um relative Zahlen zu `:set nornu`, und `:set nornu`, um dieselben auszuschalten.

Wenn Sie auch `:set number` oder `:set number` bereits eingestellt haben, erhalten Sie die Zeilennummer der Zeile, in der sich der Cursor befindet.

```
3 sometimes, you see a block of
2 text. You want to remove
1 it but you
4 cannot directly get the
1 exact number of
2 lines to delete
3 so you try
4 10d , 5d
5 3d until
6 you
7 remove all the block.
```

Zeilennummern anzeigen

Um die Zeilennummern in der Standardansicht anzuzeigen, geben Sie ein

```
:set number
```

Zeilennummern ausblenden

```
:set nonumber
```

Es gibt auch eine Abkürzung für oben. `nu` ist gleich `number`.

```
:set nonu
```

Zum Ausblenden von Zeilennummern können wir auch verwenden

```
:set nu!
```

Zuordnungen zum Beenden des Einfügemodus

Für viele Vim-Benutzer ist der `Esc` zu schwer zu erreichen, und am Ende finden Sie ein anderes Mapping, das von der Home-Reihe aus leicht zu erreichen ist. Beachten Sie, dass `Strg - [` auf einer englischen Tastatur mit `Esc` übereinstimmen kann und viel einfacher zu erreichen ist.

j k

```
inoremap jk <ESC>
```

Dieser ist wirklich leicht auszulösen; Zerschlage einfach deine ersten beiden Finger gleichzeitig in der Heimatreihe. Es ist auch schwer aus Versehen auszulösen, da "jk" in keinem englischen Wort vorkommt und wenn Sie sich im normalen Modus befinden, tut es nichts. Wenn Sie nicht zu viel "Blackjack" `inoremap kj <ESC>` , sollten Sie auch die `inoremap kj <ESC>` hinzufügen, damit Sie sich keine Gedanken über das Timing der beiden Schlüssel machen müssen.

Feststelltaste

Linux

Unter Linux können Sie mit `xmodmap` dasselbe wie `Esc` mit `Caps Lock` tun. Fügen Sie dies in eine Datei ein:

```
!! No clear Lock
clear lock
!! make caps lock an escape key
keycode 0x42 = Escape
```

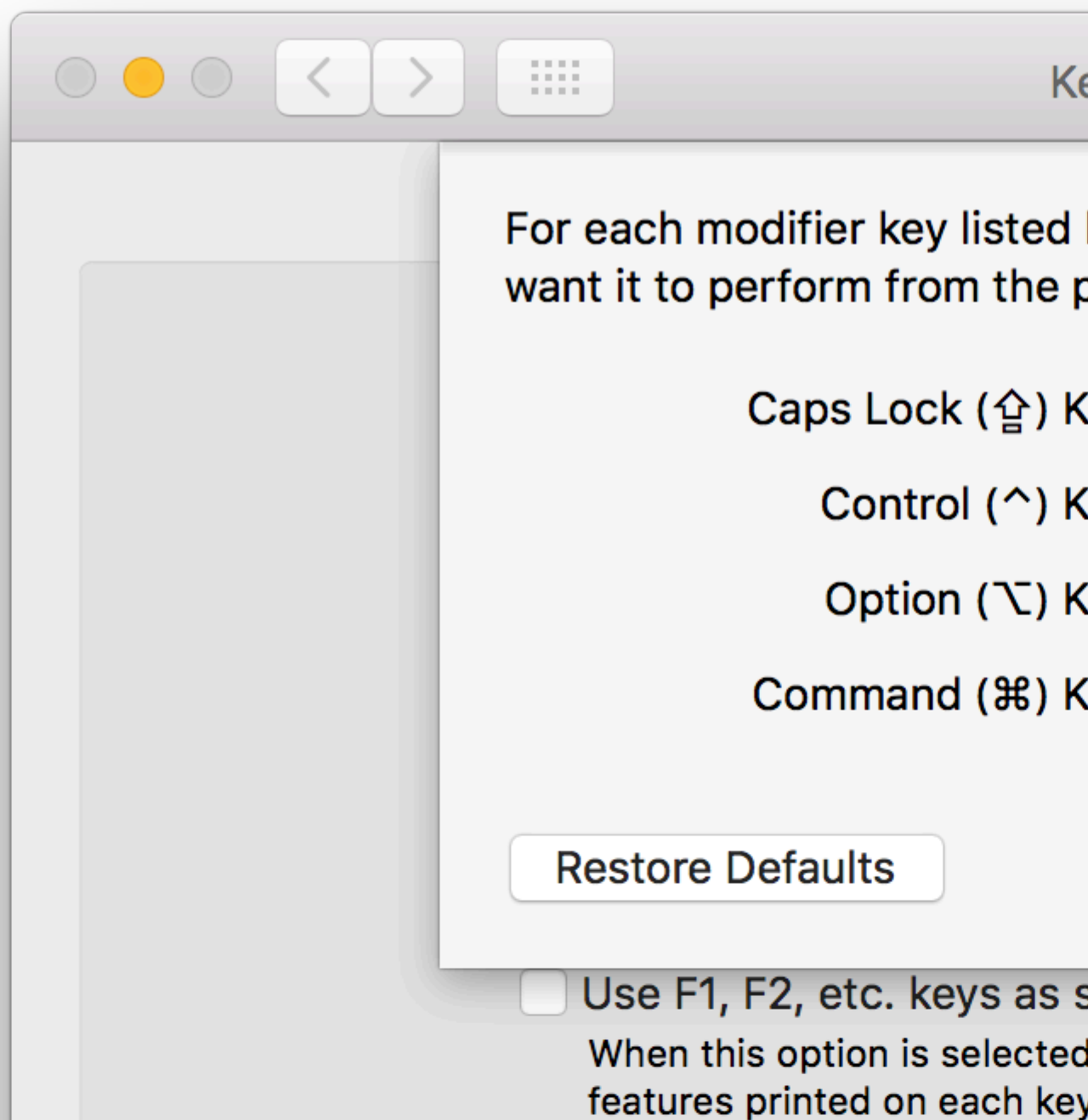
Führen Sie dann die `xmodmap file` . Damit wird `Caps Lock` wieder mit `Esc` belegt .

Windows

Unter Windows können Sie [SharpKey](#) oder [AutoHotkey](#) verwenden .

Mac OS

Wenn Sie über macOS 10.12.1 oder höher verfügen, können Sie `Caps Lock` mit `Escape` in den Systemeinstellungen neu zuordnen. Wählen Sie `Tastatur` aus, wechseln Sie zur Registerkarte `Tastatur` und klicken Sie auf `Modifikatortasten`.



Block umgibt den Cursor verwenden Sie den folgenden Befehl (<ESC> - Escape - Taste, <CR> - Enter - Taste):

```
vi{<ESC>/\%Vfoo<CR>
```

Jetzt können Sie durch Drücken von `n` und `p` zwischen den Übereinstimmungen innerhalb des Blocks springen. Wenn Sie die `hlsearch` Option aktiviert haben, werden alle Übereinstimmungen hervorgehoben. `\%V` ist ein spezieller Suchmusterteil, der vim anweist, nur im visuell ausgewählten Bereich zu suchen. Sie können auch ein Mapping wie folgt erstellen:

```
:vnoremap g/ <ESC>/\%V
```

Danach wird der obige Befehl folgendermaßen gekürzt:

```
vi{g/foo<CR>
```

Ein weiterer nützlicher Trick besteht darin, alle Zeilen mit dem Muster zu drucken:

```
vi{  
: '<, '>g/foo/#
```

Der Bereich `'<, '>` wird automatisch eingefügt.

Siehe `:help range` und `:help :g`.

Gefundene Zeile kopieren, verschieben oder löschen

Viele Benutzer befinden sich in einer Situation, in der sie eine Zeile nur schnell kopieren, verschieben oder löschen und dorthin zurückkehren möchten, wo sie sich befanden.

Wenn Sie eine Zeile verschieben möchten, die das `lot` unterhalb der aktuellen Zeile enthält, geben Sie normalerweise Folgendes ein:

```
/lot<Esc>dd<C-o>p
```

Um die Produktivität zu steigern, können Sie diese Abkürzung in folgenden Fällen verwenden:

```
" It's recommended to turn on incremental search when doing so  
set incsearch  
  
" copy the found line  
cnoremap $t <CR>:t'<CR>  
" move the found line  
cnoremap $m <CR>:m'<CR>  
" delete the found line  
cnoremap $d <CR>:d<CR>``
```

Also eine Suche wie diese:


```
/lot$m
```

verschiebt die Zeile mit dem `lot` unter die Zeile, auf der sich der Cursor befand, als Sie die Suche gestartet haben.

Schreiben Sie eine Datei, wenn Sie ``sudo`` vergessen haben, bevor Sie vim starten

Dieser Befehl speichert die geöffnete Datei mit Sudo-Rechten

```
:w !sudo tee % >/dev/null
```

Sie können auch `w!!` eine Datei als root ausschreiben

```
:cnoremap w!! w !sudo tee % >/dev/null
```

Laden Sie vimrc beim Speichern automatisch neu

Um `vimrc` beim Speichern automatisch neu zu laden, fügen Sie dem `vimrc` Folgendes `vimrc` :

```
if has('autocmd') " ignore this section if your vim does not support autocommands
  augroup reload_vimrc
    autocmd!
    autocmd! BufWritePost $MYVIMRC,$MYGVIMRC nested source %
  augroup END
endif
```

und dann zum letzten Mal:

```
:so $MYVIMRC
```

`vimrc` Sie das nächste Mal Ihren `vimrc` , wird er automatisch neu geladen.

`nested` ist nützlich, wenn Sie vim-airline verwenden. Beim Laden der Airline werden einige automatische Befehle ausgelöst. Da Sie jedoch gerade einen automatischen Befehl ausführen, werden diese übersprungen. `nested` ermöglicht das Auslösen verschachtelter Autobefehle und das ordnungsgemäße Laden der Fluggesellschaft.

Kommandozeilenabschluss

`wildmenu` im `wildmenu` , `wildmenu` Vorschläge zur Fertigstellung der Befehlszeile `wildmenu` werden sollen.

Führen Sie folgendes aus

```
set wildmenu
set wildmode=list:longest,full
```

Wenn Sie jetzt sagen `:color Tab` ,

Du wirst kriegen

```
256-jungle Benokai BlackSea C64 CandyPaper Chasing_Logic ChocolateLiquor  
:color 0x7A69_dark
```

Tipps und Tricks zur Steigerung der Produktivität online lesen:

<https://riptutorial.com/de/vim/topic/3382/tipps-und-tricks-zur-steigerung-der-produktivitat>

Kapitel 39: Unterschiede zwischen Neovim und Vim

Examples

Konfigurationsdateien

In Vim ist Ihre Konfigurationsdatei `~/.vimrc`, weitere Konfigurationsdateien in `~/.vim`.

In Neovim befinden sich Konfigurationsdateien in `~/.config/nvim`. Es gibt auch keine `~/.nvimrc` Datei. Verwenden Sie stattdessen `~/.config/nvim/init.vim`.

Sie können Ihre Vim-Konfiguration mit diesem Unix-Befehl direkt in Neovim importieren:

```
ln -s ~/.vimrc ~/.config/nvim/init.vim
```

Unterschiede zwischen Neovim und Vim online lesen:

<https://riptutorial.com/de/vim/topic/7848/unterschiede-zwischen-neovim-und-vim>

Kapitel 40: Verbessertes Rückgängigmachen und Wiederherstellen mit einem Undodir

Examples

Konfigurieren Sie Ihre Vimrc für die Verwendung eines Undodir

Seit der vim-Version 7.3 wird die Funktion 'persistent_undo' unterstützt, sodass Änderungen auch rückgängig gemacht werden können, nachdem vim geschlossen oder der Computer neu gestartet wurde.

Sie können es konfigurieren, indem Sie Folgendes zu Ihrem vimrc hinzufügen, aber erstellen Sie zunächst ein Verzeichnis, in dem Ihre Undofiles gespeichert werden sollen. Sie können die Datei überall erstellen, ich empfehle jedoch die Verwendung des Verzeichnisses ".vim".

```
if has('persistent_undo')           "check if your vim version supports
    set undodir=$HOME/.vim/undo      "directory where the undo files will be stored
    set undofile                     "turn on the feature
endif
```

Nachdem Sie dies zu Ihrem vimrc hinzugefügt und das vimrc erneut beschafft haben, können Sie die Funktion mithilfe der [grundlegenden Befehle zum Rückgängigmachen / Wiederherstellen verwenden](#)

Verbessertes Rückgängigmachen und Wiederherstellen mit einem Undodir online lesen:

<https://riptutorial.com/de/vim/topic/7875/verbessertes-ruckgangigmachen-und-wiederherstellen-mit-einem-undodir>

Kapitel 41: Vertiefung

Examples

Einrücken einer gesamten Datei mithilfe der eingebauten Einrückungs-Engine

Geben Sie im Befehlsmodus (Esc) Folgendes ein :gg=G , um die eingebaute Indention-Engine von Vim zu verwenden.

Befehlsteil	Beschreibung
gg	Beginn der Datei
=	Einzug (wenn equalprg leer ist)
G	Ende der Datei

Sie können equalprg in Ihrer .vimrc festlegen, um ein equalprg Werkzeug zur automatischen Formatierung zu verwenden.

Um beispielsweise das clang-format für C / C ++ zu verwenden, .vimrc die folgende Zeile in Ihre .vimrc Datei ein:

```
autocmd FileType c,cpp setlocal equalprg=clang-format
```

Ersetzen Sie für andere Dateitypen c, cpp durch den Dateityp, den Sie formatieren möchten, und das clang-format mit Ihrem bevorzugten Formatierungstool für diesen Dateityp.

Zum Beispiel:

```
" Use xmllint for indenting XML files. Commented out.
"autocmd FileType xml setlocal equalprg=xmllint\ --format\ --recover\ -\ 2>/dev/null
" Tidy gives more formatting options than xmllint
autocmd FileType xml setlocal equalprg=tidy\ --indent-spaces\ 4\ --indent-attributes\ yes\ --
sort-attributes\ alpha\ --drop-empty-paras\ no\ --vertical-space\ yes\ --wrap\ 80\ -i\ -xml\
2>/dev/null
```

Zeilen einrücken oder ausrücken

Um die aktuelle Zeile im **normalen Modus** einzurücken, drücken Sie die Taste größer als > oder die Taste kleiner als < zweimal. Um das Gleiche in mehreren Zeilen zu tun, fügen Sie einfach eine Zahl vor 6>>

Befehl	Beschreibung
>>	Aktuelle Zeile einrücken

Befehl	Beschreibung
<<	aktuelle Linie ausfahren
6>>	Einrücken der nächsten 6 Zeilen

Sie können auch mit [Bewegungen einrücken](#) . Hier einige nützliche Beispiele.

Befehl	Beschreibung
>gg	Einzug von der aktuellen Zeile zur ersten Zeile in der Datei
>G	Einzug von der aktuellen Zeile bis zur letzten Zeile in der Datei
>{	Spiegelstrich des vorherigen Absatzes
>}	nächsten Absatz einrücken

Im [visuellen Modus](#) durch einmaliges Drücken der Taste größer oder kleiner als. Beachten Sie, dass dies den [visuellen Modus](#) verlässt. Dann kannst du verwenden . Um die Bearbeitung zu wiederholen, müssen Sie `u` und es rückgängig machen.

Vertiefung online lesen: <https://riptutorial.com/de/vim/topic/6324/vertiefung>

Kapitel 42: Verwenden von Python für Vim-Scripting

Syntax

- : [Bereich] py [Thon] {Anweisung}

Examples

Überprüfen Sie die Python-Version in Vim

Vim hat einen eigenen integrierten Python-Interpreter. Daher könnte eine andere Version des Standardinterpreters für das Betriebssystem verwendet werden.

Geben Sie den folgenden Befehl ein, um zu überprüfen, mit welcher Version von Python Vim kompiliert wurde:

```
:python import sys; print(sys.version)
```

Dadurch wird das `sys` Modul importiert und seine `version` mit der Version des aktuell verwendeten Python-Interpreters gedruckt.

Führen Sie die Vim-Befehle im Normalmodus über die Python-Anweisung aus

Um vim-Befehle in Python verwenden zu können, sollte das `vim` Modul importiert werden.

```
:python import vim
```

Nachdem dieses Modul importiert wurde, hat der Benutzer Zugriff auf die `command` :

```
:python vim.command("normal iTText to insert")
```

Dieser Befehl würde `i` im normalen Modus ausführen, dann `Text to insert` und zum normalen Modus zurückkehren.

Mehrzeiligen Python-Code ausführen

Jeder Python-Anweisung in Vim sollte der Befehl `:python` vorangestellt werden, um Vim anzuweisen, dass der nächste Befehl nicht Vimscript, sondern Python ist.

Um zu vermeiden, dass dieser Befehl in jeder Zeile eingegeben wird, kann bei der Ausführung von mehrzeiligem Python-Code Vim angewiesen werden, den Code zwischen zwei Markerausdrücken als Python zu interpretieren.

Um dies zu erreichen, verwenden Sie:

```
:python << {marker_name}
a = "Hello World"
print(a)
{marker_name}
```

Dabei ist {marker_name} das Wort, das Sie zum {marker_name} des Endes des Python-Blocks verwenden möchten.

Z.B:

```
:python << endpython
surname = "Doe"
forename = "Jane"
print("Hello, %s %s" % (forename, surname))
endpython
```

würde drucken:

```
Hello, Jane Doe
```

Verwenden von Python für Vim-Scripting online lesen:

<https://riptutorial.com/de/vim/topic/5604/verwenden-von-python-fur-vim-scripting>

Kapitel 43: vglobal: Führt Befehle in Zeilen aus, die nicht global übereinstimmen

Einführung

: vglobal oder: v ist das Gegenteil von: global oder: g, das auf Zeilen wirkt, die nicht mit dem angegebenen Muster übereinstimmen (invers).

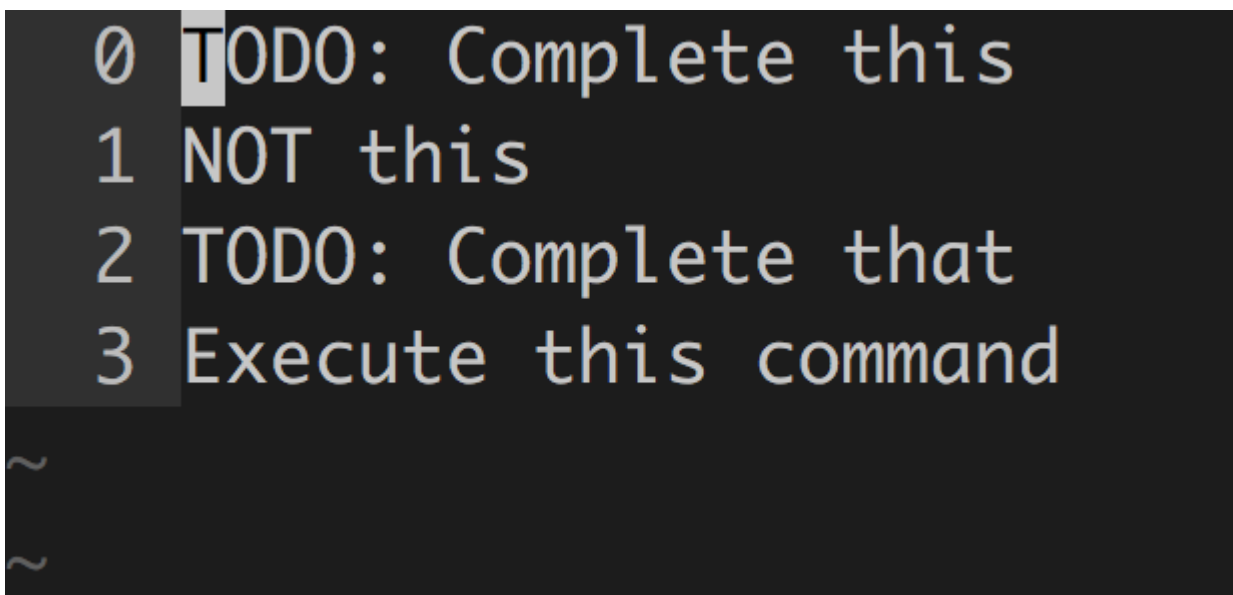
Examples

: v / pattern / d

Beispiel:

```
> cat example.txt
TODO: complete this
NOT this
NOT that
TODO: Complete that
```

Öffnen Sie die `example.txt` mit `vim` und geben Sie im `Ex` Modus `:v/TODO/d` . Dadurch werden alle Zeilen gelöscht, die das `TODO` Muster nicht enthalten.



```
0 TODO: Complete this
1 NOT this
2 TODO: Complete that
3 Execute this command
~
~
```

```
1 TODO: Complete this
0 TODO: Complete that
```

vglobal: Führt Befehle in Zeilen aus, die nicht global übereinstimmen online lesen:
<https://riptutorial.com/de/vim/topic/9867/vglobal--fuhr-befehle-in-zeilen-aus--die-nicht-global-ubereinstimmen>

Kapitel 44: Vim erweitern

Bemerkungen

Ein Plugin ist ein Skript oder eine Gruppe von Skripts, das das Standardverhalten von Vim ändert, indem entweder nicht vorhandene Features hinzugefügt oder vorhandene Features erweitert werden.

Zu den häufig hinzugefügten "nicht vorhandenen Funktionen" gehören:

- kommentieren,
- Eindrückungserkennung,
- Autovervollständigung,
- Fuzzy-Matching,
- Unterstützung für eine bestimmte Sprache
- usw.

Zu den häufig vorhandenen "vorhandenen Funktionen" gehören:

- omni-vervollständigung,
- Textobjekte & Bewegungen,
- zerren & setzen,
- Statuszeile,
- Suchen & Ersetzen,
- Puffer- / Fenster- / Registerseitenumschaltung,
- falten,
- usw.

Examples

Wie funktionieren Plugins?

Ein Plugin kann sich als einzelne Datei präsentieren, die 30 Zeilen Vimscript enthält, oder als 20 MB Vimscript / Python / Ruby / was auch immer in viele Dateien aufgeteilt wird, die sich auf ein Dutzend Verzeichnisse beziehen, die von einer Reihe externer Tools abhängen.

Ersteres ist offensichtlich einfach zu installieren und zu verwalten, aber letzteres könnte eine ziemliche Herausforderung darstellen.

Das Prinzip

Die Option `'runtimepath'` teilt Vim mit, wo nach Laufzeitskripten `'runtimepath'` werden soll. Der Standardwert macht Vim sucht Skripte in die folgenden Verzeichnisse *in der Reihenfolge*:

- auf UNIX-ähnlichen Systemen

- `$HOME/.vim/`
- `$VIM/vimfiles/`
- `$VIMRUNTIME/`
- `$VIM/vimfiles/after/`
- `$HOME/.vim/after/`

- unter Windows

- `$HOME/vimfiles/`
- `$VIM/vimfiles/`
- `$VIMRUNTIME/`
- `$VIM/vimfiles/after/`
- `$HOME/vimfiles/after/`

Installieren Sie Plugins nur in den fett gedruckten Verzeichnissen. Die anderen verursachen aus keinem guten Grund Instabilität. Die Installation eines Plugins führt dazu, dass sich jede seiner Komponenten im rechten Verzeichnis unter `$HOME/.vim/` oder `$HOME/vimfiles/`.

Die manuelle Methode

Plugin für einzelne Dateien

Legen Sie die Datei unter `$HOME/.vim/plugin` oder `$HOME/vimfiles/plugin`

Dies würde das Plugin beim Start von Vim beziehen. Nun kann der Benutzer alles, was darin definiert ist, verwenden. Wenn das Plugin jedoch aktiviert werden muss, muss der Benutzer den Befehl entweder selbst ausführen, wenn er es verwenden möchte, oder den Befehl zu `.vimrc`

Bündeln

Ein Bundle ist eine Verzeichnisstruktur, die das Plugin verwendet. Es besteht aus allen Dateien des Plugins unter den entsprechenden Unterverzeichnissen.

Um ein solches Plugin zu installieren, müssen die Unterverzeichnisse mit ihren Gegenstücken in `$HOME/.vim/plugin`. Dieser Ansatz führt jedoch zum Mischen der Dateien verschiedener Plugins in denselben Verzeichnissen und kann möglicherweise zu Namensraumproblemen führen.

Ein anderer Ansatz besteht darin, das gesamte Verzeichnis nach `$HOME/.vim/bundle` zu kopieren.

Bei diesem Ansatz sollte sich mindestens eine `.vim` Datei im `.vim $HOME/.vim/bundle/autoload`. Diese Dateien werden beim Start von vim bereitgestellt.

Hinweis: Abhängig vom Betriebssystem des Benutzers kann das Präfix aller Pfade `$HOME/vimfiles`. Weitere Informationen finden Sie unter Funktionsweise von [Plugins](#)

VAM

<https://github.com/MarcWeber/vim-addon-manager>

Vundle

Vundle ist ein Plugin-Manager für Vim.

Vundle installieren

(Die vollständigen Installationsdetails finden Sie im [Vundle-Schnellstart](#).)

1. Installieren Sie [Git](#) und klonen Sie Vundle in `~/.vim/bundle/Vundle.vim`.
2. Konfigurieren Sie Plugins, indem Sie oben in `.vimrc` hinzufügen und ggf. Plugins hinzufügen oder entfernen (die Plugins in der Liste dienen nur zur Veranschaulichung)

```
set nocompatible          " be iMproved, required
filetype off              " required

" set the runtime path to include Vundle and initialize
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()
" alternatively, pass a path where Vundle should install plugins
"call vundle#begin('~/.vim/bundle')

" let Vundle manage Vundle, required
Plugin 'VundleVim/Vundle.vim'

" All of your Plugins must be added before the following line
call vundle#end()          " required
filetype plugin indent on  " required
" To ignore plugin indent changes, instead use:
"filetype plugin on

"place non-Plugin stuff after this line
```

3. Installieren Sie Plugins: indem Sie Vim starten und `:PluginInstall`.

Unterstützte Plugin-Formate

Nachfolgend finden Sie Beispiele für verschiedene unterstützte Formate. Behalten Sie die Plugin-Befehle zwischen `vundle#begin` und `vundle#end`.

Plugin-Speicherort	Verwendungszweck
Plugin auf GitHub	Plugin 'tpope/vim-fugitive'
Plugin von http://vim-scripts.org/vim/scripts.html	Plugin 'L9'
Git-Plugin nicht auf GitHub gehostet	Plugin 'git://git.wincent.com/command-t.git'
Git Repos auf Ihrem lokalen Rechner (zB wenn Sie	Plugin 'file:///home/gmarik/path/to/plugin'

Plugin-Speicherort	Verwendungszweck
an Ihrem eigenen Plugin arbeiten)	
Das sparkup vim-Skript befindet sich in einem Unterverzeichnis dieses Repos namens vim. Übergeben Sie den Pfad, um den Laufzeitpfad richtig einzustellen.	Plugin 'rstacruz/sparkup', {'rtp': 'vim/'}
Installieren Sie L9 und vermeiden Sie einen Namenskonflikt, wenn Sie bereits eine andere Version installiert haben.	Plugin 'ascenator/L9', {'name': 'newL9'}

Das Arbeiten an einem gemeinsam genutzten Konto, z. B. auf einem Cluster- `.vim` kann Probleme verursachen, wenn das `.vim` Verzeichnis die Datenträgerverwendung verwendet. Es gibt einige Pakete, die viel Speicherplatz **benötigen**, beispielsweise **YCM**. Wählen Sie Ihr `Vundle` Plugin-Verzeichnis also mit `rtp`. `rtp` sehr einfach, indem Sie `rtp`. Und auch wenn Sie planen, ein vim-Plugin zu installieren, machen Sie `git clone` nicht direkt im `bundle` Verzeichnis. Benutze den Vundle-Weg.

Die Zukunft: Pakete

Siehe `:help packages`.

Erreger

vim-pathogen ist ein `runtimepath` Manager, der von Tim Pope erstellt wurde, um Plugins und Laufzeitdateien in ihren eigenen privaten Verzeichnissen zu installieren.

Pathogen installieren

1. Legen Sie den Erreger in `~/ .vim/bundle` (hier mit Git, aber es ist nicht zwingend erforderlich):

```
git clone https://github.com/tpope/vim-pathogen.git
```

2. Fügen Sie die folgenden Zeilen am oberen `.vimrc` Ihrer `.vimrc`:

```
" enable vim-pathogen
runtime bundle/vim-pathogen/autoload/pathogen.vim
execute pathogen#infect()
```

- Die `runtime` gibt den Pfad zum Autoload-Skript von `vim-pathogen`.
- `execute pathogen#infect()` initiiert es.

Nach dem Start startet Pathogen automatisch einen Durchlauf durch die Ordner in `~/ .vim/bundle` und lädt das Plugin von jedem Ordner.

Pathogen verwenden

1. Legen Sie das oberste Verzeichnis Ihres Plugins in `~/.vim/bundle/` , um es beim nächsten Start von Vim verfügbar zu machen.
2. Run `:Helptags` um die Dokumentation Ihres neuen Plugins zu indizieren.

Leistungen

- Jedes Plugin befindet sich in einem eigenen Verzeichnis unter `~/.vim/bundle/` .
- Ihr `.vimrc` bleibt von der Konfiguration frei, die zum Laden von Plugins benötigt wird.

Der Aufwand zum "Verwalten" eines Plugins reduziert sich somit auf:

- **legt** sein Top-Level - Verzeichnis unter `~/.vim/bundle/` zu *installieren*,
- **das** oberste Verzeichnis **ersetzen** , um es zu *aktualisieren* ,
- **Löschen Sie** das **Verzeichnis** der obersten Ebene, um es zu *deinstallieren* .

Wie Sie diese drei Aktionen ausführen (manuell, über ein Automatisierungswerkzeug, mit Git / Svn / Hg / was auch immer...), liegt ganz bei Ihnen.

Vim erweitern online lesen: <https://riptutorial.com/de/vim/topic/3659/vim-erweitern>

Kapitel 45: Vim konfigurieren

Examples

Die vimrc-Datei

Die `.vimrc` Datei (ausgesprochen Vim-wreck) ist eine Vim-Konfigurationsdatei. Es enthält Befehle, die von Vim bei jedem Start ausgeführt werden.

Standardmäßig ist die Datei leer oder nicht vorhanden. Sie können es verwenden, um Ihre Vim-Umgebung anzupassen.

Um herauszufinden, wo Vim erwartet, dass die Vimrc-Datei gespeichert wird, öffnen Sie Vim und führen Sie Folgendes aus:

```
:echo $MYVIMRC
```

Unix: Auf einem Unix-System wie Mac oder Linux heißt Ihre vimrc `.vimrc` und befindet sich normalerweise in Ihrem Home-Verzeichnis (`$HOME/.vimrc`).

Windows: Unter Windows heißt es `_vimrc` und befindet sich in Ihrem Home-Verzeichnis (`%HOMEPATH%/_vimrc`).

Beim Start sucht Vim an mehreren Stellen nach einer Vimrc-Datei. Das erste, was existiert, wird verwendet, die anderen werden ignoriert. Eine vollständige Referenz finden Sie im `:h $MYVIMRC` Dokumentationsartikel.

Welche Optionen kann ich verwenden?

Wenn Sie nicht wissen, welche Optionen Sie verwenden sollen, sind Sie möglicherweise an dem Befehl `:options` interessiert.

Dadurch wird ein Split geöffnet, in dem alle Vim-Optionen aufgelistet sind und der aktuelle Wert angezeigt wird. Es gibt 26 Abschnitte, in denen alle Optionen angezeigt werden, die Sie ausprobieren können.

z.B

```
4 displaying text

scroll      number of lines to scroll for CTRL-U and CTRL-D
            (local to window)
            set scr=20
scrolloff   number of screen lines to show around the cursor
            set so=5
wrap        long lines wrap
            set nowrap    wrap
```


...

In einer `set nowrap` (zB `set nowrap`) können Sie `CR` drücken, um den Wert `set nowrap` (wenn es sich um einen binären Wert handelt). In einer Optionszeile (z. B. `wrap long line wrap`) können Sie `CR` drücken, um auf die Dokumentation für diese Option zuzugreifen.

Dateien und Verzeichnisse

Was auch immer Sie tun, um Vim anzupassen, es sollte NIE außerhalb von `$HOME` passieren:

- Unter Linux, BSD und Cygwin ist `$HOME` normalerweise `/home/username/` ,
- Unter Mac OS X ist `$HOME` `/Users/username/` ,
- Unter Windows ist `$HOME` normalerweise `C:\Users\username\` .

Der kanonische Speicherort für Ihr `vimrc` und Ihr `vim` Verzeichnis befindet sich im Stammverzeichnis dieses `$HOME` Verzeichnisses:

- auf Unix-ähnlichen Systemen

```
$HOME/.vimrc      <-- the file
$HOME/.vim/       <-- the directory
```

- unter Windows

```
$HOME\_vimrc      <-- the file
$HOME\vimfiles\   <-- the directory
```

Das obige Layout funktioniert garantiert jetzt und in der Zukunft.

Vim 7.4 hat es möglich gemacht, Ihre schöne `vimrc` in Ihrem `vim` Verzeichnis zu behalten. Es ist wirklich eine gute Idee, schon weil es einfacher ist, die Konfiguration zu verschieben.

Wenn Sie ausschließlich 7.4 verwenden, reicht Folgendes aus:

- auf Unix-ähnlichen Systemen

```
$HOME/.vim/vimrc
```

- unter Windows

```
$HOME\vimfiles\vimrc
```

Wenn Sie die Vorteile eines in sich geschlossenen `vim/` nutzen möchten, aber sowohl 7.4 als auch eine ältere Version oder nur eine ältere Version verwenden möchten, besteht die einfachste und zukunftsicherste Lösung darin, diese Zeile *und nur diese Zeile* anzugeben:

```
runtime vimrc
```

in dieser Datei:

- auf Unix-ähnlichen Systemen

```
$HOME/.vimrc
```

- unter Windows

```
$HOME\_vimrc
```

und führen Sie Ihre Konfiguration in `$HOME/.vim/vimrc` oder `$HOME\vimfiles\vimrc` .

Optionen

Es gibt drei Arten von Optionen:

- boolesche Optionen,
- String-Optionen,
- Anzahl Optionen.

Um den Wert einer Option zu überprüfen,

- Verwendung `:set option?` den Wert einer Option prüfen,
- Verwendung `:verbose set option?` zu sehen, wo es zuletzt eingestellt wurde.

Boolesche Optionen einstellen

```
set booloption      " Set booloption.
set nobooloption    " Unset booloption.

set booloption!     " Toggle booloption.

set booloption&     " Reset booloption to its default value.
```

String-Optionen einstellen

```
set stroption=baz   " baz

set stroption+=buzz " baz,buzz
set stroption^=fizz " fizz,baz,buzz
set stroption-=baz  " fizz,buzz

set stroption=      " Unset stroption.

set stroption&     " Reset stroption to its default value.
```

Anzahl Optionen einstellen

```
set numoption=1      " 1
set numoption+=2     " 1 + 2 == 3
set numoption-=1     " 3 - 1 == 2
set numoption^=8     " 2 * 8 == 16
```

Einen Ausdruck als Wert verwenden

- mit Verkettung:

```
execute "set stroption=" . my_variable
```

- mit `:let` :

```
let &stropion = my_variable
```

Siehe `:help :set` und `:help :let` .

Zuordnungen

- Fügen Sie nach den Zuordnungen keine Kommentare ein, da dies zu einem Bruch führt.
- Verwenden Sie `:map <F6>` zu sehen, was in welchem Modus auf `<F6>` abgebildet ist.
- Verwenden Sie `:verbose map <F6>` um auch zu sehen, wo sie zuletzt zugeordnet wurde.
- `:map` und `:map!` sind zu generisch. Verwenden Sie `:n[nore]map` für Normalmoduszuordnungen, `:i[nore]map` für Einfügemodus `:x[nore]map` für visuellen Modus usw.

Rekursive Zuordnungen

Verwenden Sie *rekursive* Zuordnungen **nur**, wenn Sie beabsichtigen, andere Zuordnungen in Ihren Zuordnungen zu verwenden:

```
nnoremap b      B
nmap    <key> db
```

In diesem Beispiel arbeitet `b` im Normalmodus wie `B` . Da wir `b` in einem *rekursiven* Mapping verwenden, funktioniert das Drücken von `<key>` wie `dB` und nicht wie `db` .

Nicht-rekursive Zuordnungen

Verwenden Sie *nicht-rekursive* Zuordnungen **nur**, wenn Sie in Ihren Zuordnungen Standardbefehle verwenden möchten. Dies ist fast immer das, was Sie möchten:

```
nnoremap <key> db
```

In diesem Beispiel verwenden wir `b` in einem *nichtrekursiven* Mapping, sodass das Drücken der

Taste immer wie `db` funktioniert, unabhängig davon, ob wir `b` oder nicht.

Befehl aus einem Mapping ausführen

```
nnoremap <key> :MyCommand<CR>
```

Mehrere Befehle aus einem Mapping ausführen

```
nnoremap <key> :MyCommand <bar> MyOtherCommand <bar> SomeCommand<CR>
```

Aufruf einer Funktion aus einem Mapping

```
nnoremap <key> :call SomeFunction()<CR>
```

Zuordnen einer `<Plug>` -Zuordnung

```
map <key> <Plug>name_of_mapping
```

Siehe `:help map-commands`, `:help key-notation` und `:help <plug>`.

Weitere Informationen finden Sie unter Tastenzuordnungen [in Vim](#)

Variablen

Wie die meisten Skriptsprachen verfügt Vimscript über Variablen.

Man kann eine Variable mit dem Befehl: `:let` :

```
let variable = value
```

und lösche es mit `:unlet` :

```
unlet variable
```

In Vim können Variablen festgelegt werden, indem ihrem Namen ein einzelner Buchstabe und ein Doppelpunkt vorangestellt wird. Plugin-Autoren verwenden diese Funktion, um Optionen nachzuahmen:

```
let g:plugin_variable = 1
```

Siehe `:help internal-variables`.

Befehle

- Vergessen Sie nicht den Knall, damit Vim diesen Befehl beim nächsten Laden Ihres vimrc überschreiben kann.
- Benutzerdefinierte Befehle müssen mit einem Großbuchstaben beginnen.

Beispiele

```
command! MyCommand call SomeFunction()
command! MyOtherCommand command | Command | command
```

- Siehe `:help user-commands`.

Funktionen

- Vergessen Sie nicht den Knall, damit Vim diese Funktion das nächste Mal überschreiben kann, wenn Sie das Skript neu laden, in dem die Funktion definiert ist.
- Benutzerdefinierte Funktionen müssen entweder mit einem Großbuchstaben (globale Funktionen) oder mit `s:` (Skriptlokale Funktionen) beginnen oder ihnen muss der Name vorangestellt werden, der dem Autoload-Plugin zugeordnet ist, wo sie definiert sind (z. B. in `{&rtp}/autoload/foo/bar.vim` könnten wir `foo#bar#functionname()` definieren).
- Um die Parameter in der Funktion verwenden zu können, verwenden Sie `a:parameter_name`. Mit den Auslassungspunkten `...` können variadische Funktionen definiert werden. Für den Zugriff auf die Parameter verwenden Sie `a:000` (Liste aller Parameter) oder `a:0` (Anzahl der Parameter entspricht `len(a:000)`), `a:1` zuerst unbenannt Parameter und so weiter.
- Funktionen können wie `:call MyFunction(param1, param2)`
- Jede Zeile in einer Funktion beginnt implizit mit ein `:`, damit alle Befehle Doppelpunkt - Befehle
- Um zu verhindern, dass die Funktion im Fehlerfall ihre Ausführung fortsetzt, sollten Sie die Funktionssignatur am besten mit `abort`

Beispiel

```
function! MyFunction(foo, bar, ... ) abort
    return a:foo . a:bar . (a:0 > 0 ? a:1 : '')
endfunction
```

Skriptfunktionen

Wenn Sie planen, Ihre Funktion nur in der Datei zu verwenden, in der sie definiert ist (entweder weil Sie eine größere Funktion in kleineren Teilen beschädigt haben oder weil Sie sie in einem Befehl, einem Mapping usw. verwenden), können Sie das Präfix verwenden es mit `s:`, um zu vermeiden, dass Ihr globaler Namespace mit nutzlosen internen Funktionen verschmutzt wird:

```
function! s:my_private_function() " note we don't need to capitalize the first letter this time
    echo "Hi!"
endfunction
```

S: Funktionen aus Zuordnungen verwenden

Wenn Ihre lokale Skriptfunktion in einem Mapping verwendet werden soll, müssen Sie sie mit dem speziellen Präfix `<SID>` referenzieren:

```
nnoremap <your-mapping-key> :call <SID>my_private_function()<CR>
```

Siehe `:help user-functions`.

Beachten Sie jedoch, dass es seit Vim 7 eine bewährte Methode ist, Mappings, Befehle und Menüs in (ft) -Plugins und Funktionen für das automatische Laden von Plugins zu definieren - mit Ausnahme der Funktionen, die die Plugins beim Laden benötigen. Dies bedeutet, dass heutzutage die Notwendigkeit, lokale Funktionen von Skripten über Zuordnungen aufzurufen, nicht so relevant ist wie zuvor.

Autobefehlsgruppen

- Autocommand-Gruppen sind gut für die Organisation, sie können jedoch auch für das Debugging nützlich sein. Stellen Sie sich diese als kleine Namespaces vor, die Sie nach Belieben aktivieren / deaktivieren können.

Beispiel

```
augroup MyGroup
  " Clear the autocmds of the current group to prevent them from piling
  " up each time you reload your vimrc.
  autocmd!

  " These autocmds are fired after the filetype of a buffer is defined to
  " 'foo'. Don't forget to use 'setlocal' (for options) and '<buffer>'
  " (for mappings) to prevent your settings to leak in other buffers with
  " a different filetype.
  autocmd FileType foo setlocal bar=baz
  autocmd FileType foo nnoremap <buffer> <key> :command<CR>

  " This autocmd calls 'MyFunction()' everytime Vim tries to create/edit
  " a buffer tied to a file in '/path/to/project/**/'.
  autocmd BufNew,BufEnter /path/to/project/**/* call MyFunction()
augroup END
```

Siehe `:help autocommand`.

Conditionals

```
if v:version >= 704
  " Do something if Vim is the right version.
endif

if has('patch666')
  " Do something if Vim has the right patch-level.
endif
```

```
if has('feature')
    " Do something if Vim is built with 'feature'.
endif
```

Siehe `:help has-patch` und `:help feature-list`.

Einstellmöglichkeiten

Normalerweise verwenden Sie `:set .vimrc` Sie fest, welche Optionen in Ihrem `.vimrc` nach Ihren Wünschen `.vimrc`. Es gibt viele Optionen, die geändert werden können.

Zum Beispiel, um Leerzeichen für die Einrückung zu verwenden:

```
:set expandtab
:set shiftwidth=4
:set softtabstop=4
```

Satzstellung markieren

Aktivieren Sie die Syntaxhervorhebung, wenn das Terminal Farben hat

```
if &t_Co > 2 || has("gui_running")
    syntax on
end
```

Nachgestellte Leerzeichen und Tabulatoren anzeigen. Das Anzeigen von Registerkarten kann besonders bei der Suche nach Fehlern in Makefiles hilfreich sein.

```
set list listchars=tab:\|_,trail:.
highlight SpecialKey ctermfg=DarkGray
```

Farbschemata

Vim verfügt über mehrere vorinstallierte Farbschemata. In Linux werden die mit Vim `/usr/share/vim/vim74/colors/` Farbschemas in `/usr/share/vim/vim74/colors/` gespeichert (wobei 74 Ihre Versionsnummer ist, mehrere Zeiträume). MacVim speichert sie in `/Applications/MacVim.app/Contents/Resources/vim/runtime/colors`.

Farbschemas ändern

Der Befehl `colorscheme` wechselt das aktuelle Farbschema.

Um zum Beispiel das Farbschema auf "Robokai" zu setzen:

```
:colorscheme robokai
```

Das Standardfarbschema hat den kreativen Namen `default` , um es wieder zu verwenden

```
:colorscheme default
```

Um alle derzeit installierten Farbschemata `:colorscheme` , geben Sie `:colorscheme` gefolgt von Leerzeichen und dann entweder `Tab` oder `Strg d` .

Farbschemas installieren

`~/.vim/colors/` Benutzer installierte Farbschemata können in `~/.vim/colors/` . Wenn ein Farbschema zu diesem Verzeichnis hinzugefügt wird, wird es als Option für den Befehl `colorscheme` .

Um neue Farbschemata zu finden, gibt es Standorte wie [Vimcolors](#), die verschiedene Farbschemata enthalten. Es gibt auch Tools wie [vim.ink](#) und [Vivify](#) , die Sie beim Erstellen Ihrer eigenen Farbschemas unterstützen, oder Sie können sie von Hand erstellen.

Zeilenanzählung umschalten

Um es zu aktivieren, geben Sie Folgendes ein:

```
:set number oder :set nu .
```

Zu deaktivieren - geben Sie ein:

```
:set nonumber oder :set nonu .
```

Um die Anzählung *relativ* zur Cursorposition zu aktivieren, geben Sie Folgendes ein:

```
:set relativenumber .
```

Um die Anzählung *relativ* zur Cursorposition zu deaktivieren, geben Sie Folgendes ein:

```
:set norelativenumber .
```

Hinweis: Um zu ändern, ob in der aktuellen Zeile die tatsächliche Zeilennummer oder 0 `:set nonumber` Befehle `:set number` und `:set nonumber` , während das Attribut `relativenumber` aktiv ist.

Plugins

Vim-Plugins sind Addons, mit denen die Funktionalität von vim geändert oder erweitert werden kann.

[Vimawesome](#) bietet eine gute Liste von Plugins

Vim konfigurieren online lesen: <https://riptutorial.com/de/vim/topic/2235/vim-konfigurieren>

Kapitel 46: Vim verlassen

Parameter

Parameter	Einzelheiten
:	Aktivieren Sie den Befehlszeilenmodus
w	Schreiben
q	Verlassen
a	Alles
!	Überschreiben

Bemerkungen

Der Befehlszeilenmodus wird im normalen Modus aufgerufen. Sie werden wissen, dass Sie sich im Befehlszeilenmodus befinden, wenn in der unteren linken Ecke des Terminalfensters ein `:` angezeigt wird.

Der Normalmodus ist der Standardmodus von `vi` / `vim` und kann durch Drücken der `ESC`-Taste umgeschaltet werden.

`Vi` / `Vim` verfügt über integrierte Prüfungen, um zu verhindern, dass nicht gespeicherte Arbeit verloren geht, und andere nützliche Funktionen. Um diese Prüfungen zu umgehen, verwenden Sie die Überschreibung `!` in deinem Befehl

In `Vi` / `Vim` können mehrere Dateien gleichzeitig (in verschiedenen Fenstern) gleichzeitig angezeigt werden. Verwenden Sie `a`, um alle geöffneten Fenster zu schließen.

Examples

Beenden Sie mit Speichern

```
: wq
```

```
ZZ
```

Beenden ohne zu speichern

```
: q!
```

Zwangsweise beenden (ohne Speichern)

: q!

ZQ

Zwangsweise beenden (mit Speichern)

: wq!

Beenden Sie das Öffnen aller Fenster (ohne Speichern)

: qa!

wenn mehrere Dateien geöffnet sind

```
:wqall
```

Mehrere Dateien mit dem Speichern von Inhalten beenden

```
:qall!
```

Mehrere Dateien ohne Speichern des Inhalts beenden

Vim verlassen online lesen: <https://riptutorial.com/de/vim/topic/5074/vim-verlassen>

Kapitel 47: Vim-Optionen

Syntax

- `:set [no] (option|shortcut)`
- `:set (option|shortcut)=value`
- `:set (option|shortcut) (?|&)`
- Verwenden Sie nicht `:` in der Vimrc-Datei

Bemerkungen

Siehe [Video zu Vimcast 1](#)

Siehe [vimcast 1-Transkription](#)

Examples

einstellen

Um die Optionen einzustellen, verwenden Sie `:set` Anweisung setzen. Beispiel:

```
:set ts=4
:set shiftwidth=4
:set expandtab
:set autoindent
```

So zeigen Sie den aktuellen Wert der Option an `:set {option}? .` Beispiel:

```
:set ts?
```

Um den Wert der Option auf den Standardwert zurückzusetzen, geben Sie Folgendes ein `:set {option}& .` Beispiel:

```
:set ts&
```

Vertiefung

Breite

Einrückungen mit 4 Feldern machen:

```
:set shiftwidth=4
```

Räume

Leerzeichen als Einrückungen verwenden, 4 Leerzeichen breit:

```
:set expandtab
:set softtabstop=4
```

`softtabstop` und `sts` sind äquivalent:

```
:set sts=4
```

Tabs

Um Tabulatoren als Einrückungen zu verwenden, 4 Felder breit:

```
:set noexpandtab
:set tabstop=4
```

`tabstop` und `ts` sind gleichwertig:

```
:set ts=4
```

Automatische Einrückung

```
:set autoindent
```

Anweisungsbeschreibungen

Anweisung	Beschreibung	Standard
Tabulator	Breite des Tabulatorzeichens	8
expandtab	bewirkt, dass Leerzeichen anstelle von Tabulatorzeichen verwendet werden	aus
Softtabstop	stimmen Sie den Whitespace ab	0
Verschiebungsbreite	bestimmt den Leerraum im normalen Modus	8

Unsichtbare Zeichen

Unsichtbare Zeichen anzeigen oder ausblenden

Um unsichtbare Zeichen anzuzeigen:

```
:set list
```

Um unsichtbare Zeichen auszublenden:

```
:set nolist
```

Um zwischen unsichtbaren Zeichen ein- und auszublenden:

```
:set list!
```

Standardsymbolzeichen

Symbol	Charakter
^ Ich	Tab
\$	Neue Zeile

Anpassen von Symbolen

Um das Tabulatorzeichen auf ****> **** und das neue **Zeilenzeichen** auf **↵** zu setzen

```
set listchars=tab:>\ ,eol:↵
```

Um die Leerzeichen auf **_** zu setzen

```
set listchars=spaces
```

Um eine Liste der Zeichenoptionen anzuzeigen

```
:help listchars
```

Vim-Optionen online lesen: <https://riptutorial.com/de/vim/topic/2407/vim-optionen>

Kapitel 48: Vim-Register

Parameter

Funktionalität	Register
Standardregister	" "
historische Register	" [1-9]
Register reißen	" 0
benannte Register	" [az] , " [AZ] wie " [az] jedoch angehängt
aktuelles Suchmuster abrufen	" /
kleine Löschungen (diw, cit, ...)	" _
Ausdrucksregister für einfache Mathematik	" =
Schwarzes Lochregister, um große Stücke von gelöschtem Text aus dem Mem zu löschen	" -
letzter Befehl	" :
zuletzt eingefügter Text	" .
Dateiname	" %
Zwischenablage	" *
ausgewählter Text	" +
Text fallen gelassen	" ~

Examples

Löschen Sie einen Zeilenbereich in einem benannten Register

Geben Sie in Normal Folgendes ein, um einen Zeilenbereich in einem benannten Register zu löschen

```
:10,20d a
```

Dadurch werden die Zeilen 10, 20 im Register "a gelöscht. Wir können dies durch Eingabe

überprüfen

```
:reg
```

Dadurch wird der Text angezeigt, der im Register "a gelöscht wurde.

Um den Inhalt in "a , geben Sie einfach ein

```
"ap
```

Fügen Sie den Dateinamen im Einfügemodus mit dem Dateinamenregister ein

Drücken Sie im Einfügemodus <Cr> und dann % , um den Dateinamen einzufügen.

Diese Technik ist auf alle Register anwendbar.

Wenn Sie beispielsweise das aktuelle Suchmuster im Einfügemodus einfügen möchten, können Sie <Cr> und dann / <Cr> .

Kopieren / Einfügen zwischen Vim und der Systemzwischenablage

Verwenden Sie das Anführungszeichen-Register, um zwischen Vim und der Systemzwischenablage zu kopieren / einfügen

"*yy kopiert die aktuelle Zeile in die Systemzwischenablage

"*p fügt den Inhalt der Systemzwischenablage in Vim ein

An ein Register anhängen

Ziehen Sie alle Zeilen, die TODO enthalten, mit der Anfügeoperation in ein Register

```
:global/TODO/yank A
```

Hier suchen wir global nach einem TODO Schlüsselwort und ziehen alle Zeilen in das Register a (A Register hängt alle Zeilen an a Register an).

HINWEIS: Im Allgemeinen ist es eine gute Praxis, ein Register zu löschen, bevor der Anfügevorgang ausgeführt wird.

Um ein Register zu löschen, geben Sie im normalen Modus qaq . Stellen Sie sicher, dass das a Register leer ist, indem Sie Folgendes eingeben :reg und beobachten, dass a Register leer ist.

Vim-Register online lesen: <https://riptutorial.com/de/vim/topic/4278/vim-register>

Kapitel 49: Vim-Ressourcen

Bemerkungen

Dieses Thema behandelt **Quellcode-Spiegel** , **Bücher** , **Vim-Wikis** . Es geht **NICHT** um Blogeinträge, Wikipedia, Tutorials. Die Ressourcen sollten nicht auf Meinungen basieren.

Examples

Vimscript auf die harte Tour lernen

Ein Buch, das erklärt, wie Vimscript funktioniert, voller Beispiele. Es kann auf <http://learnvimscriptthehardway.stevelosh.com/> gefunden werden.

Vim-Ressourcen online lesen: <https://riptutorial.com/de/vim/topic/6383/vim-ressourcen>

Kapitel 50: Vimscript

Bemerkungen

Die Befehle in einer Vimscript-Datei werden standardmäßig im `command mode` ausgeführt. Daher sollten alle Direktiven im Nicht- `command mode` vorangestellt werden.

Examples

Hallo Welt

Beim Versuch, etwas zum Debuggen in Vimscript zu drucken, ist es verführerisch, einfach die folgenden Schritte auszuführen.

```
echo "Hello World!"
```

Im Zusammenhang mit einem komplexen Plugin werden jedoch oft viele andere Dinge direkt nach dem Versuch ausgeführt, Ihre Nachricht zu drucken. Es ist daher wichtig, den `sleep` nach der Nachricht hinzuzufügen, damit Sie sie tatsächlich sehen können, bevor sie verschwindet.

```
echo "Hello World!"  
sleep 5
```

Verwenden der Befehle im Normalmodus in Vimscript

Da eine Vimscript-Datei eine Sammlung von Aktionen im Befehlsmodus ist, muss der Benutzer angeben, dass die gewünschten Aktionen im normalen Modus ausgeführt werden sollen.

Das Ausführen eines normalen Modusbefehls wie `i`, `a`, `d` usw. in Vimscript erfolgt durch Anhängen des Befehls an den `normal` Befehl:

Zum Ende der Datei gehen und die letzten 5 Zeilen auswählen:

```
normal GV5k
```

Hier weist das `G` an vim, die Cursorposition auf die letzte Zeile zu ändern, das `v`, um in den visuellen Modus für die Zeilenrichtung zu wechseln, und die `5k`, um 5 Zeilen nach oben zu gehen.

Einfügen Ihres Namens am Ende der Zeile:

```
normal ABoris
```

Dabei setzt `A` den Editor in den Einfügemodus am Ende der Zeile und der Rest ist der einzufügende Text.

Vimscript online lesen: <https://riptutorial.com/de/vim/topic/5136/vimscript>

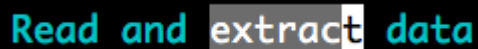
Kapitel 51: Vim-Textobjekte

Examples

Wählen Sie ein Wort ohne umgebende Leerzeichen aus

Angenommen, wir möchten ein Wort ohne umgebende Leerzeichen auswählen, verwenden Sie das `iw` für das innere Wort im visuellen Modus:

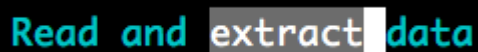
1. Durch Drücken von `ESC` in den Normalmodus wechseln
2. `viw` am Anfang eines Wortes `viw` ein
3. Dadurch wird das innere Wort ausgewählt



Wählen Sie ein Wort mit umgebendem Leerzeichen aus

Angenommen, wir möchten ein Wort mit einem umgebenden Leerzeichen auswählen, verwenden Sie das Textobjekt `aw` für ein Wort im visuellen Modus:

1. Durch Drücken von `ESC` in den Normalmodus wechseln
2. `vaw` am Anfang eines Wortes `vaw` ein
3. Dadurch wird das Wort mit Leerzeichen ausgewählt



Wählen Sie Text innerhalb eines Tags aus

Wir können einen Text innerhalb eines auswählen `html` oder `xml` - Tag unter Verwendung von visuellen Auswahl `v` und Text - Objekt `it` .

1. Gehen Sie in den Normalmodus, indem Sie `ESC` drücken
2. Geben Sie `vit` an einer beliebigen Stelle im `html` oder `xml` Abschnitt ein
3. Dadurch wird der gesamte Text innerhalb des `tag` visuell ausgewählt

```

11      <head>
10
9      <!-- Latest compiled an
8      <link rel="stylesheet"
7      <link rel="stylesheet"
6
5      <!-- Optional theme -->
4      <link rel="stylesheet"
>\css">
3      <!-- for datepicker -->
2      <link rel="stylesheet"
1
14  </head>
1

```

Alle anderen Textobjekte können auch verwendet werden, um den Text innerhalb des Tags zu bearbeiten

1. `cit` - lösche Text innerhalb des Tags und setze ihn in den `insert`
2. `dit` - löscht den Text innerhalb des Tags und verbleibt im `normal` Modus
3. `cat` - Rundum-Tag löschen und im `insert`
4. `dat` - löscht den Text um das Tag und verbleibt im `normal` Modus

Vim-Textobjekte online lesen: <https://riptutorial.com/de/vim/topic/4050/vim-textobjekte>

Kapitel 52: Vorteile von vim

Examples

Anpassung

Der Vorteil der Verwendung von **vim** gegenüber einem einfachen Texteditor wie **Notepad** oder **gedit** besteht darin, dass der Benutzer fast alles an sich selbst anpassen kann. Wenn Sie feststellen, dass Sie immer wieder eine Aktion ausführen, bietet vim eine Vielzahl von Funktionen, mit denen Sie diese Aktion schneller und einfacher durchführen können.

Die meisten gängigen IDEs wie **MS Visual Studio** oder **IntelliJ IDEA** bieten ihren Benutzern nützliche Verknüpfungen und sogar Anpassungen. Sie beziehen sich jedoch in der Regel auf bestimmte Aktionen, die in einem bestimmten Kontext üblich sind Situationen, ohne sich zu widersprechen. Es wäre vielleicht angenehm, C ++ - Programme in Visual Studio und Java in IntelliJ zu entwickeln, aber Sie würden dort keinen Python-Code schreiben. Dafür gibt es natürlich eine andere IDE, aber in vim können Sie die Sprache, die Sie möchten, ziemlich ohne bearbeiten die Bequemlichkeit zu verlieren.

Es gibt natürlich noch andere anpassbare Editoren, und ich kann nicht sagen, dass vim für jeden das Beste ist. Dies ist eine Frage der persönlichen Vorlieben. Ich glaube nicht, dass jemand argumentieren wird, dass **Emacs** das Maß an Anpassung dem von Vim unterlegen macht (und manche würden es auch anders sagen), aber man muss es wirklich selbst ausprobieren, um herauszufinden, was am besten zu Ihnen passt.

Einige Leute sagen, sie wollen nicht Monate damit verbringen, einen Editor zu lernen, nur um darin arbeiten zu können. Aber die meisten sind sich einig, dass es sich gelohnt hat. Für mich persönlich war es nie ein Problem, neues Wissen über vim zu lernen und effizienter zu werden, macht einfach Spaß. Und es gibt viel zu lernen.

Leicht

Vim ist (wie GNU Nano oder GNU Emacs) leicht. Es braucht keine grafische Oberfläche (wie x11, wayland & co).

Dies macht vim zu einem besten Systembetreuer. Sie können es mit ssh verwenden, und das ist wirklich wichtig auf wirklich kleinen Geräten, die keine grafische Oberfläche haben.

Das Programmieren und Verwalten von Remote-Servern wurde in den letzten Jahren immer wichtiger. Die Verwendung von vim (oder emacs) ist der beste Weg.

Im Gegensatz zu vielen IDEs bietet vim die Möglichkeit, mit vielen Arten von Dateien zu arbeiten, und das Schreiben eigener Befehle und Syntax hl ist einfach.

Und nicht zuletzt sollte ein vim-Benutzer vi verwenden können, das auf den meisten UNIX-Systemen vorinstalliert ist.

Vorteile von vim online lesen: <https://riptutorial.com/de/vim/topic/9653/vorteile-von-vim>

Kapitel 53: Whitespace

Einführung

So können Sie Leerzeichen aufräumen.

Bemerkungen

Siehe [vimcast 4-Transkription](#)

Examples

Löschen Sie nachfolgende Leerzeichen in einer Datei

Sie können nachfolgende Leerzeichen mit dem folgenden Befehl löschen.

```
:%s/\s\+$/e
```

Dieser Befehl wird wie folgt erklärt:

- Kommando-Modus aufrufen mit :
- Tun Sie dies für die gesamte Datei mit % (Standardeinstellung wäre für die aktuelle Zeile)
- Ersatzaktion s
- / Start des Suchmusters
- \s Leerzeichen
- \+ escaped + sign, ein oder mehrere Leerzeichen sollten übereinstimmen
- vor dem Zeilenende \$
- / Ende des Suchmusters, Beginn des Ersetzungsmusters
- / Ende des Ersetzungsmusters. Im Grunde durch nichts ersetzen.
- e Unterdrücken Sie Fehlermeldungen, wenn keine Übereinstimmung gefunden wird

Löschen Sie leere Zeilen in einer Datei

Sie können alle leeren Zeilen in einer Datei mit dem folgenden Befehl löschen: g / ^ \$ / d

Dieser Befehl wird wie folgt erklärt:

- Kommando-Modus aufrufen mit :
- g ist ein globaler Befehl, der in der gesamten Datei vorkommen soll
- / Start des Suchmusters
- Das Suchmuster der Leerzeile ist ^g
- / Ende des Suchmusters
- Ex-Befehl d löscht eine Zeile

Konvertieren Sie Tabs in Leerzeichen und Leerzeichen in Tabs

Sie können Tabulatoren in Leerzeichen konvertieren, indem Sie folgende Schritte ausführen:

Prüfen Sie zunächst, ob `expandtab` ausgeschaltet ist

```
:set noexpandtab
```

Dann

```
:retab!
```

Leerzeichen einer bestimmten Länge werden durch Tabulatoren ersetzt

Wenn Sie `expandtab` wieder aktivieren, :set `expandtab` und führen Sie das `:retab!` Befehl, dann werden alle Registerkarten zu Leerzeichen.

Wenn Sie dies für ausgewählten Text ausführen möchten, wählen Sie den Text zunächst im [visuellen Modus](#) aus .

Whitespace online lesen: <https://riptutorial.com/de/vim/topic/8288/whitespace>

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit vim	A. Raza , akavel , Ashok , carrdelling , Christian Rondeau , Community , Cows quack , Daniel , Daniel Käfer , Daniel Margosian , Deborah V , depperm , ericdwang , ExistMe , GiftZwergrapper , gmoshkin , HerrSerker , James , Js Lim , KerDam , LittleByBlue , liuyang1 , LotoLo , Marek Skiba , Mattias , Miljen Mikic , mnoronha , Nasreddine , Nhan , Nick Weseman , pktangyue , redBit Device , Romain Vincent , romainl , ropata , Rory O'Kane , Sardathrion , sascha , SeekAndDestroy , sjas , sudo bangbang , Sumner Evans , tbodt , Tejus Prasad , TheMole , timss , Tom Gijssels , Tom Lord , user2314737 , user45891 , Vin , Vishnu Kumar , vvnraman , Wieland , Wojciech Kazior , zarak , Zaz
2	: global	cmlaverdiere , DJMcMayhem , LittleByBlue , tbodt , Vin
3	Auswechslung	cmlaverdiere , LittleByBlue , Nikola Geneshki , Timur
4	Autobefehle	joeytwiddle , tbodt , Tom Hale
5	Bauen von vim	grochmal , Josh Petrie , LittleByBlue , Luc Hermitte
6	Befehle im Normalmodus	Tom Hale
7	Befehle im Normalmodus (Bearbeitung)	A. Raza , rodericktech , romainl , The Nightman
8	Befehlszeilenbereiche	RamenChef , romainl
9	Bewegung	Aidan Miles , akavel , Boysenb3rry , Caek , Chris H , Cows quack , depperm , fedorqui , Georgi Dimitrov , gmoshkin , JacobLeach , jamessan , KerDam , Madis Pukkonen , Noam Hacker , rgoliveira , sjas , SnoringFrog , Sundeeep , timss , Tyler , Vin , Wazam , zarak
10	Bewegungen und Textobjekte	tbodt
11	Bitten Sie beim Erstellen einer neuen Datei um das Erstellen nicht vorhandener	Tom Hale

	Verzeichnisse	
12	Code automatisch formatieren	Philip Kirkbride
13	Dateityp-Plugins	Luc Hermitte , romainl
14	Der Punktoperator	Js Lim , LittleByBlue
15	Ex von der Kommandozeile aus verwenden	DJMcMayhem , Matt Clark
16	Falten	Josh Petrie , LittleByBlue , sudo bangbang
17	Fenster teilen	beardc , Boysenb3rry , Downgoat , Goluxas , grenangen , HerrSerker , Johnathan Andersen , KerDam , madD7 , Sachin Divekar , sudo bangbang , timss , Victor Schröder , zarak
18	Get: help (mit dem integrierten Handbuch von Vim)	Aidan Miles , Luc Hermitte
19	Konvertieren von Textdateien von DOS nach UNIX mit vi	grochmal , LazyBrush
20	Makros	Johan , Johnathan Andersen , lazysoundsystem , LittleByBlue , Oliver Wespi , rjmill , romainl , TheMole , Victor Schröder , vielmetti , Wenzhong
21	Modi - Einfügen, normal, visuell, z	rgoliveira , romainl
22	Nützliche Konfigurationen, die in .vimrc eingefügt werden können	Cows quack , maniacmic , Tomh
23	Ostereier	Aaron Thoma , Andrea Romagnoli , Christian Rondeau , cmlaverdiere , Daniel Käfer , Gerard Roche , LittleByBlue , Mateusz Piotrowski , Mattias , mcarton , nobe4 , NonlinearFruit , sudo bangbang , tbodt , Tejus Prasad
24	Plugins	Nick Weseman
25	Puffer	Chris Jones , eli , joeytwiddle , sudo bangbang
26	Rechtschreibprüfung	andipla , Johnathan Andersen , lwassink

27	Reguläre Ausdrücke	4444 , sudo bangbang
28	Reguläre Ausdrücke im Ex-Modus	UNagaswamy
29	Schlüsselzuordnungen in Vim	Christian Rondeau , Ingo Karkat , KerDam , Luc Hermitte , madD7 , New Alexandria , Nikola Geneshki , RamenChef , romainl
30	Scrollen	Delapouite , evuez
31	So kompilieren Sie Vim	Ingo Karkat , Josh Petrie , romainl
32	Solarized Vim	Luc Hermitte , Nick Weseman , satyanarayan rao
33	Speichern	abidibo
34	Suche im aktuellen Puffer	Abdelaziz Dabebi , DJMcMayhem , LittleByBlue , romainl
35	Suchen und Ersetzen	DJMcMayhem , Kara , naveen.panwar , ncmathsadist , romainl , sudo bangbang , zzz
36	Text bearbeiten	Chris Nager , Christopher Bottoms , LittleByBlue , Nikola Geneshki , Philip Kirkbride , romainl , till , Tom Hale , zarak
37	Text einfügen	Batsu , Boysenb3rry , Christopher Bottoms , cmlaverdiere , codefly , DJMcMayhem , Eric Bouchut , GiftZwergrapper , gmoshkin , Johnathan Andersen , Kent , lazysoundsystem , Mahmood , omul , Promarbler , RamenChef , rodrigo , romainl , satyanarayan rao , Scroff , SnoringFrog , sudo bangbang , Sundeep , timss , Tom Lord , UNagaswamy , Xavier Nicollet
38	Tipps und Tricks zur Steigerung der Produktivität	Abdelaziz Dabebi , adelarsq , Chris Midgley , depperm , GanitK , gath , gmoshkin , Hotschke , KerDam , LittleByBlue , LotoLo , mash , naveen.panwar , RamenChef , rjmill , romainl , Simone Bronzini , souparno majumder , Stryker , sudo bangbang , tbodt , Tom Hale
39	Unterschiede zwischen Neovim und Vim	still_dreaming_1 , tbodt
40	Verbessertes Rückgängigmachen und Wiederherstellen mit einem Undodir	GiftZwergrapper
41	Vertiefung	dallyingllama , Daniel , RamenChef , toto21

42	Verwenden von Python für Vim-Scripting	Nikola Geneshki
43	vglobal: Führt Befehle in Zeilen aus, die nicht global übereinstimmen	UNagaswamy
44	Vim erweitern	baptistemm , LittleByBlue , Nikola Geneshki , romainl , satyanarayan rao , Sumner Evans , void
45	Vim konfigurieren	Aaron Thoma , bn. , Christian Rondeau , Cometsong , Cows quack , Daniel , Johnathan Andersen , KerDam , Luc Hermitte , lwassink , mezzode , nobe4 , romainl , SnoringFrog , sudo bangbang , Sumner Evans , timss , Wojciech Kazior , Yosh
46	Vim verlassen	Arulpandiyan Vadivel , asclepix , AWippler , Nick Weseman , Yosef Nasr
47	Vim-Optionen	dallyingllama , LittleByBlue , mezzode , Wojciech Kazior , Yosh
48	Vim-Register	maniacmic , romainl , UNagaswamy
49	Vim-Ressourcen	Nikola Geneshki
50	Vimscript	merlin2011 , Nikola Geneshki
51	Vim-Textobjekte	UNagaswamy
52	Vorteile von vim	gmoshkin , LittleByBlue
53	Whitespace	dallyingllama