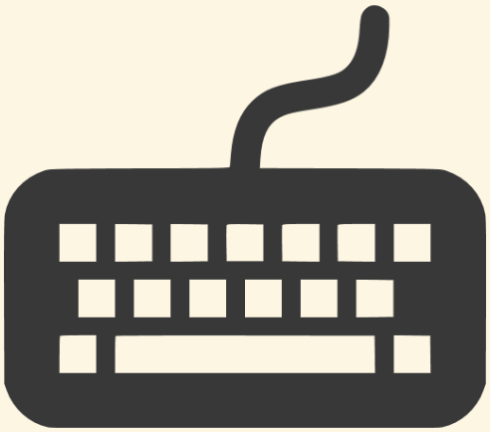
Hochschule Augsburg

Usability engineering



Modale tastaturorientierte User Interfaces

Student:
Christoph Piechula

Dozenten:
Prof. Dr. Christian Märtin
Christian Herdin
Jürgen Engel

14. Dezember 2015

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 | Einleitung

Heutzutage ist die Bedienung des Computers mit der Maus oder dem Touchscreen mit Hilfe von Gesten üblich. Tastaturorientierte Bedienkonzepte gelangen oft in den Hintergrund, da viele neue Consumer–Geräte einerseits keine physikalische Tastatur mehr mitbringen andererseits die Anwendungen immer stärker auf das Arbeiten mit Maus oder dem Touchscreen ausgelegt sind.

Inhalte werden heute vermehrt auf dem Smartphone oder Tablet konsumiert. Für das Surfen im Internet oder das Betrachten von Photoalben sind Touchscreens nach Meinung des Autors gut geeignet. Anders beim Arbeitsrechner, welcher in erster Linie dazu dient Inhalte zu generieren beziehungsweise bestimmte Arbeiten zu verrichten, hier ist die Bedienung mit Maus oder dem Touchscreen nicht immer vorteilhaft.

Je nach Branche und benutzten Applikationen ist manchmal nach Meinung des Autors sogar eine rein tastaturorientierte Bedienung des PCs vorteilhaft. Hier wäre beispielsweise der Bereich der Informatik (Softwareentwicklung, Systemadministration unter *Unix*, et cetera) aber auch das Ausarbeiten von wissenschaftlichen Publikationen (beispielsweise mit LaTeX) zu nennen. Da bei tastaturorientierter Steuerung oft neben grafischen Oberflächen[1] auch zeichenorientierte Anwendungen[2] zum Einsatz kommen, sind diese ebenso fester Bestandteil dieser Ausarbeitung.

Die Studienarbeit soll modale und tastaturorientierte Bedienkonzepte vorstellen und die Möglichkeiten sowie Vor– und Nachteile welche sich dadurch ergeben an Hand von Beispielen demonstrieren. Des Weiteren sollen Anregungen für alternative Arbeitsweisen geschaffen werden.

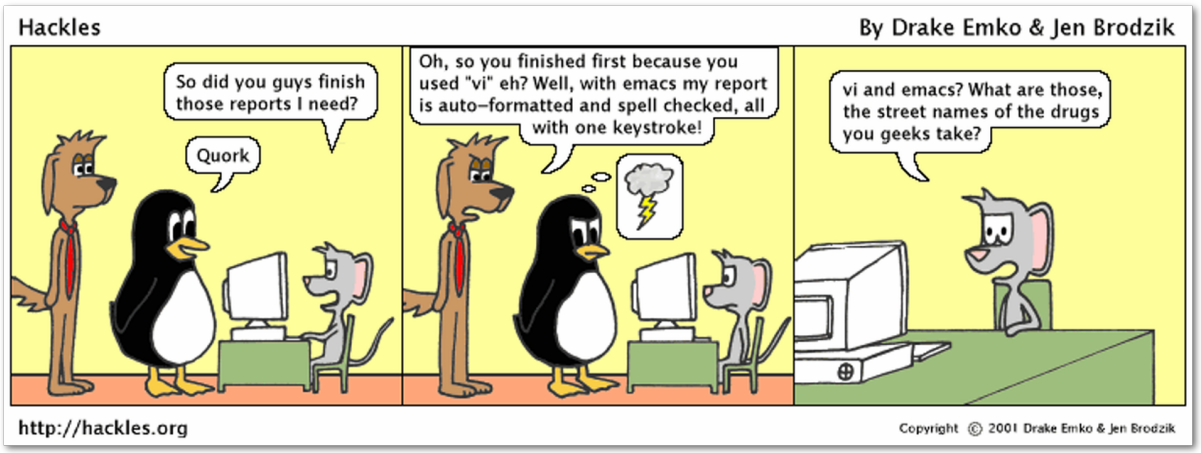


Abbildung 1.1: Der „vi“ ist ein unter *Unix* bekannter Editor welcher druchaus für Poweruser interessante Vorteile bieten kann.[3]

[1] Grafische Benutzerschnittstelle: https://de.wikipedia.org/wiki/Grafische_Benutzeroberfl%C3%A4che
[2] Zeichenorientierte Benutzerschnittstelle: https://de.wikipedia.org/wiki/Zeichenorientierte_Benutzerschnittstelle
[3] Hackles Comic, Quelle: http://www.hackles.org/strips/cartoon94.png

Dies soll am Beispiel des modalen Texteditors *Vim* (auch „vi“ genannt, siehe Comic Abbildung 1.1), des Fenstermanagers *i3* und des Terminal Multiplexers *Tmux* erfolgen. Die Studienarbeit kann jedoch aufgrund des Umfangs leider nur einen Überblick über die genannten Konzepte und Werkzeuge vermitteln.

Das Zielpublikum ist hier aus Sicht der Usability weniger der Otto–Normal–User, viel mehr dienen die genannten Werkzeuge, welche auf tastaturorientiertes Arbeiten ausgelegt sind, der Effizienz bei verschiedenen Tätigkeiten.

Studien zeigen, dass textbasierte User Interfaces — verglichen mit grafischen User Interfaces, sogenannten GUIs — nicht zwangsläufig schlechter sein müssen. Lediglich Gelegenheitsnutzer haben Vorteile bei der Verwendung von grafischen User Interfaces (vgl. [1]).

2 | Tastaturorientierte User Interfaces

Tastaturorientierte Benutzerschnittstellen waren in den Anfangszeiten des Personal Computer Zeitalters sehr populär. Da hier beispielsweise Terminals ohne grafische Möglichkeiten — sogenannte zeichenorientierte Benutzerschnittstellen — als User Interfaces dienten.

Erst Anfang der achtziger Jahre kamen grafische User Interfaces auf den Markt (vgl. [2]). Mit dem Aufkommen der grafischen Oberflächen änderten sich die Bedienmöglichkeiten. Heutzutage werden neben der Tastatur hauptsächlich Computermäuse, beziehungsweise Touchpads oder Trackpads bei Laptops, für die gängigen Arbeiten am Computer verwendet. Des Weiteren gibt es für bestimmte Anwendungen noch spezielle Eingabehardware wie grafische Tablets et cetera. Die Maus dient heutzutage neben der Tastatur als „das Eingabegerät“ schlecht hin.

Für viele Anwendungen bietet die Steuerung mit der Maus/Touchscreen eindeutige Vorteile, hier wären beispielsweise Anwendungen wie *Photoshop* oder auch Computerspiele zu nennen.

Neben diesen Anwendungen, welche eindeutig von der Maussteuerung profitieren, gibt es jedoch auch eine Reihe von Anwendungen beziehungsweise Anwendungsgebiete welche ursprünglich für ein tastaturorientiertes Arbeiten ausgelegt waren, heutzutage jedoch größtenteils auch über die Mauseingaben gesteuert werden. Zu nennen wäre hier beispielsweise in erster Linie das Verfassen und Manipulieren von Texten jeglicher Art.

Da die rein textuelle Beschreibung der Nutzbarkeit der tastaturorientierten Werkzeuge (*Vim*, *i3* und *Tmux*) diesen nicht gerecht werden würde, sind im Dokument Links zu Screencasts angegeben, welche für das Verständnis der Arbeitsweise mit Editor wichtig sind. Insgesamt wurden ca. 20 Minuten Screencasts erstellt. Die Screencasts sind eingerahmt und mit einem V–Symbol (*Vimeo* Videoplattform) versehen. Alle Screencasts auf *Vimeo* sind mit dem Passwort `ue2015` versehen.

3 | Der Texteditor „vi“

Der „vi“ ist ein modaler Texteditor. Das modale Bedienkonzept des „vi“ ist sehr ungewohnt. Es ist für Neulinge schwer vorstellbar wie ein effizientes Arbeiten mit diesem Editor ohne Maus möglich sein soll.

3.1 | Historisches zum „vi“

Der Editor „vi“ wurde Ende der siebziger Jahre von Bill Joy — einem Informatik–Absolventen — entwickelt (vgl. [3], [4]).

Der „vi“ wurde in einer Zeit entwickelt, in der die User Interfaces hauptsächlich aus textbasierten Interfaces bestanden. Das auch heute noch vorliegende Tastatur–Bedienkonzept beim *Vim* (moderner Nachfolger vom „vi“) ergibt sich durch den Umstand, dass der „vi“ für die Benutzung eines *ADM–3A* Computer Terminals entwickelt wurde und aus diesem Grund auch bestimme „Bedieneigenschaften“ vom Tastatur–Layout des *ADM-3A* erbt (vgl. [5]). Die Abbildung 3.1 zeigt das Layout des *ADM–3A*.

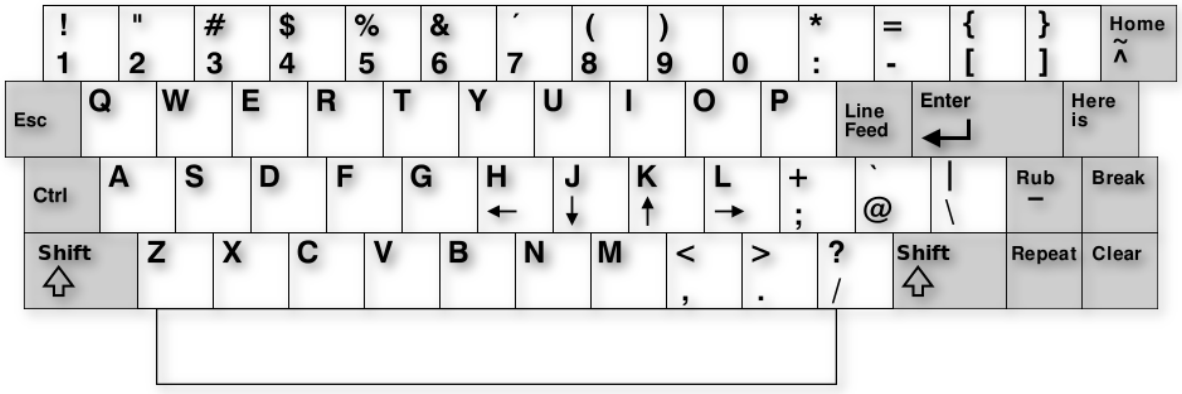


Abbildung 3.1: ADM-3A Keyboard Layout.[1]

3.2 | „Vi“ improved

Der heutzutage anzutreffende Texteditor *Vim (VImproved)* ist eine verbesserte und freie Neuentwicklung des „vi“ Editors, welcher von Bram Moolenaar entwickelt wurde. Aufgrund seines modalen Konzeptes erzeugt *Vim* gerade bei der ersten Verwendung große Verwirrung und wird oft von Einsteigern als kompliziert oder sogar nicht funktionierend eingestuft. Abbildung 3.2 zeigt die Lernkurve verschiedener Texteditoren auf scherzhafte Art und Weise.

[1] Quelle: https://upload.wikimedia.org/wikipedia/commons/a/a0/KB_Terminal_ADM3A.svg

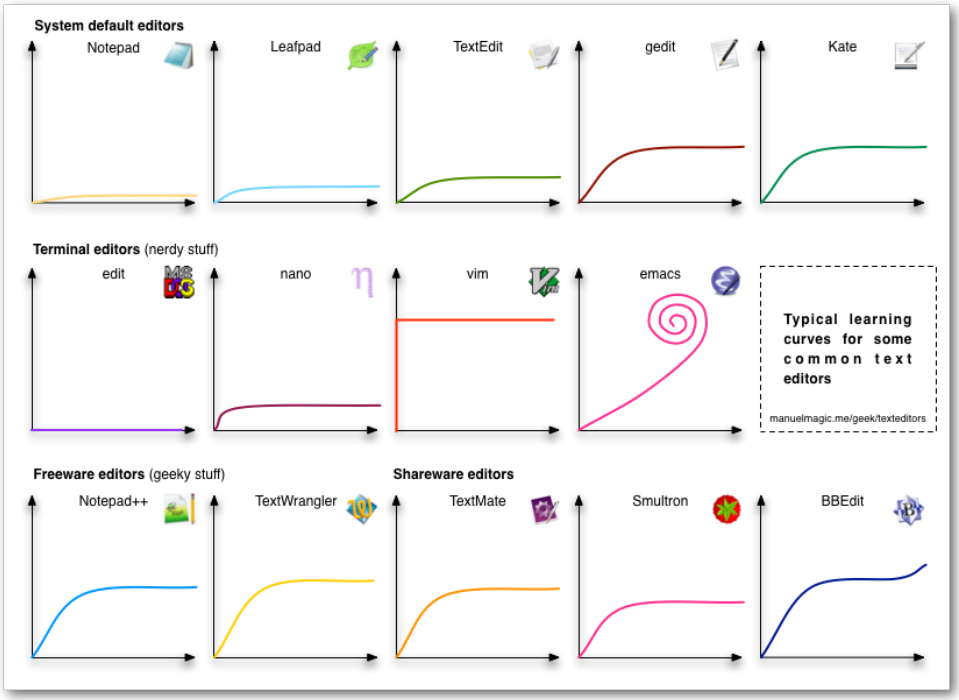


Abbildung 3.2: Typische Lernkurven der gängigen Texteditoren.

3.3 | Modales Konzept

Das modale Konzept von *Vim* charakterisiert sich dadurch, dass jeder Modus Kontextbezogen verschiedene Tastenbefehle interpretiert. Die Syntax für die Erklärung der folgenden Beispiele sieht wie folgt aus:

- *Vim*–Kommandos und Modi werden in der Regel in `monospaced`–Schriftart verfasst.
- `[parameter]`, Parameter ist optional.
- `<parameter>`, Parameter wird benötigt.
- Mit „`:`“ eingeleitete Befehle beziehen sich immer auf den `COMMAND-LINE`–Modus (siehe weiter unten)

3.3.1 NORMAL, INSERT und VISUAL Modi

Vim besitzt im Gegensatz zu den heute gängigen Programmen ein *modales* User Interface, das heißt, im Vergleich zu den heute gängigen Editoren bietet *Vim* verschiedene *Modi* um Textmanipulationen möglichst effizient durchführen zu können. Beim Start des Editor befindet sich dieser in der Regel im `NORMAL`–Modus. Die Abbildung 3.3 zeigt *gVim* (*Vim*–GTK–Version[2]). Die Statusbar unten zeigt an in welchem Modus sich der Editor befindet, des Weiteren ist der Pfad zur Datei, die Programmiersprache *Go* und das Datei–Encoding *UTF-8* gelistet. Zusätzlich sind am Ende der Statusbar noch Informationen zur aktuellen Cursorposition.

[2]Gnome Toolkit: http://www.gtk.org/

Abbildung 3.3: *gVim*–Setup mit diversen Plugins, Hybrid-Colorscheme und Syntax Highlighting der Programmiersprache *Go*.

Im NORMAL–Modus ist es nicht möglich Texte zu verfassen. Dieser Modus ist für Steuerungskommandos und zum Navigieren gedacht. Um einen Text zu verfassen muss sich der Benutzer in den INSERT–Modus begeben. Dieser Modus ist ausschließlich dazu gedacht Text einzufügen. Ein weiterer Modus ist der VISUAL–Modus welcher ebenso für erweiterte Textmanipulationen gedacht ist. Über bestimmte Tasteneingaben kann man zwischen den Modi wechseln.

Abbildung 3.4 zeigt grafisch den Wechsel zwischen den Modi. Die Tasten–Kommandos — in diesem Fall für das Navigieren zwischen den verschiedenen Modi — sind im englischsprachigen auf aktionsbezogene Verben zurückzuführen (i = insert, a = append), was das Merken dieser Befehle erleichtert.

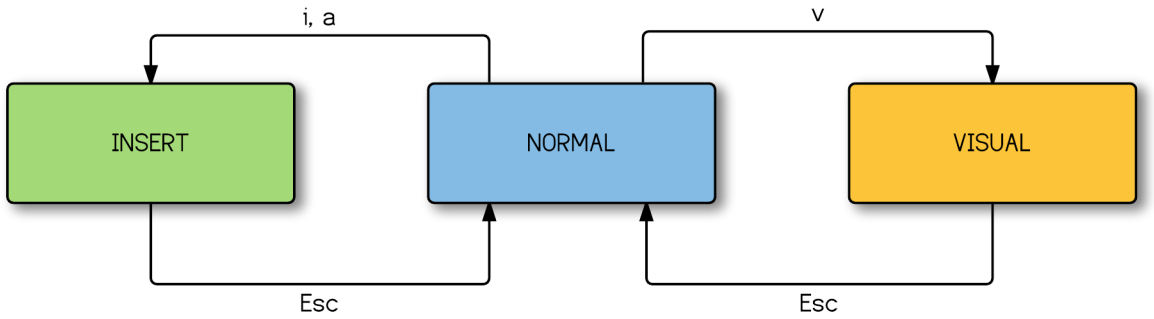


Abbildung 3.4: Die drei gängigsten Vim-Modi im Überblick.

3.3.2 COMMAND–LINE, REPLACE, VISUAL–BLOCK und VISUAL–LINE Modi

Neben den genanten Modi gibt es noch den `REPLACE`, `VISUAL-LINE`, `VISUAL-BLOCK` und `COMMAND-LINE` Modus. Der `REPLACE`–Modus dient zum direkten Ersetzen von Text, im `VISUAL--BLOCK`–Modus kann Text blockweise markiert und effizient verarbeitet werden (vgl. [6]). Im `VISUAL-LINE`–Modus findet eine zeilenweise Markierung statt.

Der `COMMAND-LINE`–Modus wird durch den „`:`“ eingeleitet. In diesem Modus können *Vim* jetzt bestimmte Befehle beispielsweise zum Suchen und Ersetzen, Laden einer Datei, Speichern einer Datei et cetera übergeben werden.

3.4 | Laden und Speichern von Dateien

Wie bereits erwähnt kann der `COMMAND-LINE`–Modus zum Laden und Speichern von Dateien verwendet werden. Beispielsweise kann die C–Quelltextdatei `/home/qitta/code/hacking.c` (absoluter Dateipfad) mit dem folgenden Befehl

```
:e /home/qitta/code/hacking.c
```

geladen werden. Anstatt `:e` kann hier auch der ausgeschriebene Befehl `:edit` verwendet werden. Tabelle 3.1 zeigt weitere grundlegende Befehle zum Laden und Speichern von Dateien.

Tabelle 3.1: Grundlegende Kommandos zum Laden und Speichern von Dateien.

Lade– und Speicher–Befehle	Erläuterung
`:w[write] [dateiname]`	Speichert Datei.
`:e[dit] [dateiname]`	Öffnet eine Datei zum Editieren.
`:r[ead] [dateiname]`	Fügt eine externe Datei an der Stelle des Cursors ein.
`:q[uit] [dateiname]`	Schließt Datei/Editor.

Befehle lassen sich auch kombinieren, der Befehl `:wq` würde eine Datei schreiben bevor der Editor geschlossen wird. Fügt man noch ein `!` mit an `:wq!` würde man das Speichern und Schließen erzwingen. Alternativ kann auch anstatt des `:wq` Befehls einfach nur der Befehl `ZZ` (zweimaliges Drücken des Großbuchstaben `Z`) im `NORMAL`–Modus abgesetzt werden. Wie für diesen Befehl gibt es auch für viele andere *Vim*–Befehle Alternativen, welche jedoch aufgrund des großen Umfang nicht weiter erläutert werden. Für Details oder Alternativen siehe *Vi and Vim Editors - Pocket Reference*, [7].

3.5 | Navigation

Unter *Vim* gibt es verschiedene Arten der Navigation. Da der Editor in erster Linie auf reine tastaturorientierte Benutzung ausgelegt ist, müssen die Navigationskonzepte ausgeklügelter sein um mindestens ebenso schnell wie mit einer Computermaus navigieren zu können.

3.5.1 h-j-k-l Navigation

Aufgrund der Tatsache, dass *Vim* für ein *ADM–3A* Terminal entwickelt wurde, erbt es die Navigation von dessen Tastaturlayout. Schaut man sich Abbildung 3.1 genauer an, dann fällt auf, dass die Pfeiltasten auf den Buchstaben `h (links)`, `j (unten)`, `k (oben)`, `l (rechts)` liegen. Der früher vorliegende Umstand dieses Layouts hat heute noch bei der Nutzung von *Vim* den Vorteil, dass sich die Hände beim Navigieren auf der sogenannten „Homerow“ befinden.

Tabelle 3.2 zeigt die Möglichkeiten der grundlegenden zeilenbasierten Navigation.

Tabelle 3.2: Zeilenbasierte Navigationsmöglichkeiten im `NORMAL`–Modus.

Navigations–Kommando	Funktion bzw. Cursor–Position
`[num]h`	Cursor geht num Zeichen nach links.
`[num]j`	Cursor geht num Zeilen nach unten.
`[num]k`	Cursor geht num Zeilen nach oben.
`[num]l`	Cursor geht num Zeichen nach rechts.
`0`	Cursor geht um Anfang der aktuellen Zeile.
`^`	Cursor geht zum ersten nicht whitespace Buchstaben der aktuellen Zeile.
`$`	Cursor geht zum letzten Buchstaben der aktuellen Zeile.
`g_`	Cursor geht zum letzten nicht whitespace Buchstaben der aktuellen Zeile.

3.5.2 Erweiterte Navigationsmöglichkeiten

Neben der *einfachen* Navigation mit den `h-j-k-l`–Tasten bietet *Vim* zur effektiven Navigation erweiterte Möglichkeiten. Tabelle 3.3 listet die erweiterten Möglichkeiten zur Navigation im `NORMAL`-Modus.

Tabelle 3.3: Erweiterte Navigationsmöglichkeiten im `NORMAL`–Modus.

Navigations–Kommando	Funktion bzw. Cursor–Position
`gg`	Geht zum Dateianfang.
`G`	Geht zum Dateiende.
`[Zeilennummer]G`	Springt in die gewünschte Zeile.
`f<buchstabe>`	Springt zum nächsten Buchstaben
`SHIFT + H`	Springt in erste Zeile des sichtbaren Bereiches.
`SHIFT + L`	Springt in letzte Zeile des sichtbaren Bereiches.

Navigations–Kommando	Funktion bzw. Cursor–Position
`SHIFT + M`	Springt in die Mitte des sichtbaren Bereiches.

3.5.3 Wort–basierte Navigation

Eine Besonderheit von *Vim* ist, dass es anhand von Textobjekten (Wörter, Sätze, Paragraphen, et cetera) navigieren kann. Befindet sich der Cursor auf einem bestimmten Wort, so kann mit folgender Syntax navigiert werden:

```
[anzahl] navigationtstaste
```

Die Möglichkeiten für das Navigieren anhand von Wörtern ergeben sich hier durch die Navigationstasten welche in Tabelle 3.4 weiter gelistet sind.

Tabelle 3.4: Erweiterte Navigationsmöglichkeiten im `NORMAL`–Modus.

Navigationstaste	Funktion bzw. Cursor–Position
`w (word)`	Cursor springt zum nächsten Wort (Anfang) weiter.
`e (end)`	Cursor springt auf den letzten Buchstaben des aktuellen Wortes.
`b (begin)`	Cursor springt zum Anfang vom Wort, wenn er bereits auf dem Anfang ist dann zum Anfang des vorangegangenen Wortes.
`[anzahl]w (word)`	Analog zu `w` Cursor springt Anzahl Wörter weiter.

3.6 | Einfügen von Text

Die in Abbildung 3.4 gelisteten Tasten sind nur ein kleiner Ausschnitt der möglichen Kommandos, um vom `NORMAL`–Modus in den `INSERT`-Modus zu kommen. Weitere Shortcuts, die den Editor in den `INSERT`–Mode versetzen finden sich der Vollständigkeit halber in Tabelle 3.5.

Tabelle 3.5: Möglichkeiten in den INSERT–Mode zu wechseln um Text einzufügen.

Keyboard-Shortcut	Funktionalität beim Einfügen
`a (append)`	Fügt Text hinter Cursor Position ein.
`i (insert)`	Fügt Text an Cursor Position ein.
`o`	Fügt Text unterhalb aktueller Zeile ein.
`A`	Füge Text am Ende der Zeile ein.
`I`	Füge Text am Anfang der Zeile ein.
`O`	Füge Text über der aktuellen Zeile ein.

Vim Navigation

Der Screencast zeigt die Navigation, den Wechsel zwischen den verschiedenen Modi sowie das Einfügen von Text unter *Vim* https://vimeo.com/148817104. Unten links in der Statusleiste ist immer der jeweilige Modus sichtbar, in welchem sich *Vim* bei einer bestimmten Operation befindet.

3.7 | Bearbeiten von Text

Das Paradigma hinter *Vim* ist mit möglichst wenigen Tastendrücken Text effizient zu bearbeiten. Wie bereits erwähnt wird im `INSERT`–Modus lediglich Text geschrieben, aber nicht manipuliert. Zwar ist es möglich in diesem Modus Text wie in einem üblichen Editor zu Manipulieren (Löschen, Kopieren, Einfügen, Markieren, et cetera), dies ist jedoch aus *Vim*–Perspektive betrachtet sehr ineffizient, da ständig zur Maus gegriffen werden muss aber auch weil der normale Editor keine erweiterten Funktionalitäten zur Textmanipulation bietet.

Im `NORMAL` Modus können *Befehle* abgesetzt werden. *Vim* hat hier ein Konzept welches wie folgt aussieht:

```
Modifikator-Taste [Anzahl] <Textobbjekt oder Bewegung>
```

Ein Beispiel hierfür wäre `d2w` [d(elete) = Modifikator, 2 = Anzahl, w(ord) = Textobjekt]. Der Befehl würde das Wort auf dem der Cursor ist und das darauf folgende Wort löschen (Anzahl = 2 Wörter). Die Tabelle 3.6 zeigt weitere mögliche Modifikatoren.

Tabelle 3.6: Manipulationstasten im `NORMAL`-Modus.

Modifikator	Erläuterung
`c (change)`	Ändert ein bestimmtes Objekt (Löscht und geht in den `INSERT`-Modus).
`d (delete)`	Löscht ein bestimmtes Objekt.
`y (yank)`	Kopiert ein bestimmtes Objekt.
`dd`	Löscht komplette Zeile in der sich Cursor befindet.
`D`	Löscht ab Cursor bis Ende der Zeile.
`cc`	Löscht komplette Zeile in der sich Cursor befindet und geht in den `INSERT`-Modus.
`C`	Löscht ab Cursor bis Ende der Zeile und geht in den `INSERT`–Modus.
`yy`	Kopiert komplette Zeile in der sich Cursor befindet.
`p`	Fügt den Inhalt des aktuellen Registers (“) unterhalb des Cursors ein.
`P`	Fügt den Inhalt des aktuellen Registers (“) oberhalb des Cursors ein.

3.8 | Weitere Textobjekte

Vim unterstützt neben dem bereits genannten Textobjekt `word` welches beispielsweise beim Löschen eines Wortes (`dw` im `NOMRAL`-Mode) verwendet wird, weitere komplexere Textobjekte. Tabelle 3.7 zeigt eine Liste der gängigsten Textobjekte.

Tabelle 3.7: Grundlegende oft genutzte Textobjekte.

Textobjekt	Beschreibung
`aw (a word)`	Ein Wort.
`as (a sentence)`	Ein Satz wird als Textobjekt selektiert.
`ap (a paragraph)`	Ein Paragraph wird als Textobjekt selektiert.
`a] (a paragraph)`	Ein Textobjekt zwischen `[]`–Block wird selektiert.
`a} (a paragraph)`	Ein Textobjekt zwischen `{}`–Block wird selektiert.
`a) (a paragraph)`	Ein Textobjekt zwischen `()`–Block wird selektiert.
`a" (a paragraph)`	Ein Textobjekt zwischen `""`–Block wird selektiert.

Die in Tabelle 3.7 genanten Textobjekte sind in Kombination mit einem Manipulationsbefehl (Modifikator-Taste) wie `d` (delete), `c` (change) oder auch `v` (visualize) zu benutzen.

3.9 | Buffers, Windows und Tabs

Unter *Vim* werden die bearbeitenden Dateien in sogenannten Buffers (dt.: Buffern) angezeigt. Diese entsprechen im weitem Sinne den normalen Tabs in üblichen Anwendungen. Ein *Window* bezeichnet unter *Vim* den sogenannten *Viewport* (dt.: Ansicht) auf eine Datei beziehungsweise einen Buffer.

Tabs repräsentieren wiederrum eine Sammlung von einem oder mehreren *Windows*. Abbildung 3.5 zeigt auf der linken Seite einem *Tab* mit mehrere *Windows* die jeweils ein *Buffer* enthalten, wechselt man mit dem Kommando `:tabn` (:tabNext) zum nächsten *Tab*, hat dieser nur ein *Window* welches auf nur einen Buffer zeigt. Der zweite Tab ist lediglich nur eine *andere Ansicht* auf die gleichen Buffer beziehungsweise in diesem Fall auf nur einen bestimmten Buffer (Datei b.txt). Die Tabelle 3.8 zeigt grundlegende Kommandos für Tabs, Buffer und Window Handling.

Tabelle 3.8: Kommandos zum Arbeiten mit Buffern und Tabs.

Befehl	Funktion
`:tabn[ext]`	Wechselt zum nächsten Tab.
`:tabp[revious]`	Wechselt zum vorherigen Tab.
`:bn[ext]`	Wechselt zum nächsten Buffer (Dateirepräsentation).
`:bp[revious]`	Wechselt zum nächsten Buffer (Dateirepräsentation).
`:buffers`	Listet die offenen (gerade nicht sichtbaren) Buffer (Dateien) mit Index.
`:<num>b`	Geht zum Buffer Nummer

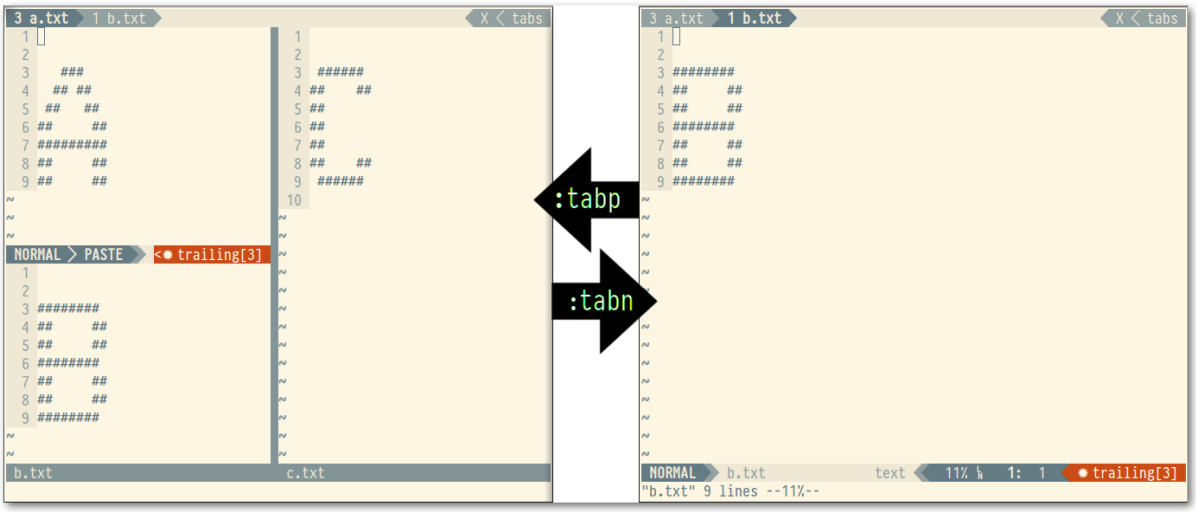


Abbildung 3.5: Links mehrere Buffer, Rechts nächster Tab mit nur einem Buffer.

Neben den genannten Befehlen in Tabelle 3.8 können auch alternative und effizientere Befehle im NORMAL–Modus abgesetzt werden. Beispielhaft wäre hier `gt` (go to next tab) für das Wechseln zum nächsten Tab zu nennen. Für weitere Befehle siehe *Vi and Vim Editors - Pocket Reference* [7] oder *Vim*–Hilfe.

3.10 | Makros

Makros erlauben es komplexe Arbeitsabläufe aufzuzeichnen und zu wiederholen. Makros können durch Drücken der Taste `q<buchstabe>` gestartet werden. Hierbei erscheint in der *Vim* Statusleiste das Wort `recording`. Der `<buchstabe>` ist dabei eine beliebige Taste auf welche das Makro aufgezeichnet werden soll. Nun können beliebige Textmanipulationen durchgeführt werden. Beendet wird die Aufzeichnung des Makros mit dem erneuten Drücken der Taste `q`.

Anschließend kann das Makro auf weiteren Text, durch Drücken von `@[<anzahl>]<buchstabe>` ausgeführt werden. Wird keine Anzahl angegeben wird das Marko nur einmal ausgeführt.

3.11 | UNDO und REDO unter *Vim*

Vim hat die Möglichkeiten der persistenten *undo*–Funktionalität[3]. Eine durchgeführte Aktion kann mit der Taste `u` für *undo* rückgängig gemacht werden. Will man eine rückgängig gemachte Aktion doch behalten, kann mit der Tastenkombination `STRG + r` im NORMAL–Modus ein *redo* durchgeführt werden. Eine weitere Möglichkeit der *REDO* und *UNDO* Funktionalität bieten die Kommandos:

```
:later[anzahl][zeiteinheit]
:earlier[anzahl][zeiteinheit]
```

[3] UNDO funktioniert auch wenn der Editor in der Zwischenzeit geschlossen wurde.

Bei der Angabe der Anzahl wird die jeweilige Anzahl von Änderungen gezählt. Eine weitere Möglichkeit ist die zusätzliche Angabe der Zeiteinheit. Mit beispielsweise `:earlier2m` bekommt man den Zustand von vor zwei Minuten.

3.12 | Plugins und externe Tools

Neben den genannten Funktionalitäten gibt es unter *Vim* unzählig viele Möglichkeiten Texte oder bestimmte Dateitypen effizient zu manipulieren. Je nach Einsatzgebiet gibt es *Plugins* die sich mehr oder weniger gut in *Vim* integrieren lassen.

Gerade unter *unixoiden* Betriebssystemen bietet sich oft die Möglichkeit Systemtools wie beispielsweise `sort` zu verwenden.

3.13 | Das `:global`–Kommando

Das Global–Kommando ist ein sehr nützliches Kommando um parallel Manipulationen an mehreren Zeilen mit einem bestimmten Schlüsselwort durchzuführen. Die Syntax sieht wie folgt aus:

```
:[bereich]g[lobal]/{muster}/{befehl}
```

Das `:global`–Kommando `:%g/hans/delete` würde alle Zeilen des aktuellen Buffers die das Muster `hans` enthalten löschen. Um das Gegenteil zu erreichen, sprich alle Zeilen die `hans` enthalten zu behalten und die restlichen Zeilen zu löschen kann `:g!` oder alternativ das `:v`–Kommando verwendet werden.

Effizientes Editieren von Text in *Vim*

Der Screencast vereint die vorangegangenen Kapitel und demonstriert das Arbeiten unter Vim. Es wird das Arbeiten mit Textobjekten sowie das erstellen von Makros gezeigt. Des Weiteren werden weitere bisher nicht genannte Features verwendet um einen beispielhaften Text in ein bestimmtes Zielformat zu überführen. https://vimeo.com/148817254.

Für weitere Informationen und *Best Practices* zum Thema *Effizientes Editieren von Text* siehe auch *Practical Vim - Edit Text at the Speed of Thought* [8].

3.14 | Softwareentwicklung unter *Vim*

Der Editor bietet von Haus aus viele Features, wie beispielsweise Autovervollständigung und Syntaxhervorhebung, die auch in modernen Entwicklungsumgebungen zu finden sind. Je nach Sprache sind bestimmte Features schon integriert oder können alternativ über *Plugins* hinzugefügt werden. Die Möglichkeiten sind hier fast unbegrenzt.

Für die Erweiterbarkeit durch *Plugins* wurde die Skriptsprache *Vim-Script* entwickelt. Man ist jedoch nicht auf *Vim-Script* beschränkt und kann je nach Vorliebe auch eine andere Skriptsprache wie beispielsweise *Python* verwenden.

Um einen kurzen Einblick zu den Möglichkeiten beim Entwickeln zu geben wurde ein Screencast mit der Programmiersprache *Go* erstellt. Die hier hauptsächlich verwendeten externen *Plugins* sind *neosnippet*[4] und *vim-go*[5].

Programmieren mit *Vim*

Der Screencast demonstriert eine Möglichkeit *Vim* als Entwicklungsumgebung zum Programmieren zu nutzen: https://vimeo.com/148817418

[4] Neosnippet, stellt Code Snippets für bestimmte Sprachen bereit: https://github.com/Shougo/neosnippet.vim
[5] Integriert das Go-Tool in *Vim*: https://github.com/fatih/vim-go

3.15 | Hilfesystem

Vim setzt als Editor ein heute eher ungewöhnliches modales Bedienkonzept um. Anfänger tun sich oft schwer das Konzept umzusetzen beziehungsweise nutzen den Editor nur in dem Umfang wie man auch einen nicht modalen Editor nutzen würde. Um dies zu vermeiden gibt mehrere Ansätze das Konzept von *Vim* korrekt zu erlernen.

3.15.1 Integrierte Dokumentation

Um eine Hilfe zu einem bestimmten Befehl zu erhalten reicht es im `COMMAND--LINE`–Modus

```
:help <schlüsselwort>
```

aufzurufen. *Vim* öffnet dann den Hilfetext beziehungsweise die Dokumentation zu dem jeweiligen Schlüsselwort (soweit vorhanden). Beispielhafter Aufruf der Hilfe mit `:help 42`. *Vim* liefert dazu als „Easter egg“ folgende Dokumentation (hier optisch abgegrenzt durch *–Rahmen):

```
************************************************************************
* What is the meaning of life, the universe and everything?  *42* *
* Douglas Adams, the only person who knew what this question      *
* really was about is now dead, unfortunately.  So now you might  *
* wonder what the meaning of death is...                          *
*                                                                 *
* =============================================================== *
*                                                                 *
* Next chapter: |usr_43.txt|  Using filetypes                     *
*                                                                 *
* Copyright: see |manual-copyright|  vim:tw=78:ts=8:ft=help:norl: *
************************************************************************
```

Grundsätzlich kann gesagt werden, dass *Vim* beziehungsweise die Funktionalitäten des Editors verglichen mit anderen Softwareprodukten sehr ausführlich dokumentiert sind.

3.15.2 Vimtutor

Für den einfacheren Einstieg bietet *Vim* den *Vimtutor* welcher durch die Eingabe jenes unter *unixoiden* Betriebssystemen zu starten ist. Der Tutor ist dazu gedacht dem Benutzer das Arbeiten mit dem modalen Konzept anhand von sieben *Lessons* beizubringen.

3.15.3 Vim Adventures

Eine weitere Möglichkeit sich an *Vim* als effizientes Textverarbeitungswerkzeug spielerisch heranzutasten ist das webbrowserbasierte Spiel *Vim Adventures*[6], siehe Abbildung 3.6.

Abbildung 3.6: Vim Advantures Browser-Spiel zum spielerischen Erlernen von *Vim*.

3.16 | Projekte mit Vim–Bedienkonzept

Das Konzept von *Vim* weist zur üblichen Arbeitsumgebung — je nach Betriebssystem — einige Unterschiede in der Usability auf. Da anscheinend für einige Personen das Konzept sehr angenehm und effizient ist, entstanden Plugins für bestehende Anwendungen, aber auch komplette Neuentwicklungen, haben das Bedienkonzept von *Vim* übernommen.

3.16.1 Plugins

Das *Vim*–Paradigma hat sich soweit verbreitet, dass es mittlerweile Plugins für verschiedene Entwicklungsumgebungen wie beispielsweise *Vrapper*[7] gibt. Aber auch außerhalb der Entwicklerwelt gibt es Bemühungen das Bedienkonzept vom *Vim* zu übernehmen. Beispiele hierfür wären die Plugins für den *Firefox Browser* (*Pentadactyl*, [9]) oder *Google–Chrome* Browser (*Vimium*, [10]).

[6] Vim Adventures http://vim-adventures.com
[7] Plugin für die Eclipse Entwicklungsumgebung um Vim-Shortcuts zu imitieren: http://vrapper.sourceforge.net/home/

3.16.2 Weitere Anwendungen

Neben Plugins wurden auch komplette Anwendungen mit dem Bedienkonzept von *Vim* entwickelt. Abbildung 3.7 zeigt einen unter *Arch Linux* verbreiteten Dateibrowser für die Kommandozeile.

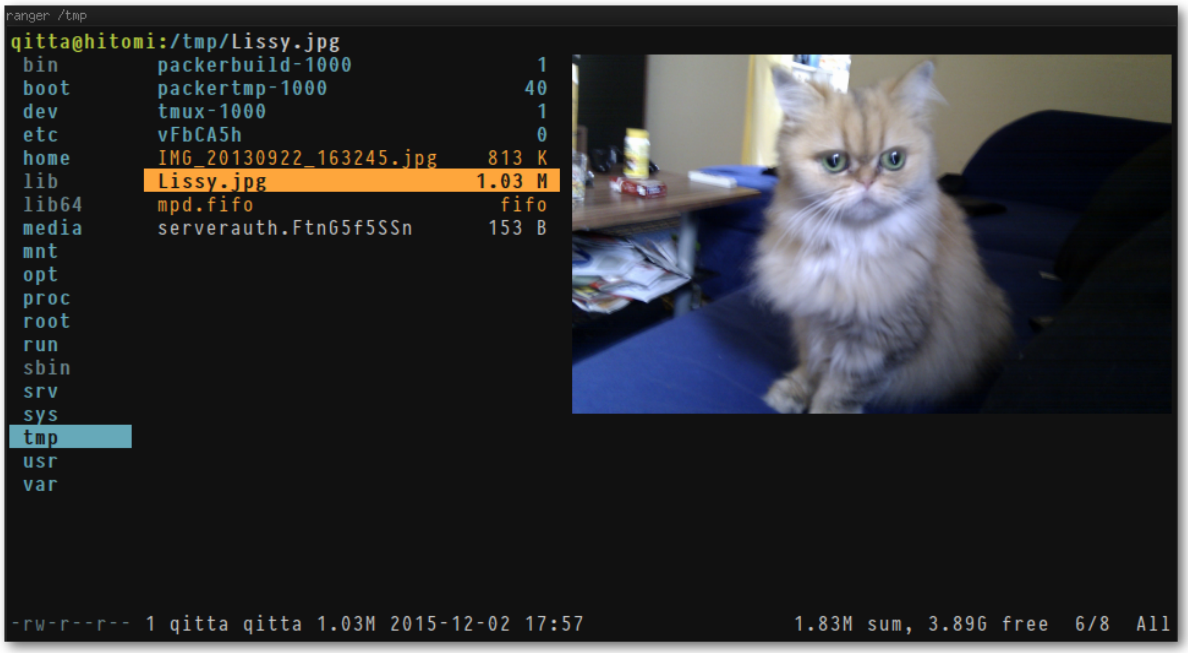


Abbildung 3.7: Ranger als kommandozeilenbasierter Dateimanager mit Drei–Spalten–Ansicht und Bild–Vorschau.

Weitere Softwareprojekte die Bedienkonzepte von *Vim* adaptieren sind beispielsweise der PDF–Viewer *Zathura* [11] oder der minimalistische Webbrowser *uzbl* [12].

3.16.3 Neovim

Aufgrund der — unter *unixoiden* Betriebssystemen — großen Verbreitung von *Vim* und der konservativen Entwicklung durch den Hauptentwickler, Bram Moolenaar, wurde das Projekt *Neovim*[8] gestartet. Das Projekt hat sich zum Ziel gemacht, den teilweise veralteten Quellcode aufzuräumen und *Vim* weiterhin zu verbessern.

[8] Neovim–Projekt: https://neovim.io/charter/

4 | Window–Manager

Ein Window–Manager (dt.: Fenstermanager) ist für den Benutzer eine nicht direkt wahrnehmbare Software welche jedoch — je nach Anwendungsgebiet — gewisse Vorteile mit sich bringen kann und somit die Produktivität beziehungsweise ein strukturierteres Arbeiten unterstützen kann.

4.1 | Was ist ein Window–Manager?

Ein Window–Manager ist die Software der grafischen Oberfläche die für das Anordnen und Dekorieren der Programm–Fenster zuständig ist. Im Grunde ist diese Software für den Benutzer nicht direkt sichtbar weil Sie als Teil des Betriebssystems aufgefasst werden kann. Abbildung 4.1 zeigt an welcher Stelle im Betriebssystem der Window–Manager einzuordnen ist.

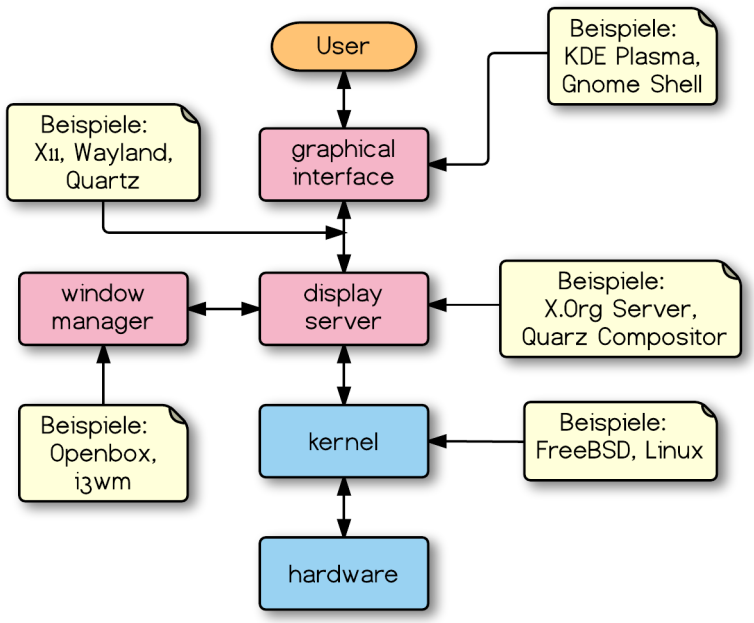


Abbildung 4.1: Window–Manager als Teil des grafischen Stacks eines Betriebssystems[1].

Unter *Windows*, *MacOS X* oder *Linux*–basierten Systemen sind sogenannte *floating–window–manager* oder auch *compositing–window–manager* üblich. Diese unterscheiden sich in der Art wie die Fenster auf den Bildschirm gebracht werden. Ältere Versionen von *Windows* und *MacOS X* haben noch *floating–window–manager* verwendetet, die aktuelleren Systeme haben mittlerweile alle einen *compositing–window–manager*.

[1] Angelehnt an: https://upload.wikimedia.org/wikipedia/commons/9/95/Schema_of_the_layers_of_the_graphical_user_interface.svg

Der Unterschied liegt darin, dass bei den reinen *floating–window–managern* kein *compositing* stattfindet. Das heißt ohne *compositing* wird jedes Fenster direkt in den Framebuffer (Bildspeicher der Grafikkarte) geschrieben, hierdurch kann es zu Artefakten (Darstellungsfehler) kommen. Bei einem Fenstermanager der *compositing* unterstützt werden die Fenster der Anwendungen zuerst in einem separatem Buffer „zusammengeführt“ und erst anschließend an den Framebuffer der Grafikkarte geschickt. Abbildung 4.2 zeigt den Unterschied anhand vom *Desktop–Window–Manager* von *Windows*.

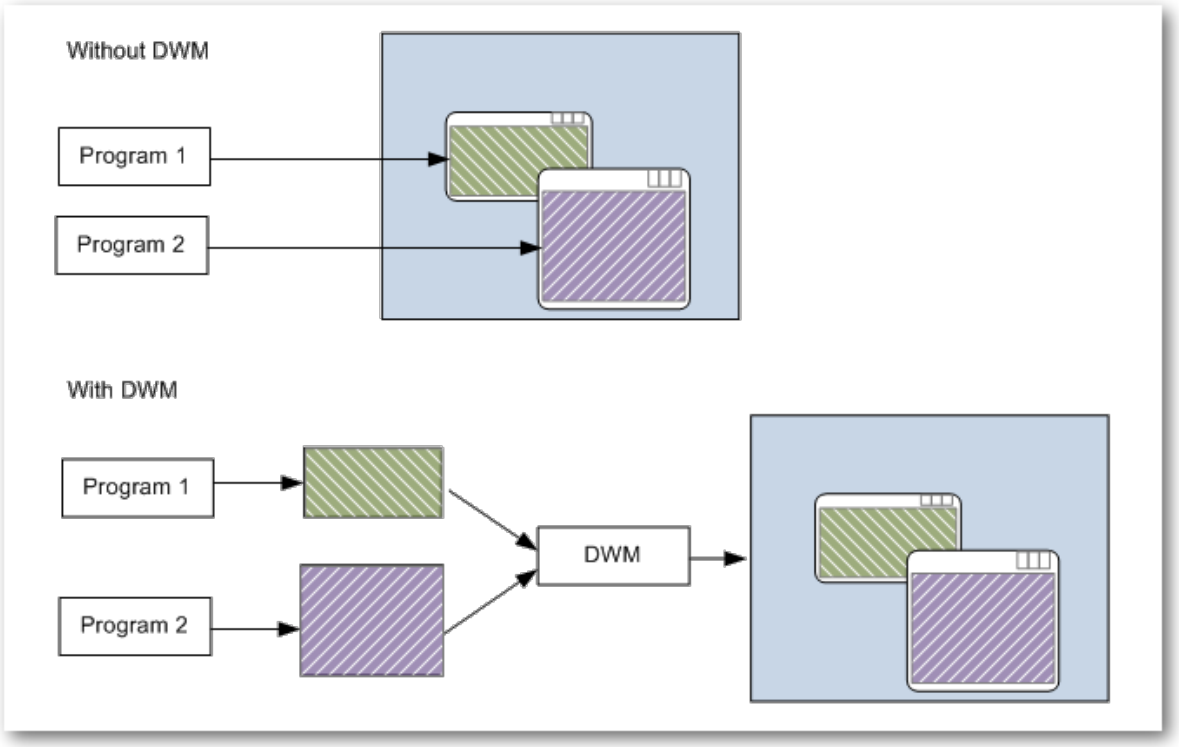


Abbildung 4.2: Rendering mit und ohne Desktop–Window–Manager[2].

Floating–window–manager ermöglichen es dem Benutzer die Fenster mit der Maus beliebig zu verschieben beziehungsweise anzuordnen. Als *window–decoration* bekommen die Fenster der Anwendungen unter den gängigen Systemen Steuerelemente zum Vergrößern, Schließen oder Minimieren von Anwendungen. Diese Fenstermanager sind in der Regel auch stark auf die Benutzung einer Maus oder eines Touchscreens ausgelegt. Des Weiteren ermöglichen *floating–window–manager* es dem Benutzer Fenster überlappend nach individuellem Geschmack anzuordnen.

Einen anderen Ansatz bieten sogenannte *tiling–window–manager*. Diese erlauben es die Fenster der Applikationen *kachelförmig* anzuordnen und vermeiden so ein Überlappen verschiedener Anwendungen. *Tiling–window–manager* haben oft die Eigenschaft, dass sie sich ausschließlich mit der Tastatur benutzen lassen. Was auf den ersten Blick kompliziert erscheinen mag, kann unter bestimmten Umständen das Arbeiten erleichtern beziehungsweise effizienter gestalten. Durch das *kachelförmige* Anordnen kann der Bildschirmplatz oft besser ausgenutzt werden. Abbildung 4.3 zeigt beispielsweise einen *dreigeteilten* Bildschirm der einen Texteditor, einen PDF Viewer und ein Konsolenfenster zeigt.

[2]Quelle: https://msdn.microsoft.com/en-us/library/windows/desktop/ff684179(v=vs.85).aspx

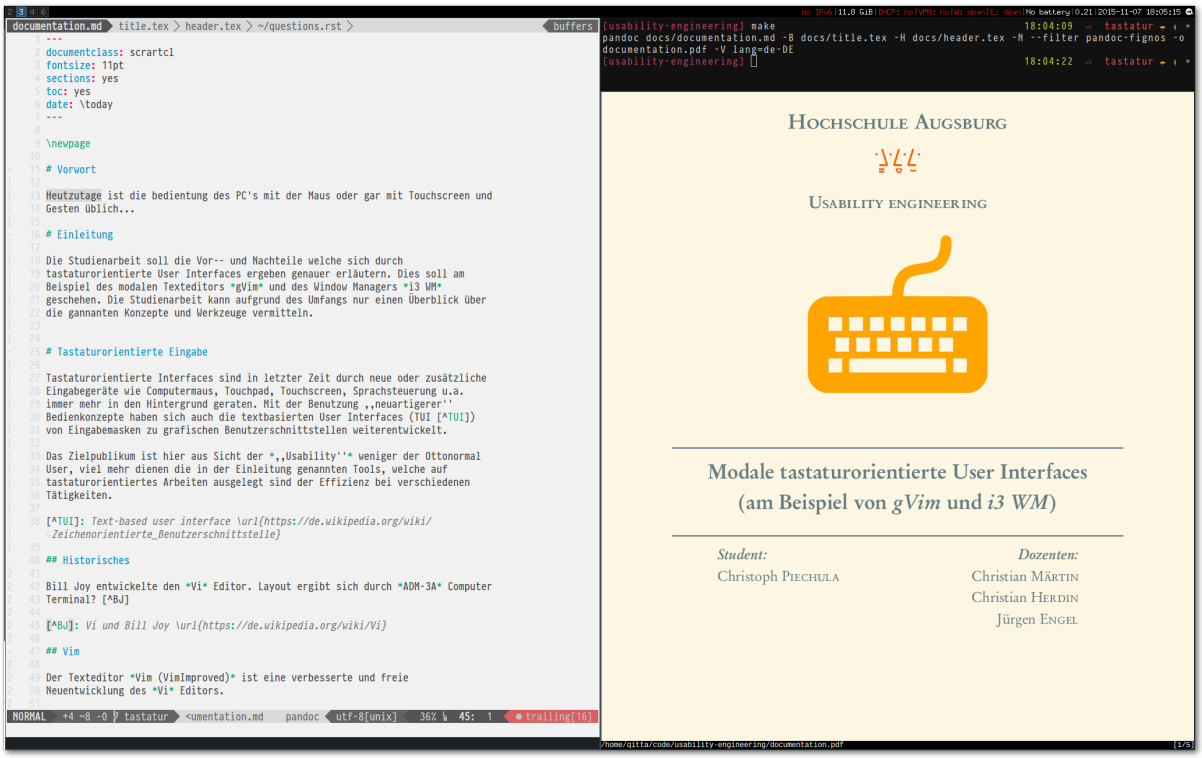


Abbildung 4.3: Ein i3-workspace mit *gVim*, urxvt–Terminal und Zathura PDF–Viewer

Es ist somit nicht nur eine optimale Aufteilung auf großen Bildschirmen gegeben, auch beim mobilen Arbeiten am Laptop kann der Einsatz einer Maus — je nach Anwendung — komplett wegfallen ohne dass die Produktivität darunter leidet.

Je nach Betriebssystem gibt es die Möglichkeit einen *tiling–window–manager* nachzuinstallieren. Tabelle 4.1 zeigt *tiling–window–manager* für die gängigen Plattformen.

Tabelle 4.1: Tiling–Window–Manager Beispiele für die gängigen Plattformen.

Plattform	Tiling–window–manager	Projekt– bzw. Quellcode–Webseite
Windows	*HashTWM*	https://github.com/ZaneA/HashTWM
	bug.n	https://github.com/fuhsjr00/bug.n
MacOS X	*Amethyst*	http://ianyh.com/amethyst/
	Slate	https://github.com/jigish/slate
Linux u. BSD (X11)	*i3 WM*	https://i3wm.org/
	Awesome WM	http://awesome.naquadah.org/

Gerade für den X11–Desktop gibt es eine große Anzahl an Fenstermanagern, insbesondere *tiling–window–managern*[3].

[3] Auswahl an *tiling–window–managern* für X: https://en.wikipedia.org/wiki/Tiling_window_manager#List_of_tiling_window_managers_for_X

4.2 | i3–Window–Manager

Der *i3*–Fenstermanager ist ein von Michael Stapelberg entwickelter *tiling–window–manager* welcher für die meisten User eine sinnvolle Konfiguration bereits *out–of–the–box* liefert. Mittels *Modifier*–Key versetzt man *i3* in einen Modus der bestimmte Kommandos vom Benutzer empfangen kann.

Beim ersten Starten des Fenstermanagers kann zwischen der `win`– oder `alt`–Taste als *Modifier*–Key gewählt werden (andere Tasten können natürlich auch über die Konfigurationsdatei definiert werden). In den folgenden Beispielen wird `alt` als *Modifier*-Key verwendet.

Über diesen *Modifier*–Key lassen sich anschließend verschiedene Befehle an den Fenstermanager absetzen. Neben dem *Modifier-Key*–Modus gibt es in der Standardkonfiguration einen weiteren Modus, welcher die `Shift`–Taste mit einbezieht, der *Modifier–Shift–Key*.

Tabelle 4.2 zeigt unter *i3* gängige Kommandos zur Steuerung. Das vollständige Standard–Keyboard–Mapping ist in Abbildung 4.4 zu sehen.

Tabelle 4.2: Grundlegende *i3–window–manager* Steuerungskommandos.

Kommando	Funktion
`alt + <num>`	Wechselt auf den virtuellen Workspace [num].
`alt + return`	Öffnet ein Konsolenfenster.
`alt + j`	Fokus auf links benachbartes Fenster wechseln.
`alt + k`	Fokus auf unten benachbartes Fenster wechseln.
`alt + l`	Fokus auf oben benachbartes Fenster wechseln.
`alt + ;`	Fokus auf rechts benachbartes Fenster zu wechseln.
	ermöglicht die Fenster mit der Maus oder Pfeiltasten in der Größe zu verändern.
`alt + d`	Startet einen Applikationslauncher, in Standardfall dmenu,
	über welchen anschließend Anwendungen gestartet werden können.

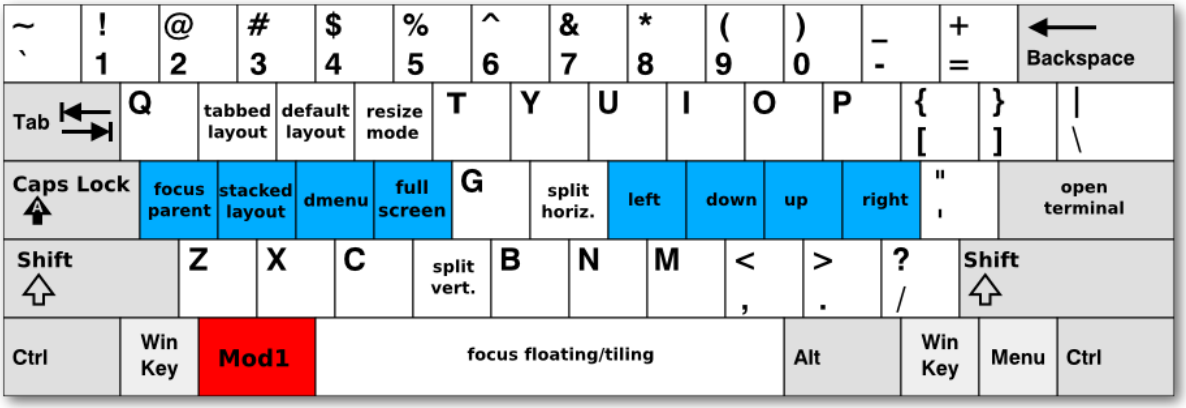


Abbildung 4.4: Standard–Keyboard–Mapping für den Modifier Modus[4].

[4]Quelle: http://i3wm.org/docs/userguide.html

Der Fenstermanager bringt eine eigene Statusleiste (*i3status*) mit. Diese zeigt, neben ein paar nützlichen Systeminformationen, an auf welchem virtuellen Workspace sich Anwendungen befinden. Verlässt man einen virtuellen Workspace und laufen auf diesem keine Anwendungen, dann wird dieser vom *i3*–Fenstermanager „zerstört“. Man sieht jeweils immer nur den aktiven Workspace oder weitere virtuelle Workspaces auf welchen Anwendungen laufen.

Abbildung 4.5 zeigt die Statusleiste von einem *i3*–System.

Abbildung 4.5: *i3*–Statusleiste mit diversen Systeminformationen. Workspace 1 ist aktiv, Workspace 2 und 3 sind sichtbar, da auf diesen Workspaces jeweils Anwendungen laufen.

Anwendungen werden beim Starten standardmäßig horizontal auf dem Bildschirm angeordnet. Das Anordnen der Anwendungen kann durch bestimmte Shortcuts manipuliert werden. Tabelle 4.3 zeigt mögliche Shortcuts um Anwendungen/Fenster in ihrer Position zu verschieben. Abbildung 4.6 zeigt das vollständige Standardmapping vom *Modifier–Shift*–Modus.

Tabelle 4.3: *i3–window–manager* Shortcuts zum verschieben von Fenstern.

Kommando	Funktion
`alt + shift + j`	Verschiebt Fenster nach links.
`alt + shift + k`	Verschiebt Fenster nach unten.
`alt + shift + l`	Verschiebt Fenster nach oben.
`alt + shift + ;`	Verschiebt Fenster nach rechts.
`alt + shift + [num]`	Verschiebt Fenster auf Workspace [num].

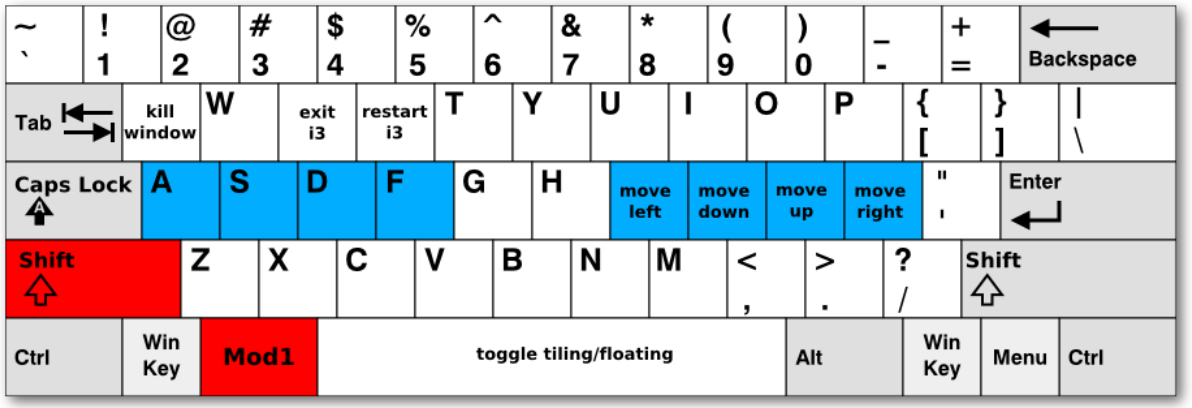


Abbildung 4.6: Standard–Keyboard–Mapping für den Modifier–Shift Modus[5].

Neben dem Verschieben der Fenster, kann auch das Anordnungsverhalten manipuliert werden. Tabelle 4.4 zeigt die Default–Shortcuts für das Modifizieren von Fenstern.

[5] Quelle: http://i3wm.org/docs/userguide.html

Tabelle 4.4: *i3–window–manager* Shortcuts zum Modifizieren von Fenstern.

Kommando	Funktion
`alt + f`	Schaltet eine Anwendung auf Vollbild.
`alt + v`	Teilt ein Fenster in vertikaler Richtung auf.
`alt + h`	Teilt ein Fenster in horizontaler Richtung auf.
`alt + r`	Resize–Modus, dieser erlaubt das Vergrößern und Verkleinern von Fenstern.

i3 wird über eine Konfigurationsdatei (`$HOME/.config/i3/config`) konfiguriert, hier können eigene Shortcuts für beispielsweise das Starten von Anwendungen oder Ausführen von Skripten definiert werden, welche dann über bestimmte Tastenkombinationen gestartet werden können.

Um beispielsweise das Starten des *Firefox*–Browsers über ein Shortcut zu definieren muss man lediglich folgenden Befehl in die Konfigurationsdatei von *i3* einfügen.

```
bindsym $mod+c exec firefox
```

Die Zeile definiert, dass durch das Drücken der *Modifier*–Taste (`$mod`) in Kombination mit dem Buchstaben `c` die Anwendung `firefox` ausgeführt (`exec firefox`) werden soll.

Des Weiteren lässt sich auch das Aussehen der Statusleiste von *i3* konfigurieren.

Neben dem *kachelförmigen* Anordnen der Fenster unterstützt *i3* auch einen sogenannten *floating–mode*, in diesem Modus werden die Fenster wie bei einem üblichen *floating–window–manager* aus dem Anordnungsgitter von *i3* herausgelöst und können Fenster–überlappend mit der Maus hin und her bewegt werden.

Arbeiten mit *i3*

Der Screencast zeigt die grundlegende Funktionsweise vom *i3–Fenstermanager* unter dem Betriebssystem *Arch Linux*: https://vimeo.com/148817504

Um weitere Eindrücke zu den Möglichkeiten beziehungsweise dem Workflow (dt.: Arbeitsfluss) von *i3* zu bekommen sind auch die Screencasts[6] des Entwicklers empfehlenswert.

Für weitere Details zu den Einstellungs– und Konfigurationsmöglichkeiten, siehe i3 Userguide[7].

[6] i3 Screencasts: http://i3wm.org/screenshots/
[7] i3 WM Userguide: https://i3wm.org/docs/userguide.html

5 | Kommandozeile als Benutzerschnittstelle

Mit dem *i3–Fenstermanager* wurde bisher tastaturorientierte Software betrachtet die hauptsächlich auf die Arbeitsweise unter einem grafischen System Einfluss hat.

Eine weitere, wenn auch in vielen Bereichen weniger verbreitete, Möglichkeit seine Arbeit zu verrichten ist die Kommandozeile. Diese wird beispielsweise unter *Mac OS X* oder anderen *unixoiden* Betriebssystemen (*Linux Distributionen*, *FreeBSD*, et cetera) verwendet um beispielsweise administrative Aufgaben zu bewerkstelligen, kann aber genau so für das Verfassen von schriftlichen Dokumenten oder zur Softwareentwicklung (vgl. [13]) verwendet werden.

5.1 | Was ist ein Terminal Multiplexer?

Beim Arbeiten auf der Kommandozeile ist man je nach System auf einem bestimmten (meist virtuellen) Terminal eingeloggt. Aufgrund des Konzeptes besteht hier in erster Linie nur die Möglichkeit eine Anwendung aktiv auf dem Bildschirm zu haben.

Da unter der Kommandozeile die Benutzung mit der Maus nicht so verbreitet ist, wird hier sehr viel mit Kommandos gearbeitet, aber auch Anwendungen wie *Vim* oder der oben genante Dateimanager *ranger* können für die Arbeiten unter der Kommandozeile verwendet werden.

Um die anstehenden Aufgaben nicht nacheinander ausführen zu müssen, gibt es sogenannte Terminal Multiplexer welche es erlauben mehrere virtuelle Terminals parallel anzeigen zu lassen.

5.2 | Tmux

Tmux ist ein Terminal Multiplexer welcher unter *MacOS X* oder *Linux* verfügbar ist. Der Terminal Multiplexer arbeitet wie auch der *i3–Fenstermanager* mit einem modalen User Interface, dieses ist in diesem Fall aber nicht von einer grafischen Oberfläche abhängig.

5.3 | Tmux–Sitzungen

Eine *Tmux*–Sitzung kann mit dem Befehl `tmux new -s mein_sitzungsname` gestartet werden, bereits offene Sitzungen können mit dem Befehl `tmux list-sessions` angezeigt werden. Für weitere Kommandozeilenparameter siehe Manpage (Handbuch unter Linux: `man tmux`). Um sich zu einer bereits offenen Sitzung zu verbinden kann man `tmux a -t mein_sitzungsname` verwenden. Abbildung 5.1 zeigt eine *Tmux*–Sitzung.

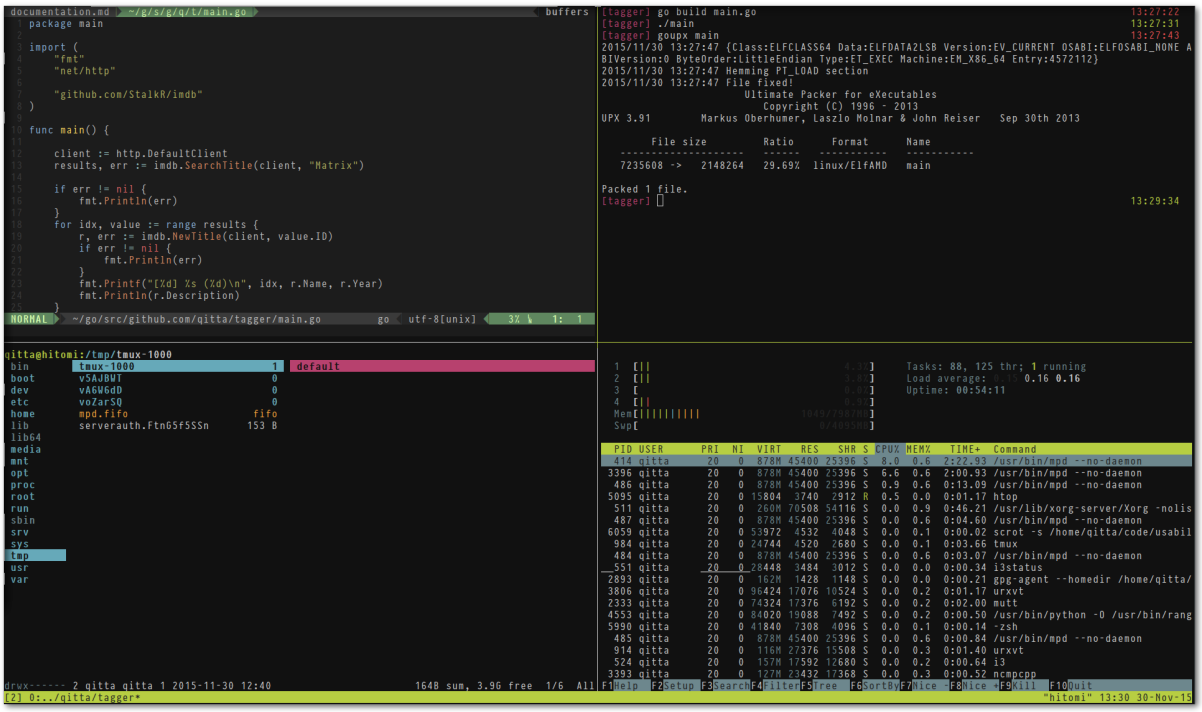


Abbildung 5.1: Tmux mit vier parallel offenen Terminals.

Tmux hat am unteren Ende des Terminals eine Statusleiste, welche standardmäßig Informationen zum Hostnamen sowie den geöffneten Fenstern (*Windows*) zeigt. Abbildung 5.2 zeigt die Statusbar der Sitzung „mailserv“ mit zwei offenen Fenstern (`bash` und `ranger`).

Abbildung 5.2: Tmux Statusleiste mit Sessionname „mailsrv“, im Window `0` läuft die Bash, im Window 1 läuft der ranger–Dateimanager. Auf der rechten Seite ist der Hostname mit Timestamp.

5.4 | Tmux Sessions, Panes und Windows

Eine *Tmux*–Sitzung zeigt nach dem Starten ein *Window*. In diesem kann jede beliebige Konsolenanwendung laufen. Das *Window* kann in sogenannte *Panes* unterteilt werden, welche ebenso eine separate Konsolenanwendung verarbeiten können. Abbildung 5.3 zeigt den Zusammenhang zwischen einer *Tmux*–Sitzung, einem *Window* und *Panes*.

Tmux nutzt, wie auch *i3*, ein tastaturorientiertes User Interface. Der Standard *Modifier*–Key — auch *Prefix* bei *Tmux* genannt — für *Tmux* ist `STRG + b`. Tabelle 5.1 zeigt die standardmäßig definierten Shortcuts für das Arbeiten mit *Windows*, Tabelle 5.2 die standardmäßig definierten Shortcuts für das Arbeiten mit *Panes*.

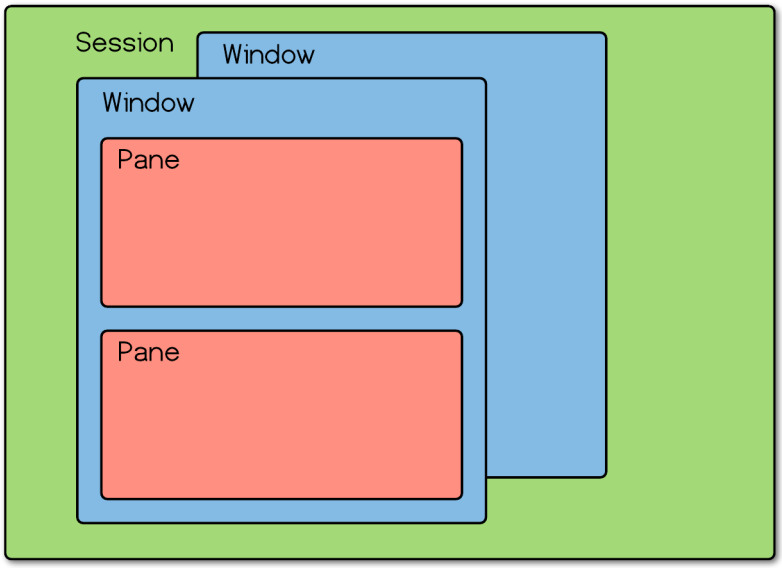


Abbildung 5.3: Zusammenhang zwischen Sessions, Windows und Panes.

Tabelle 5.1: Keybindings für *Window*–Handling unter *Tmux*.

Shortcut	Funktion
`STRG + b + c`	Erstellt ein neues *Window*.
`STRG + b + w`	Listet alle bekannten *Windows* auf.
`STRG + b + n`	Springt zum nächsten *Window*.
`STRG + b + p`	Springt zum vorherigen *Window*.
`STRG + b + f`	Suchfunktion ein *Window* zu suchen.
`STRG + b + ,`	Umbennenen eines *Window*.
`STRG + b + &`	Zerstören eines *Window*.
`STRG + b + %`	*Window* vertikal aufteilen in zwei *Panes*
`STRG + b + "`	*Window* horizontal aufteilen in zwei *Panes*

Tabelle 5.2: Keybindings für *Pane*–Handling unter *Tmux*.

Shortcut	Funktion
`STRG + b + o`	Vertauschen von *Panes*.
`STRG + b + q`	Zeigt *Pane*–Nummer an.
`STRG + b + x`	Schließt ein *Pane*.
`STRG + b + SPACE`	Wechselt zwischen verschiedenen Layouts.

Terminal Multiplexer haben den Vorteil, dass der Benutzer sich von einer Sitzung trennen kann und diese zu einem späteren Zeitpunkt wieder aufnehmen kann. Alle in der Sitzung laufenden Prozesse und Anwendungen laufen unverändert weiter auch wenn der Benutzer die Sitzung verlässt.

Dieser Umstand erleichtert beispielsweise das Arbeiten auf entfernten Systemen, auf welchen man sich für bestimmte Tätigkeiten einloggen muss um beispielsweise einen Server mit Updates zu versorgen. Der Systemadministrator kann so den Update–Prozess in einer *Tmux*-Sitzung anstoßen und sich in der Zwischenzeit um andere Aufgaben kümmern. Loggt er sich zu einem späteren Zeitpunkt wieder auf dem Server beziehungsweise der Sitzung ein so kann er über *Tmux* den Ablauf des Update–Prozesses sichten.

Arbeiten mit Tmux

Der Screencast zeigt die grundlegende Arbeitsweise und Möglichkeiten die *Tmux* als Terminal–Multiplexer bietet: https://vimeo.com/148817733

5.5 | Erweiterung Byobu

Byobu ist eine Erweiterung für Terminal Multiplexer wie *Screen* oder *Tmux*. Es ist im Grunde ein „Wrapper“ um einen Terminmal Multiplexer und ermöglicht weitere Konfigurationsmöglichkeiten sowie erweiterte Features wie beispielsweise *Status Notifications*. Für weitere Möglichkeiten und Features siehe *Byobu* Dokumentation[1].

[1] http://byobu.co/documentation.html.

6 | Fazit

Die Umstellung auf ein hauptsächlich tastaturorientiertes User Interface ist sicherlich nicht für jeden Benutzer eine gute Wahl. Benutzer die viel mit grafischen Bildmanipulationswerkzeugen arbeiten sind wahrscheinlich bei einem üblichen *floating–window–manager* besser aufgehoben.

Jedoch kann auch bei solchen Anwendungen die Arbeit effizient mit Hilfe von Shortcuts gesteigert werden. Letztendlich sollte jeder Benutzer selbst entscheiden dürfen mit welchen Werkzeugen er seine Arbeit verrichten mag.

Die Qualität eines Umstiegs auf einen Editor wie *Vim* oder einen *tiling–window–manager* wie *i3* ist stark vom persönlichem Arbeitsfluss und Vorlieben abhängig. Wenn man überwiegend mit Textmanipulation (Paper/LaTeX, E–Mail, Programmieren, …) zu tun hat oder allgemein viel über die Kommandozeile arbeitet kann man hier durchaus einen stärkeren Nutzen herausschlagen.

Für Systemadministratoren aber auch Entwickler die viel unter *unixoiden* Betriebssystemen Arbeiten, könnte sich ein Terminal Multiplexer wie *Tmux* positiv auf die Produktivität beziehungsweise den Arbeitsfluss auswirken.

Literaturverzeichnis

[1] J.-W. Chen and J. Zhang, "Comparing text-based and graphic user interfaces for novice and expert users," in *AMIA annual symposium proceedings*, 2007, vol. 2007.

[2] Wikipedia, "User interface — wikipedia, the free encyclopedia," 2015. [Online]. Available: https://en.wikipedia.org/w/index.php?title=User_interface&oldid=690789215.

[3] A. Robbins, *Learning the vi and vim editors*. Sebastopol, CA: O'Reilly Media, 2008.

[4] K. Schulz, *Hacking vim 7.2 ready-to-use hacks with solutions for common situations encountered by users of the vim editor*. Birmingham, UK: Packt Pub, 2010.

[5] C. Newham and B. Rosenblatt, *Learning the bash shell: Unix shell programming*. " O'Reilly Media, Inc.", 2005.

[6] M. McDonnell, *Pro vim*. Berkeley: Apress, 2014.

[7] A. Robbins, *Vi and vim editors pocket reference*. Sebastopol, CA: O'Reilly Media, 2011.

[8] D. Neil, *Practical vim : Edit text at the speed of thought*. Dallas, Texas: The Pragmatic Bookshelf, 2012.

[9] Wikipedia, "Pentadactyl — wikipedia, the free encyclopedia," 2014. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pentadactyl&oldid=600612554.

[10] Wikipedia, "Vimium — wikipedia, the free encyclopedia," 2015. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Vimium&oldid=691574027.

[11] Wikipedia, "Zathura (document viewer) — wikipedia, the free encyclopedia," 2015. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Zathura_(document_viewer)&oldid=680804845.

[12] Wikipedia, "Uzbl — wikipedia, the free encyclopedia," 2015. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Uzbl&oldid=653545729.

[13] B. Hogan, *Tmux : Productive mouse-free development*. Dallas, Tex: Pragmatic Bookshelf, 2012.