

Markdown & Pandoc

Michael Kofler

Syntax ■ Optionen, Templates
HTML ■ LaTeX ■ EPUB ■ Mobi
PDF ■ Folien (Beamer)
Tools und Techniken

2. Auflage

ebooks.kofler

Impressum

Markdown & Pandoc, 2. Auflage

Markdown-Syntax. Werkzeuge. LaTeX und HTML. E-Books erstellen. Präsentationen.

© Michael Kofler und ebooks.kofler 2018

Autoren	Michael Kofler (mit einem Beitrag von Axel Dürkop und Tina Ladwig)
Korrektorat	Markus Hinterreither
ISBN PDF	978-3-902643-33-9
ISBN EPUB/Mobi	978-3-902643-34-6
Verlag	ebooks.kofler , Schönbrunnngasse 54c, 8010 Graz, Österreich

Die PDF- und EPUB-Ausgabe dieses Buchs können Sie hier kaufen:

https://kofler.info/ebooks/markdown_pandoc

Inhaltsverzeichnis

Impressum	2
Vorwort	12
1 Einführung	15
1.1 Hello World!	15
1.2 Hintergrund, Geschichte	19
Das originale Markdown von John Gruber	19
Markdown-Varianten und -Implementierungen	19
Markdown-Alternativen	21
2 Werkzeuge	22
2.1 Pandoc installieren	22
Pandoc unter Windows installieren	22
Pandoc unter Linux installieren	23
Pandoc unter macOS installieren	24
Pandoc verwenden	25
Original-Markdown installieren	25
2.2 Markdown-Editoren	26
Markdown im Webbrowser	26
Editoren für Einsteiger ...	27
... und für Profis	29

3 Die Markdown-Syntax	31
3.1 Absätze	32
Textausrichtung ändern	33
Zeilenumbruch	33
Linienblöcke (Pandoc)	34
3.2 Zeichensätze und Sonderzeichen	34
Sonderzeichen	35
Programmcode	37
Feste Leerzeichen (Pandoc)	37
Gedankenstriche (Pandoc)	38
Weiche Trennzeichen (Pandoc)	38
Anführungszeichen	38
3.3 Kommentare	39
3.4 Dokumentstruktur und Überschriften	40
Zusatzinformationen in Überschriften (Pandoc)	41
3.5 Textformatierung (fett, kursiv ...)	42
Fetter und kursiver Text	42
Pandoc-Eigenheiten	43
Hochstellen, Tiefstellen, Durchstreichen und Kapitälchen (Pandoc)	43
3.6 Aufzählungen und Listen	44
Verschachtelte Listen	46
Nummerierte Listen	47
Beispiellisten (Pandoc)	48
Definitionslisten (Pandoc)	49
Todo-Listen (GFM)	50

3.7	Einrückungen bzw. Zitate	50
3.8	Listings und Programmcode	51
	Kurze Passagen	51
	Mehrzeilige Listings	52
	Listings ohne Einrückungen (Pandoc, GFM)	53
	Syntax-Hervorhebung (Pandoc, GFM)	54
3.9	Links und Querverweise	55
	Einfache Links	55
	Inline-Links	56
	Referenzen	56
	Querverweise innerhalb von Markdown-Dokumenten (Pandoc)	58
3.10	Bilder	59
	Beschriftung von Bildern (Pandoc)	59
	Bildgröße und andere Parameter (Pandoc)	60
3.11	Tabellen	61
	Linientabellen (PHP Markdown Extra, Pandoc, GFM)	62
	Einfache Tabellen (Pandoc)	63
	Tabellen mit mehrzeiligen Einträgen (Pandoc)	64
	Gittertabellen (Pandoc)	65
	Horizontale Linien	66
3.12	Fußnoten	67
	Eingebettete Fußnoten (Pandoc)	67
	Referenzierte Fußnoten (Pandoc)	67
3.13	HTML- und LaTeX-Code	68
	HTML-Code	68

Markdown-Formatierung innerhalb des HTML-Codes (Pandoc)	70
LaTeX-Formeln (Pandoc)	70
Formeln LaTeX-Makros (Pandoc)	72
LaTeX-Code (Pandoc)	72
HTML- und LaTeX-Code kombinieren (Pandoc)	73
3.14 Weitere Pandoc-spezifische Funktionen	74
Literaturangaben und Literaturverzeichnisse	74
Dokument-Metadaten	75
Dokument-Metadaten im YAML-Format	77
Import anderer Dateien	78
4 Das Pandoc-Kommando	79
4.1 Input- und Output-Formate	80
Einschränkungen	83
Markdown-Kompatibilität	84
4.2 Pandoc-Erweiterungen	85
4.3 Pandoc-Optionen	86
Allgemeine Optionen	87
Allgemeine Reader- und Writer-Optionen	88
Writer-Optionen für HTML	90
Writer-Optionen für LaTeX und PDF	91
Writer-Optionen für Folien	91
Writer-Optionen für E-Books im EPUB-Format	92
4.4 Templates	92
Template-Syntax	93
Template-Beispiel	95

Eigene Templates	96
4.5 Filter	97
AST-Aufbau	98
Traditionelle JSON-Filter	99
Lua-Filter (ab Pandoc 2)	101
Voraussetzungen und Einschränkungen	101
Fertige Filter	102
Beispiel 1: Code-Dateien einfügen	102
Beispiel 2: Abkürzungen typografisch korrekt darstellen	103
5 HTML-Dokumente	105
5.1 Der Aufbau von HTML-Dokumenten	105
Eigene CSS- und JavaScript-Dateien	107
Sonderzeichen und äöüß werden falsch dargestellt	107
5.2 Die Formatierung von HTML-Dokumenten	108
Überschriften und Inhaltsverzeichnis	110
Nummerierte Überschriften	111
Tabellen	113
Abbildungen	114
6 E-Books im EPUB- und Mobi-Format	116
6.1 EPUB	117
EPUB-Reader und -Viewer	117
EPUB-Dateien mit Pandoc erzeugen	118
Metadaten und Cover	118
Blocksatz und Formatierung	119

Bilder und Listings	120
Inhaltsverzeichnis	121
Mathematische Formeln und HTML-Code	122
Eigene Schriften	122
EPUB-Dateien validieren	125
EPUB-Interna	126
EPUB-Dateien aus HTML-Dokumenten erzeugen	128
6.2 Mobi	128
KindleGen	129
EPUB zu Mobi konvertieren	130
HTML zu Mobi konvertieren	130
Vorschau	134
Mobi-Interna ansehen	135
7 LaTeX und PDF	137
7.1 LaTeX- und PDF-Export	138
PDF-Engines	139
LaTeX-spezifische Pandoc-Optionen	140
Defaultformatierung	141
LaTeX-Template und Template-Variablen	142
7.2 Individuelle Formatierung	146
Seitenformat und Seitenlayout	147
Kopf- und Fußzeilen	148
Überschriften	148
Sprache	150

Silbentrennung	150
Fonts und Schriftgröße	151
Fließtext	152
Listen und Aufzählungen	153
Tabellen	154
Abbildungen	159
Einrückungen bzw. Zitate	160
Listings mit dem fancyvrb-Paket	162
Listings mit dem listings-Paket	164
Beispieldateien	167
8 Folien	168
8.1 Grundlagen	168
PDF oder HTML?	168
Strukturierung von Dokumenten	169
Titelseite	171
8.2 HTML-Folien	171
Standalone-Folien	173
8.3 PDF-Folien (Beamer)	173
Strukturierung	175
LaTeX-Code	177
Zweispaltige Folien	177
Abbildungen	178
Handouts mit pgfpages	178
Handouts mit pdfpages	179

9 Fortgeschrittene Arbeitstechniken	180
9.1 Atom und Pandoc kombinieren	181
Absatz neu umbrechen	181
Atom-Paketverwaltung	182
Markdown-Unterstützung ohne Konfiguration	183
Markdown-Preview-Enhanced und Pandoc	184
9.2 Der Satz dieses E-Books	185
PDF-Version	185
Pandoc- und LaTeX-Aufruf trennen	185
Preprocessing	187
Postprocessing	188
LaTeX-Dateien	190
LaTeX-Code in den Markdown-Dateien	190
Automatischer Aufruf der Scripts	191
Der Satz von »echten« Büchern	191
9.3 Docker-Container für Pandoc	192
Dockerfile	193
Image erzeugen	194
Container erzeugen und starten	195
9.4 Kollaborativ schreiben mit GitLab	196
Das große Ganze	197
Bereitstellen des Docker-Images auf Docker Hub	198
Bei gitlab.com anmelden	199
GitLab-Repository einrichten	200
Konfiguration des GitLab-Runners	200

Jobs und Pipelines	201
Artefakte finden und herunterladen	203
Kollaborativ schreiben	203
Abschließende Betrachtung	205

Vorwort

Als ich vor mehr als fünf Jahren die erste Auflage dieses E-Books verfasst habe, war die Markdown-Syntax für viele Anwender noch ein wenig exotisch. Das hat sich mittlerweile grundlegend geändert: Blog-Autoren und Software-Entwickler verfassen täglich Webseiten, Bug-Reports und Dokumentationstexte in der Markdown-Syntax. Selbst »gewöhnliche« Anwender stoßen im Internet immer häufiger auf Formulare, die die Markdown-Syntax (zumindest teilweise) erlauben.

Markdown ...

Die Grundidee von Markdown besteht darin, Texte ausschließlich durch Einrückungen bzw. durch »gewöhnliche« Zeichen so zu formatieren. Damit bleibt der Text für das menschliche Auge gut lesbar, gleichzeitig ist die Syntax für den Computer (also für Markdown-Programme wie Pandoc) eindeutig. Ein Beispiel macht das sofort klar:

```
Das ist ein Markdown-Text mit kursiv und **fett
hervorgehobenen** Wörtern. Hier beginnt eine Aufzählung:
```

- * Der erste Punkt der Aufzählung.
- * Und der zweite Punkt.

Im Vergleich zu herkömmlichen Textverarbeitungsprogrammen wie Word hat das Arbeiten mit Markdown-Texten den Vorteil, dass jeder beliebige Texteditor geeignet ist. Da es sich bei Markdown-Dateien um reine Textdateien handelt, können diese unter Linux oder macOS mit Kommandos wie `grep`, `sed`, `wc` verarbeitet und mit Versionsverwaltungssystemen wie SVN oder Git verwaltet werden. Im Klartext bedeutet das: schnelles, effizientes, plattformunabhängiges Arbeiten auch im Team, und das ganz ohne Abstürze.

Markdown bietet sich aber auch für all jene an, die im naturwissenschaftlichen Umfeld Texte verfassen müssen, vom kurzen Paper bis zur Dissertation. In diesem Segment galt (und gilt zum Teil noch immer) LaTeX als das Maß aller Dinge. Allerdings ist die Syntax

von LaTeX *viel* komplizierter und unübersichtlicher als die Markdown-Syntax. Außerdem brilliert LaTeX zwar bei der Erzeugung formvollendeter PDF-Dokumente, scheitert aber bei der Umwandlung der Texte in die Formate HTML, EPUB oder Mobi. LaTeX ist also zum zeitgemäßen Publizieren nur mit großen Einschränkungen geeignet.

... und Pandoc

An dieser Stelle kommt *Pandoc* ins Spiel: Die Hauptfunktion dieses Programms besteht darin, Markdown-Dokumente in das HTML-Format umzuwandeln. Pandoc sieht sich aber weniger als Markdown-Programm denn vielmehr als universellen Konverter zwischen allen erdenklichen Textformaten (und zwar in beide Richtungen!). Zu den wichtigen von Pandoc unterstützten Formaten zählen neben Markdown und HTML auch LaTeX, EPUB und Word. Die Dokumentation zählt fast 50 verschiedene Formate auf.

Sie können Pandoc also beispielsweise dazu verwenden, um einen mit Word verfassten Text in das Markdown-Format umzuwandeln und daraus dann eine PDF-Datei und ein E-Book im EPUB-Format zu erstellen. In der Praxis sind derartige Umwandlungen zwischen verschiedenen Formaten allerdings mit vielen Einschränkungen verbunden. Dieses E-Book konzentriert sich deswegen auf die Markdown-Funktionen von Pandoc.

Pandoc unterstützt eine erweiterte Markdown-Syntax, die auch das Hoch- und Tiefstellen von Text, die Gestaltung von Tabellen und Fußnoten sowie das Bilden von Querverweisen erlaubt. Pandoc macht Markdown zu einem universellen Ausgangsformat für nahezu jede Art von Dokument, von Blog-Beiträgen bis zur Diplomarbeit, von Vortragsfolien bis zum kompletten Buch.

Dieses E-Book

Dieses E-Book gibt eine Einführung in die großartige Markdown-Welt und ihre vielfältigen Anwendungsmöglichkeiten. Nach einem kurzen Einführungskapitel werden die folgenden Themen im Detail behandelt:

- Markdown-Tools (Installation, Markdown-Editoren)
- Markdown-Syntax inklusive der Pandoc-spezifischen Erweiterungen
- Pandoc-Optionen und -Anwendungskonzepte

- HTML-Dokumente erstellen
- E-Books (EPUB und Mobi)
- LaTeX-Export und PDF-Erzeugung
- Vortragsfolien (Slides)
- Docker-Setup und andere fortgeschrittene Arbeitstechniken

Natürlich freue ich mich über jeden Leser, jede Leserin; wenn es Ihnen aber nur darum geht, für Ihr Content Management System (CMS) hin und wieder einen Blog-Beitrag in Markdown-Syntax zu verfassen, dann reicht es vermutlich aus, wenn Sie sich schnell die Markdown-Syntaxzusammenfassung in der [Wikipedia](#) durchlesen.

Die eigentliche Zielgruppe dieses E-Books sind fortgeschrittene Benutzer, die längere Texte (Diplomarbeiten, Bücher etc.) schreiben und publizieren möchten, die also z. B. PDFs und EPUBs erzeugen wollen. Bei aller Begeisterung für Markdown und Pandoc muss ich zugeben, dass die Realisierung eigener Layout-Wünsche jedoch mit viel Mühe und Handarbeit verbunden ist. Oftmals sind eigene Header-Dateien (LaTeX) oder gar Scripts erforderlich, die den von Pandoc generierten Code nachbearbeiten. Benutzer mit Programmier- und LaTeX-Erfahrung sind hier klar im Vorteil. Insofern eignet sich die Kombination aus Markdown und Pandoc gut für *technical writing*, wo dieser Hintergrund zumeist gegeben ist.

Für die hier vorliegende zweite Auflage habe ich das E-Book umfassend überarbeitet und im Hinblick auf die Pandoc-Version 2.3 aktualisiert. Naturgemäß ist dieser Text – ebenso wie die meisten meiner im Eigenverlag bzw. im Rheinwerk Verlag erschienen E-Books und Bücher – selbst als Markdown-Dokument entstanden. Die Textdateien wurden mit Pandoc (für die Mobi- und EPUB-Ausgabe) und außerdem mit LaTeX (für die PDF-Ausgabe) verarbeitet.

Wenn Sie selbst HTML-Seiten, Artikel, Folien, Dokumentationen, Bücher oder E-Books effizienter als bisher verfassen möchten – lesen Sie weiter! Lernen Sie die wunderbare Welt von Markdown und Pandoc kennen! Viel Erfolg beim Publizieren mit Markdown und Pandoc wünscht Ihnen

Michael Kofler im Oktober 2018

<https://kofler.info>

1 Einführung

1.1 Hello World!

Die folgenden Zeilen zeigen einen einfachen Markdown-Text. Die Formatierung des Texts ist auch im Textmodus offensichtlich. Zur Texteingabe ist jeder beliebige Texteditor geeignet. Besonders komfortabel ist das Verfassen von Markdown-Dokumenten, wenn der Editor die Markdown-Syntax kennt und Textelemente farblich hervorhebt bzw. eventuell gleich eine Vorschau des formatierten Texts ermöglicht.

```
Die Markdown-Syntax
```

```
=====
```

```
Markdown-Dokumente sind simple Textdateien. Die  
Textformatierung erfolgt durch Auszeichnungselemente,  
die den Textfluss kaum stören. Ein paar Beispiele:  
*Kursiver Text*, **fetter Text**, `Text mit Sonderzeichen  
*in* Listing-Schrift`.
```

```
Einfache Links werden zwischen ``<` und `>` gestellt:  
<https://de.wikipedia.org/wiki/Markdown>.
```

```
Alternativ kann auch zwischen dem dargestellten Text und  
dem Link differenziert werden:  
[Wikipedia](https://de.wikipedia.org/).
```

```
> Eingerückter Text wird wie in E-Mails mit dem Zeichen `>`  
eingeleitet. Die Einrückung kann natürlich über mehrere
```

```
Zeilen reichen.
```

```
Aufzählungen
```

```
-----
```

```
Nicht nummerierte Aufzählungspunkte werden durch Sterne  
markiert:
```

- ```
* Der erste Aufzählungspunkt
```
- ```
* Der zweite Aufzählungspunkt
```
- ```
* Der dritte Aufzählungspunkt
```

```
Listings/Programmcode
```

```

```

```
 Programmlistings müssen im Markdown-Quelltext
 um vier Zeichen eingerückt werden. Hier können
 fast alle Sonderzeichen verwendet werden. <>(){}*&"'
```

Üblicherweise werden Markdown-Dokumente in Dateien mit der Kennung \*.md gespeichert. Um den Text nun in das HTML-Format umzuwandeln, benötigen Sie einen Markdown-Konverter, z. B. das Perl-Script `markdown` oder das Kommando `pandoc`. Die Installation und Bedienung dieser Werkzeuge ist Thema des nächsten Kapitels. Die resultierende HTML-Datei können Sie nun mit einem beliebigen Webbrowser ansehen (siehe die folgende Abbildung)

Eine individuelle und ansprechendere Darstellung des Texts erreichen Sie, indem Sie in die HTML-Datei eine CSS-Datei (Cascading Stylesheets) einbinden (siehe das Kapitel [HTML-Dokumente](#)).





Abbildung 1.1: HTML-Darstellung des Hello-World-Textes

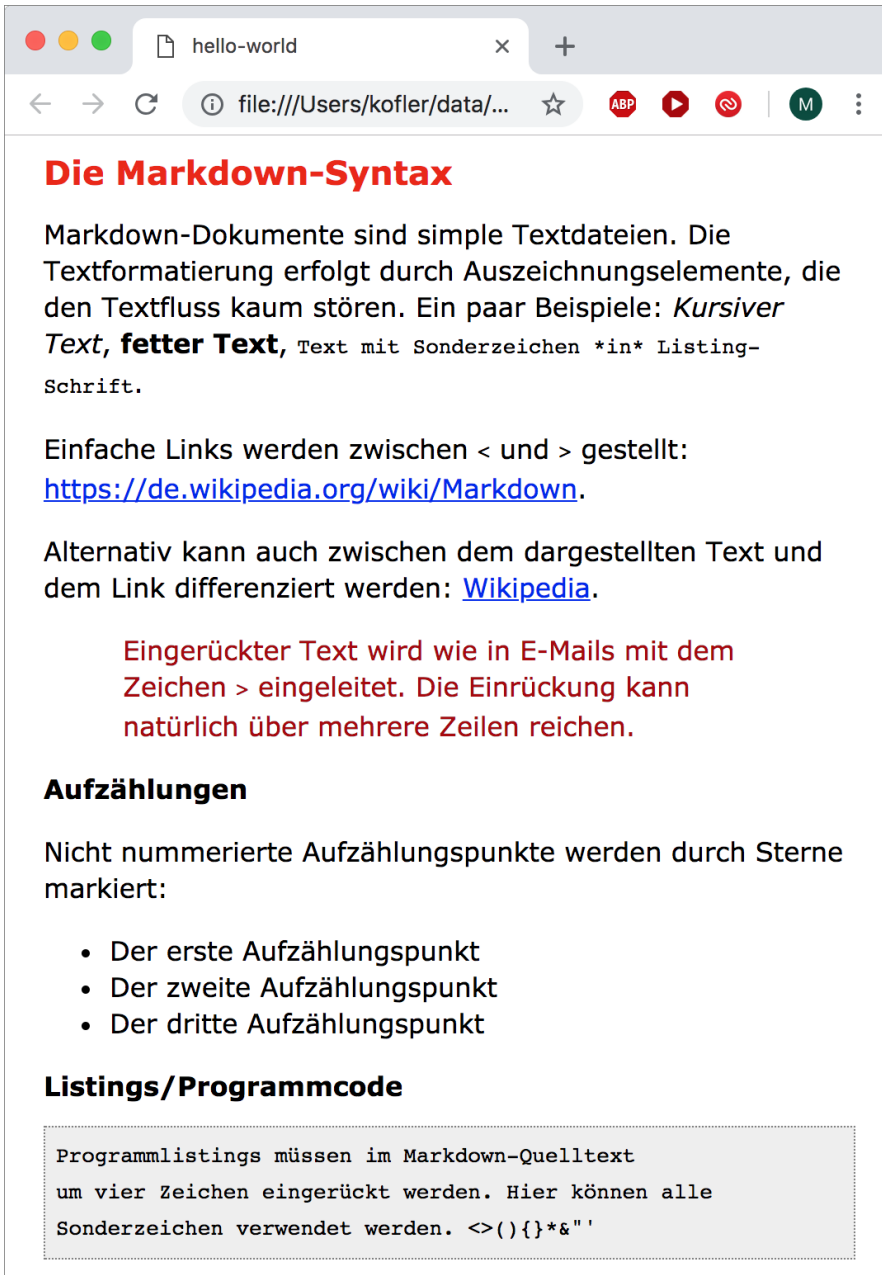


Abbildung 1.2: HTML-Darstellung des Hello-World-Textes mit CSS-Formatierung

## 1.2 Hintergrund, Geschichte

### Das originale Markdown von John Gruber

Markdown wurde 2004 von John Gruber mit Unterstützung von Aaron Swartz entwickelt. John Gruber ist in Apple-Kreisen als Betreiber der beliebten Website [Daring Fireball](#) bekannt. Auf dieser Website finden Sie auch die [Referenz](#) zur originalen Markdown-Syntax sowie den [Download-Link](#) eines relativ kleinen Perl-Scripts, das Markdown-Texte in das HTML-Format übersetzt.

John Gruber sieht Markdown ausschließlich als Werkzeug, um HTML-Seiten bequemer zu verfassen. Aus diesem Grund ist seine Markdown-Syntax auf das absolute Minimum beschränkt; sein Markdown-Konverter sieht einzig eine Übersetzung von Markdown nach HTML vor.

### Markdown-Varianten und -Implementierungen

Die Grundidee von Markdown hat viele Web-Autoren und -Entwickler auf Anhieb begeistert. Websites wie [Stack Overflow](#) oder [GitHub](#) akzeptieren die Markdown-Syntax bei der Eingabe von Texten. Darüber hinaus existieren Markdown-Parser in allen erdenklichen Programmiersprachen und machen es so einfach, Markdown in eigenen Programmen bzw. auf eigenen Websites zu integrieren.

Mit der zunehmenden Verbreitung von Markdown wuchsen freilich auch die Wünsche der Anwender: Das Erzeugen simpler HTML-Seiten war nur der Anfang. Viele Autoren wollen mit Markdown auch Folien für Vorträge gestalten, Handbücher, Hilfetexte, Artikel, wissenschaftliche Arbeiten, Bücher bzw. E-Books verfassen, E-Mails schreiben und ihre Markdown-Texte schließlich in alle anderen erdenklichen Formate umwandeln (z. B. PDF, LaTeX, LibreOffice, Microsoft Word).

Der originalen Markdown-Syntax fehlen dazu aber viele Features: Die Gestaltung von Tabellen, die spezifische Hervorhebung verschiedener Sprachelemente in Listings (Syntax-Highlighting), Querverweise im Text, Fußnoten, mathematische Formeln,

Literaturangaben und -verzeichnisse etc. sind mit dem originalen Markdown nicht bzw. nur mit großen Einschränkungen möglich.

Insofern verwundert es nicht, dass im Verlauf der Zeit diverse Markdown-Dialekte entstanden sind, die zwar auf Grubers Markdown basieren, darüber hinaus aber alle möglichen Zusatzfunktionen realisieren. Die [Markdown-Projektseite](#) auf GitHub listet über 50 Markdown-Implementationen auf!

Am weitesten geht in dieser Hinsicht das Programm *Pandoc*, das im Mittelpunkt dieses Buchs steht: Eine stark erweiterte Markdown-Syntax dient hier als Fundament, um Dokumente zwischen allen erdenklichen Formaten zu konvertieren. Zur Zeit gibt es kein anderes Markdown-Programm, das derart viele und weitreichende Anwendungsmöglichkeiten bietet.

<https://pandoc.org>

Vielleicht haben Sie es schon befürchtet: Auch wenn die originale Markdown-Syntax der unumstrittene gemeinsame Nenner aller Markdown-Varianten ist, sind diverse Erweiterungen unterschiedlich implementiert und führen zu Syntaxinkompatibilitäten. Wenn Sie Markdown-Erweiterungen nutzen möchten, müssen Sie sich daher auf ein bestimmtes Werkzeug festlegen.

Bevor Sie jetzt verzweifelt zu lesen aufhören: So schlimm, wie es hier möglicherweise klingt, ist es nicht. Die Basis ist bei allen Markdown-Dialekten dieselbe. Wenn Sie darüber hinaus Markdown-Erweiterungen nutzen möchten, müssen Sie sich für einen Markdown-Dialekt entscheiden – und dabei hängt es ganz davon ab, was Sie eigentlich mit Markdown machen möchten. Wenn Sie primär kurze Texte schreiben (Blogs, E-Mails), brauchen Sie die meisten Markdown-Erweiterungen gar nicht.

Erst wenn Sie daran denken, umfassende Dokumente mit Markdown zu gestalten (also Bücher, E-Books, technische Dokumentationen), müssen Sie sich für eine Markdown-Variante entscheiden. Für solche Fälle empfehle ich Ihnen Pandoc. Dieses Programm bietet momentan bei weitem die meisten Möglichkeiten; zudem wird Pandoc aktiv weiter entwickelt und gewinnt ständig neue Benutzer.

## Markdown-Alternativen

Markdown ist natürlich nicht die einzige textbasierte Syntax, um Texte zu verfassen und dann in andere Formate umzuwandeln. Zu den wichtigsten Alternativen zählen die folgende Formate:

- **Wikitext** ([Link](#)) ist die Syntax der Wikipedia; sie kommt in verschiedenen Varianten in vielen anderen Wiki-Systemen zum Einsatz.
- **AsciiDoc** ([Link](#)) ist ein etabliertes Werkzeug zum Verfassen technischer Dokumente, die dann in zahlreiche Formate konvertiert werden können (HTML, PDF, ePub etc.).
- **reStructuredText (ReST)** ([Link](#)) ist Bestandteil der Dokumentationswerkzeuge der Programmiersprache Python. ReST kann aber auch losgelöst von Python verwendet werden, um technische Dokumente zu verfassen und diese dann in diverse Formate zu übersetzen (HTML, LaTeX, XML etc.)

# 2 Werkzeuge

Dieses Kapitel beschreibt die Installation von Pandoc bzw. des originalen Markdown-Konverters. Außerdem stelle ich Ihnen einige Editoren mit Markdown-Unterstützung vor.

## 2.1 Pandoc installieren

Damit Sie Ihre Markdown-Texte in das HTML-Format oder in ein anderes Format umwandeln können, benötigen Sie einen Markdown-Konverter. Ich empfehle Ihnen die Installation von Pandoc, dem zur Zeit vielseitigsten Markdown-Konverter. Alternativ können Sie aber auch das Perl-Script `Markdown.pl` von John Gruber oder einen anderen Markdown-Konverter installieren.

### **Hinweis**

*Es gibt einige Editoren, die einen Markdown-Konverter gleich mitbringen. Für erste Experimente ist das durchaus bequem. Allerdings legen Sie sich damit auf die vom jeweiligen Editor unterstützte Markdown-Variante fest, die weniger Funktionen bietet als Pandoc.*

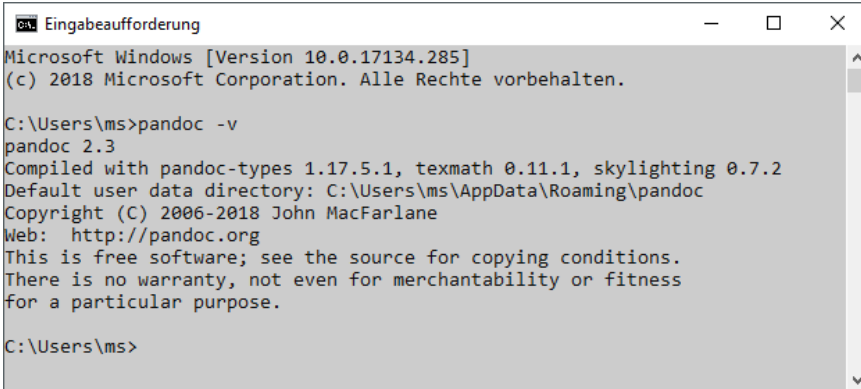
### **Pandoc unter Windows installieren**

Pandoc steht auf der folgenden GitHub-Seite zum kostenlosen Download als MSI-Datei bereit:

<https://github.com/jgm/pandoc/releases>

Ein Doppelklick auf die MSI-Datei und eine Bestätigung der Open-Source-Lizenz GPL – mehr ist nicht zu tun, um Pandoc zu installieren. Wenn Sie sich vergewissern möchten,

dass alles funktioniert hat, öffnen Sie ein Eingabeaufforderungsfenster und führen dort `pandoc -v` aus.



```
Eingabeaufforderung
Microsoft Windows [Version 10.0.17134.285]
(c) 2018 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\ms>pandoc -v
pandoc 2.3
Compiled with pandoc-types 1.17.5.1, texmath 0.11.1, skylighting 0.7.2
Default user data directory: C:\Users\ms\AppData\Roaming\pandoc
Copyright (C) 2006-2018 John MacFarlane
Web: http://pandoc.org
This is free software; see the source for copying conditions.
There is no warranty, not even for merchantability or fitness
for a particular purpose.

C:\Users\ms>
```

Abbildung 2.1: Pandoc-Versionsinformationen unter Windows lesen

## Pandoc unter Linux installieren

Unter Linux ist die Installation besonders einfach: Die meisten Distributionen liefern standardmäßig entsprechende Pakete mit, die Sie unkompliziert mit dem Paketverwaltungswerkzeugen installieren können. Unter Ubuntu führen Sie beispielsweise einfach `apt install pandoc` aus.

Wenn Ihre Distribution keine Pandoc-Pakete anbietet oder wenn diese veraltet sind (leider keine Seltenheit), müssen Sie zuerst die exotische Programmiersprache Haskell installieren (unter Ubuntu mit `apt install haskell-platform`). Haskell verfügt mit dem Kommando `cabal` über ein eigenes Paketverwaltungssystem. Damit installieren Sie nun die gerade aktuelle Pandoc-Version aus einer Haskell-Paketquelle:

```
cabal update
cabal install pandoc
```

Pandoc wird nun heruntergeladen und kompiliert. Dieser Prozess nimmt einige Minuten in Anspruch. Sollten dabei Fehler auftreten, stellen Sie sicher, dass alle Pakete

Ihrer Linux-Distribution aktuell sind. Unter Ubuntu führen Sie dazu `apt update` sowie `apt full-upgrade` aus. Anschließend versuchen Sie es nochmals. Das eigentliche Pandoc-Kommando wird in das Verzeichnis `/home/<username>/.cabal/bin` installiert. Damit Sie Pandoc ohne Angabe des Pfads direkt aufrufen können, sollten Sie dieses Verzeichnis Ihrer `PATH`-Umgebungsvariable hinzufügen. Dazu laden Sie die Datei `.bashrc` in einen Editor und fügen am Ende der Datei die folgende Zeile hinzu:

```
am Ende von .bashrc
...
export PATH=$PATH:~/cabal/bin
```

Nachdem Sie das Terminal geschlossen und neuerlich geöffnet haben, vergewissern Sie sich, dass alles funktioniert hat:

```
pandoc -v
pandoc 2.3
Compiled with pandoc-types 1.17.5.1, ...
Default user data directory: /home/kofler/.pandoc
Copyright (C) 2006-2018 John MacFarlane
```

### **Tipp**

*Wenn Sie zu einem späteren Zeitpunkt Pandoc auf eine neue Version aktualisieren möchten, wiederholen Sie einfach die beiden oben angegebenen `cabal`-Kommandos.*

## **Pandoc unter macOS installieren**

Ein fertiges Installationspaket für macOS finden Sie auf der [Pandoc-Download-Seite](#). Die Installation verläuft unkompliziert und schnell. Nach der Installation öffnen Sie ein Terminalfenster und führen dort wie unter Linux das Kommando `pandoc -v` aus. Pandoc zeigt dann die Versionsnummer an.



## Pandoc verwenden

Pandoc enthält keine grafische Benutzeroberfläche und ist somit nach der Installation unsichtbar; es gibt kein neues Icon oder Programm im Startmenü. Pandoc wird ausschließlich als Kommando in einem Eingabeaufforderungsfenster (Windows) bzw. in einem Terminal (Linux, macOS) ausgeführt.

Eine kurze Bedienungsanleitung zu Pandoc liefert man `pandoc` (nur unter Linux und macOS). Eine lange Liste aller Optionen erhalten Sie mit `pandoc --help`. Im einfachsten Fall, d. h., wenn Sie eine Markdown-Datei in das HTML-Format konvertieren möchten, rufen Sie `pandoc` so auf:

```
pandoc -o output.html input.md
```

Wenn Sie weder die Option `-o` noch eine Markdown-Datei angeben, liest `pandoc` die zu verarbeitenden Daten aus der Standardeingabe und schreibt das resultierende HTML-Dokument in die Standardausgabe.

Zum Erzeugen von HTML-Dokumenten ist es meist zweckmäßig, auch die Optionen `-s` sowie `-c name.css` zu verwenden. Damit erreichen Sie, dass das HTML-Dokument mit den vom HTML-Standard vorgeschriebenen Headern ausgestattet und die angegebene CSS-Datei eingebunden wird. Einen Überblick über weitere Optionen und andere Anwendungsformen des `pandoc`-Kommandos gibt das Kapitel [Das Pandoc-Kommando](#).

## Original-Markdown installieren

Anstelle von Pandoc können Sie auch das Perl-Script `Markdown.pl` von John Gruber oder einen anderen Markdown-Konverter installieren. `Markdown.pl` ist in einem kleinen ZIP-Archiv enthalten, das Sie von der Website [Daring Fireball](#) herunterladen.

### Hinweis

*Zur Ausführung von `Markdown.pl` benötigen Sie die Programmiersprache Perl. Der Perl-Interpreter steht unter Linux und macOS standardmäßig zur Verfügung. Sollten Sie unter Windows arbeiten, müssen Sie Perl selbst installieren. Download-Links finden Sie [hier](#).*

Nach dem Download von Markdown können Sie mit `perldoc Markdown.p1` die Bedienungsanleitung lesen. Um die Markdown-Datei `input.text` zu verarbeiten und daraus `output.html` zu erzeugen, führen Sie Markdown so aus:

```
Markdown.p1 input.text > output.html
```

## 2.2 Markdown-Editoren

Grundsätzlich können Sie Markdown-Dokumente mit *jedem beliebigen* Editor verfassen. Spezielle Markdown-Editoren bzw. -Erweiterungen machen diesen Prozess aber komfortabler, sei es durch Syntax-Highlighting, durch eine Live-Vorschau des Dokuments oder durch integrierte Export-Kommandos.

Diesen Vorteil erkaufen Sie aber in vielen Fällen mit einem wesentlichen Nachteil: Die meisten Markdown-Editoren unterstützen nur einen bestimmten Markdown-Dialekt, dessen Syntax weit weniger reichhaltig ist als bei Pandoc.

### Markdown im Webbrowser

Ganz ohne Installation können Sie Markdown in einem Web-basierten Editor ausprobieren und erlernen. Dabei stehen gleich mehrere Angebote zur Auswahl. Empfehlenswert sind die beiden folgenden Seiten:

- <https://stackedit.io>: *StackEdit* sieht mit seiner Symbolleiste fast wie ein »echtes« Programm aus (siehe die folgende Abbildung). Es unterstützt einige Markdown-Erweiterungen, z. B. Tabellen, mathematische Formeln und einfache Diagramme.
- <https://dillinger.io>: *Dillinger* bietet zwar weniger Funktionen als StackEdit, enthält dafür aber eine praktische eine PDF-Export-Funktion.

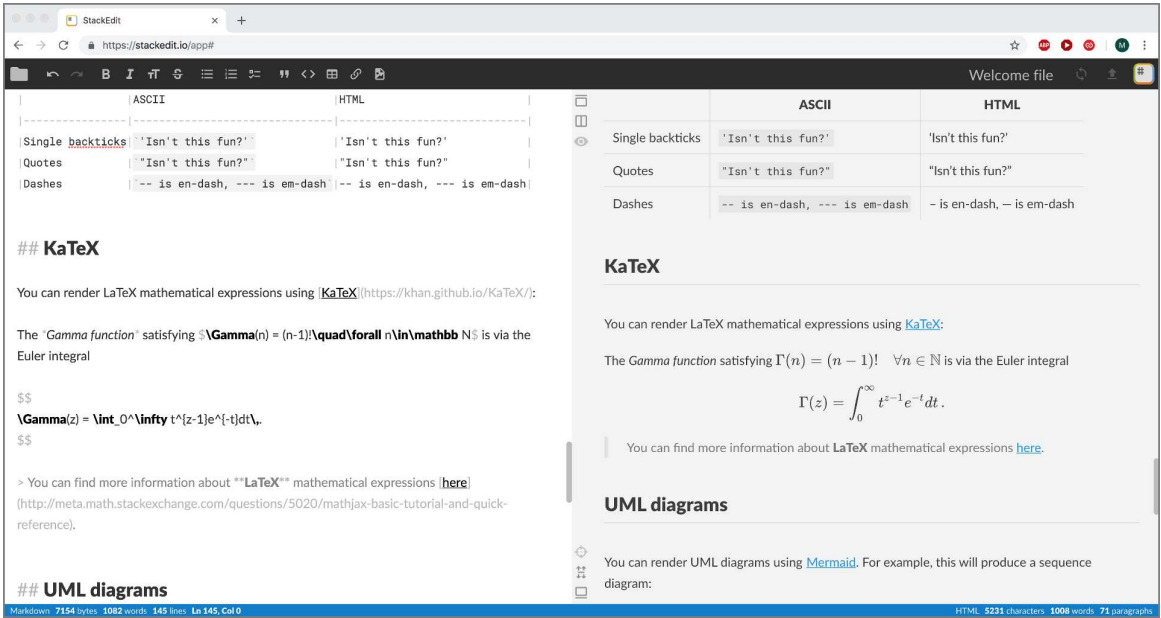


Abbildung 2.2: SlackEdit – ein Markdown-Editor im Webbrowser

## Editoren für Einsteiger ...

In den vergangenen Jahren wurden unzählige Programme vorgestellt, die das Verfassen von Markdown-Texten leichter machen sollten. Langfristig durchsetzen konnte sich bisher jedoch keines dieser Programme. Vielfach ist die Entwicklung nach einem vielversprechenden Start schnell wieder eingeschlafen. Aus Platzgründen nenne ich an dieser Stelle nur einige populäre Vertreter.

- Als bekanntester Markdown-Editor unter Windows gilt vermutlich noch immer **MarkdownPad**. Seine Bekanntheit verdankt dieses Programm seiner Vorreiterrolle. Ende 2014 wurde die Entwicklung eingestellt.
- Vielversprechend sieht der plattformunabhängige Editor **Haroopad** aus (siehe die folgende Abbildung). Allerdings stammt der letzte GitHub-Commit auch in diesem Fall aus dem Jahr 2014.

- Unter dem Namen **Typora** steht aktuell ein kommerzieller Editor für die Plattformen Windows, macOS und Linux in Entwicklung. Als ich dieses E-Book verfasste, befand sich das Programm noch im Beta-Test und war kostenlos verfügbar.
- Nur unter macOS ist das kommerzielle Programm **Marked2** verfügbar. Anders als der Name vermuten lässt, handelt es sich dabei *nicht* um einen Editor. Marked überwacht lediglich Markdown-Dateien und erzeugt nach jedem Speichervorgang eine neue Vorschau des aktuellen Dokuments. Dieser minimalistische Ansatz erweitert jeden Texteditor um eine Markdown-Vorschau.
- Unter Linux können die Standardeditoren *Gedit* und *KWrite* zumindest rudimentär Markdown-Syntaxelemente hervorheben.

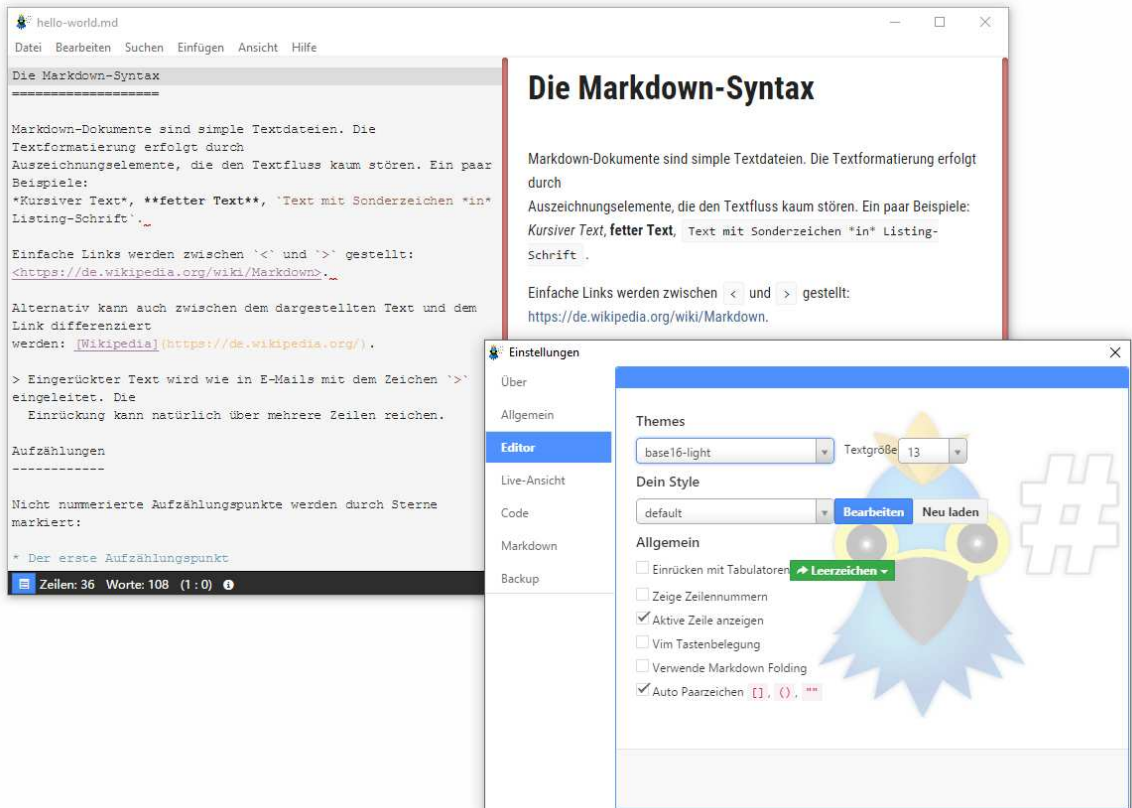


Abbildung 2.3: Haroopad bietet viele Einstellmöglichkeiten und steht für Windows, Linux und macOS zur Verfügung.

... und für Profis

Entwicklern und fortgeschrittenen Anwendern empfehle ich, ganz einfach bei ihren Lieblings-Editoren zu bleiben. Für nahezu alle gängigen Editoren, z. B. für *Atom* und *VS-Code*, aber auch für »Urgesteine« wie *Emacs* oder *Vi* (vim) gibt es Markdown-Erweiterungen. Deren Installation ist in der Regel nicht schwierig.

Wenn Sie mit keinem dieser Editoren vertraut sind, sollten Sie sich die Mühe machen, einen derartigen Editor zu erlernen. Das kostet zugegebenermaßen anfangs Zeit, lohnt sich aber längerfristig. Heute würde meine Wahl wohl auf *Atom* fallen (<https://atom.io>). Dieses für Windows, macOS und Linux kostenlos verfügbare Programm ist nahezu un- eingeschränkt konfigurier- und erweiterbar und erfüllt nahezu jeden Wunsch. Tipps zur Integration von Atom mit Pandoc finden Sie im Abschnitt [Atom und Pandoc kombinieren](#).

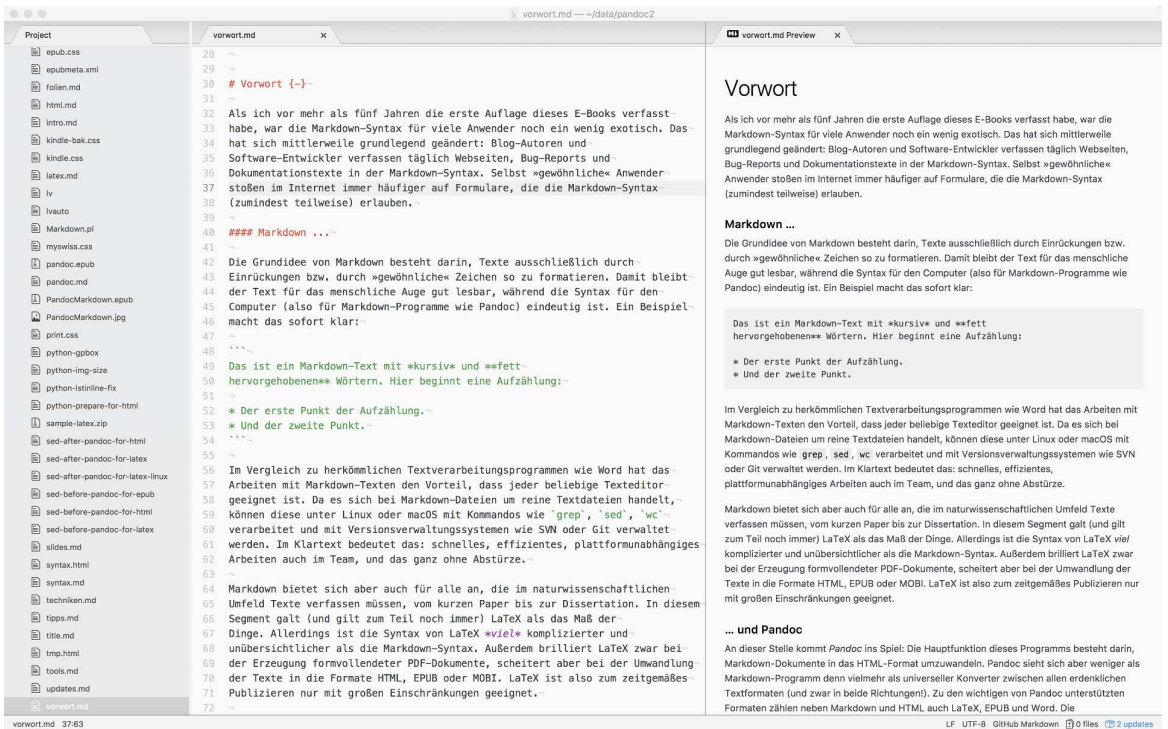


Abbildung 2.4: Mit der Installation weniger Zusatz-Packages wird Atom zum perfekten Markdown-Editor.

**Emacs über alles ...**

*Wenn man sich einmal an einen Editor gewöhnt hat, wechselt man nur ungern sein wichtigstes Schreibwerkzeug. Das erklärt, warum ich dieses E-Book – wie schon zuvor fast 100 andere Bücher zuvor – mit meinem Lieblingeditor Emacs verfasst habe. Für das Syntaxhighlighting habe ich die folgende Markdown-Erweiterung installiert:*

<https://jblevins.org/projects/markdown-mode>

*Zur Vorschau des fertigen Markdown-Dokuments verwende ich üblicherweise einen Webbrowser oder PDF-Viewer. Vorher muss ein Markdown-Konverter, in meinem Fall Pandoc, natürlich manuell ausgeführt werden. In meinem Setup kümmert sich darum ein Script, das im Hintergrund läuft und nach jedem Speichervorgang automatisch startet.*

# 3 Die Markdown-Syntax

Dieses Kapitel fasst die Markdown-Syntax zusammen. Syntaxelemente ohne nähere Erläuterungen gelten für alle Markdown-Dialekte, beziehen sich also auf die originale Markdown-Syntax von John Gruber. Daneben gehe ich aber auch auf die Erweiterungen der wichtigsten Markdown-Dialekte ein, wobei der Schwerpunkt bei *Pandoc* und *GitHub Flavored Markdown* (kurz GFM) liegt.

Ich habe mich bemüht, die Markdown-Syntax mit all ihren Erweiterungen in diesem Kapitel möglichst vollständig wiederzugeben. Auf die Beschreibung exotischer Sonderfälle habe ich allerdings verzichtet, damit die Darstellung nicht allzu unübersichtlich wird. In der folgenden Tabelle finden Sie Querverweise auf die jeweilige offizielle Syntaxbeschreibung:

| Kürzel                         | Syntaxbeschreibung                                                                                                    |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Original                       | <a href="https://daringfireball.net/projects/markdown/syntax">https://daringfireball.net/projects/markdown/syntax</a> |
| Pandoc                         | <a href="https://pandoc.org">https://pandoc.org</a>                                                                   |
| GitHub Flavored Markdown (GFM) | <a href="https://github.github.com/gfm">https://github.github.com/gfm</a>                                             |
| MultiMarkdown                  | <a href="https://github.com/fletcher/MultiMarkdown">https://github.com/fletcher/MultiMarkdown</a>                     |
| Markdown Extra                 | <a href="https://michelf.ca/projects/php-markdown/extra">https://michelf.ca/projects/php-markdown/extra</a>           |
| kramdown                       | <a href="https://kramdown.gettalong.org/">https://kramdown.gettalong.org/</a>                                         |

Tabelle 3.1: Die Syntaxbeschreibungen der wichtigsten Markdown-Dialekte

## 3.1 Absätze

In Markdown werden Absätze und andere Textelemente durch leere Zeilen voneinander getrennt. Zeilenumbrüche spielen hingegen keine Rolle für die Formatierung eines Absatzes. Der Absatz wird automatisch im Webbrowser umbrochen, abhängig davon, wie breit das Browserfenster bzw. die jeweilige Textspalte gerade ist.

*Whitespace* (also Leer- und Tabulatorzeichen) zwischen Wörtern wird so behandelt, als würde es sich um ein einziges Leerzeichen handeln. Kritischer sind Leerzeichen am Beginn der Zeile, durch die der Text eingerückt wird. Ab vier Zeichen Einrückung interpretiert Markdown den Text als Code-Listing und formatiert ihn anders. Kleinere Einrückungen werden zwar toleriert, sollten aber dennoch vermieden werden.

```
Das ist ein Absatz. Lange Abstände
im Quelltext werden ignoriert. Der
Absatz endet mit einer leeren Zeile.
```

```
Hier beginnt der nächste Absatz.
```

Das ist ein Absatz. Lange Abstände im Quelltext werden ignoriert. Der Absatz endet mit einer leeren Zeile.

Hier beginnt der nächste Absatz.

### **Hinweis**

*Viele Beispiele in diesem Kapitel folgen dem obigen Muster: Zuerst ist der Markdown-Quellcode abgedruckt, dann folgt eine Box, die zeigt, wie der Text in einem resultierenden HTML- oder PDF-Dokument aussieht.*

### **Absatzende je nach Markdown-Dialekt!**

*Beachten Sie, dass manche, gerade in Blog- oder Content-Management-Systemen weit verbreitete Markdown-Dialekte jeden Zeilenumbruch als Absatzende betrachten und daher mit jeder neuen Zeile einen neuen Absatz beginnen. In diesem Fall müssen Sie Absätze in extrem langen Zeilen formulieren.*



*Es ist schwer zu begreifen, warum sich die Markdown-Entwickler in einem so elementaren Punkt uneinig sind und vom Standardverhalten abweichen. Bei manchen Markdown-Editoren können Sie mit einer Option zwischen den beiden Absatztrennungsverfahren umschalten. Wenn Sie möchten, dass sich auch Pandoc so verhält, rufen Sie das Kommando mit der Option `-hard_line_breaks` auf. Weitere Details finden Sie im Abschnitt [Pandoc-Kompatibilität](#).*

## Textausrichtung ändern

Markdown sieht keine Möglichkeit vor, die Ausrichtung von Text zu beeinflussen (z. B. um einen Text zu zentrieren) oder um Blocksatz ein- und auszuschalten. Natürlich können Sie durch eine CSS-Datei festlegen, dass im resultierenden HTML-Dokument *alle* Absätze im Blocksatz gesetzt werden sollen, aber Sie können im originalen Markdown nicht gezielt einige Absätze so und einige weitere anders formatieren.

### **Tipp**

*Wenn Sie als einziges Ausgabeformat HTML wünschen, können Sie den betreffenden Text in `<div class="myclass">` verpacken und in der CSS-Datei für `myclass` jede beliebige Formatierung vornehmen. Diese Vorgehensweise funktioniert aber nicht, wenn Sie Ihr Markdown-Dokument mit Pandoc in andere Formate als HTML umwandeln.*

## Zeilenumbruch

Wenn Sie innerhalb eines Absatzes einen Zeilenumbruch erzwingen möchten, geben Sie am Ende der betreffenden Zeile zwei oder mehr Leerzeichen an. Beachten Sie aber, dass viele Editoren davon ausgehen, diese Leerzeichen wären überflüssig, und sie aus Ihrem Text wieder eliminieren. Dann scheitert die Kennzeichnung des Zeilenumbruchs. Suchen Sie nach der entsprechenden Option Ihres Editors und deaktivieren Sie diese!

Viele Markdown-Implementierungen sehen zusätzliche Möglichkeiten zur Kennzeichnung fester Zeilenumbrüche vor. In Pandoc gilt: Wenn eine Zeile mit dem Zeichen `\` endet, erfolgt an dieser Stelle ein fester Zeilenumbruch.

## Linienblöcke (Pandoc)

Pandoc kennt sogenannte Linienblöcke. Dabei wird der Markdown-Quelltext durch eine vertikale Linie am rechten Rand gekennzeichnet. Bei derartig formatiertem Text bleiben Zeilenumbrüche und Einrückungen erhalten. Ansonsten gelten aber die üblichen Markdown-Regeln, etwa für Hervorhebungen. Linienblöcke sind insbesondere für Gedichte oder Verse geeignet.

```
| Ach, was muss man oft von bösen
| Kindern hören oder lesen!
| Wie zum Beispiel hier von diesen,
| welche Max und Moritz hießen.
```

Ach, was muss man oft von bösen  
Kindern hören oder lesen!  
Wie zum Beispiel hier von diesen,  
welche **Max und Moritz** hießen.

## 3.2 Zeichensätze und Sonderzeichen

Markdown schreibt keinen bestimmten Zeichensatz vor. Alle Markdown-Formatierungszeichen wie `*` oder `#` sind ASCII-Zeichen und werden daher in den meisten gängigen Zeichensätzen identisch dargestellt. Zeichen mit Codes größer als 127 belässt Markdown, wie sie sind.

Mit anderen Worten: Wenn Sie Ihr Markdown-Dokument im Zeichensatz Unicode (UTF-8) verfassen, verwendet auch das resultierende HTML-Ergebnis diesen Zeichensatz. Sollten Zeichen wie ä, ö, ü oder ß im Webbrowser falsch dargestellt werden, dann fehlt dem HTML-Dokument lediglich ein Header, der den Zeichensatz festschreibt. Details dazu können Sie im Kapitel [HTML-Dokumente](#) nachlesen.

**Verwenden Sie UTF-8!**

*Auch wenn die Markdown-Syntax Ihnen in diesem Punkt fast jede Freiheit gibt, empfehle ich Ihnen dringend, den Zeichensatz UTF-8 einzusetzen. Unter Linux und macOS stellt sich diese Frage gar nicht – dort kommt UTF-8 in allen Editoren standardmäßig zum Einsatz.*

*Anders sieht es unter Windows aus: Manche Windows-Editoren verwenden noch immer den Windows-typischen Zeichensatz CP1252, der weitgehend dem Zeichensatz Latin-1 entspricht. Falls Sie einen Markdown-Text im CP1252-Zeichensatz später in ein anderes Format als HTML umwandeln möchten, werden höchstwahrscheinlich Probleme auftreten. Diese lassen sich durch diverse Zusatzoptionen, Preprocessing etc. in den Griff bekommen – aber wozu die Mühe?*

*Verwenden Sie UTF-8 und ersparen Sie sich den Ärger! Das gilt insbesondere für Pandoc, das im Gegensatz zur originalen Markdown-Syntax explizit den UTF-8-Zeichensatz vorschreibt!*

**Sonderzeichen**

Sie müssen sich keine Gedanken um die korrekte Behandlung der Zeichen `<`, `>` und `&` machen. Markdown kümmert sich selbst darum und ersetzt diese Zeichen bei der HTML-Konvertierung durch die Codes `&lt;`, `&gt;` und `&amp;`.

Wenn Sie Markdown-Formatierungszeichen wie `*` direkt im Code verwenden möchten, müssen Sie unter Umständen das Zeichen `\` voranstellen – also z. B. `\*`, wenn Sie das Zeichen `*` am Beginn einer Zeile wünschen, ohne damit einen Aufzählungspunkt einzuleiten. Die folgende Tabelle gibt an, welche Zeichen im originalen Markdown durch `\`-Codes ausgedrückt (»quotiert«) werden können. Einige Markdown-Dialekte erweitern diese Liste um weitere Zeichen. Noch einfacher ist die Regel bei Pandoc: *Alle* Symbolzeichen dürfen quotiert werden.

| Markdown-Code | Ergebnis | Markdown-Code | Ergebnis |
|---------------|----------|---------------|----------|
| \\            | \        | \( \)         | ()       |
| \'            | '        | \[ \]         | []       |
| \_            | _        | \{ \}         | {}       |
| \.            | .        | \!            | !        |
| \*            | *        | \+            | +        |
| \-            | -        | \#            | #        |

Tabelle 3.2: Sonderzeichen

HTML-Sonderzeichen: `a < b`, `a > b`, `a & b`

Markdown-Sonderzeichen: `* _ # +`

Quotierte Markdown-Sonderzeichen: `\* \_ \# \+`

Kursiv: `*a+b*`. Auch kursiv: `*a\*b*`. So geht's nicht: `*a*b*`.

HTML-Sonderzeichen: `a < b`, `a > b`, `a & b`

Markdown-Sonderzeichen: `* _ # +`

Quotierte Markdown-Sonderzeichen: `* _ # +`

Kursiv: `a+b`. Auch kursiv: `a*b`. So geht's nicht: `ab*`.

**Hinweis**

Wie die obigen Beispiele zeigen, erkennt Markdown in vielen Fällen aus dem Kontext, dass alleinstehende Zeichen wie `*` oder `#` nicht zur Formatierung gedacht sind, und interpretiert Ihre Eingabe auch ohne vorangestelltes `\`-Zeichen korrekt. Das macht Ihren Markdown-Quelltext besser lesbar, was ja der Intention von Markdown entspricht. Das Voranstellen von `\` ist also immer zulässig, aber nur selten notwendig.

## Programmcode

Mit ``` leiten Sie in den Fließtext eingebetteten Programmcode ein. Ein weiteres derartiges Zeichen beendet die Codepassage. Der gesamte darin eingeschlossene Text wird unverändert in Listing-Schrift (z. B. Courier) dargestellt.

```
Kennzeichnung von Programmcode: `if(a*b>c) { run(); }`
```

```
Kennzeichnung von Programmcode: if(a*b>c){ run(); }
```

Damit auch das Zeichen ``` selbst dargestellt werden kann, muss der entsprechende Code doppelt eingegrenzt werden:

```
`` ` ` ``

`` bla`bla ``
```

```
`

bla`bla
```

## Feste Leerzeichen (Pandoc)

In Pandoc gilt ein quotiertes Leerzeichen (also `»\ «`) als festes Leerzeichen. Aus `3\ km` wird also `3 km`, und Sie können sich sicher sein, dass zwischen `3` und `km` kein Zeilenumbruch erfolgt. Sofern Ihr Editor es unterstützt, können Sie anstelle von `»\ «` auch das Unicode-Zeichen 160 (U+00A0) eingeben.

Bei der Konvertierung des Markdown-Texts wird aus festen Leerzeichen das Unicode-Zeichen 160 (HTML) bzw. eine Tilde `~` (LaTeX). Beachten Sie, dass viele Programme das Unicode-Zeichen 160 wie ein gewöhnliches Leerzeichen darstellen.

### Vorsicht

*Die Syntax `»\ «` ist an sich komfortabel, führt aber nach meinen Erfahrungen oft zu Problemen – und zwar dann, wenn der Editor den Text eines Absatzes neu umbricht. Dann kann es passieren, dass der Backslash am Ende einer Zeile landet und dort als fester Zeilenumbruch interpretiert wird.*

Wenn Pandoc mit der Erweiterung `smart` ausgeführt wird (also mit der Option `--from markdown+smart`), fügt es nach manchen Abkürzungen (z. B. `Mr.`) ein festes Leerzeichen ein. Zufriedenstellend funktioniert dieser Automatismus leider nur für englischsprachige Texte.

### Gedankenstriche (Pandoc)

Wenn Pandoc mit der gerade erwähnten Erweiterung `smart` ausgeführt wird, ersetzt es `--` durch einen kurzen und `---` durch einen langen Gedankenstrich, also durch `–` bzw. `—`. Außerdem wird `...` durch das Ellipsis-Zeichen `…` ersetzt.

### Weiche Trennzeichen (Pandoc)

Für manche Ausgabeformate können Sie Vorschläge für die korrekte Silbentrennung machen. Sofern Ihr Editor dazu in der Lage ist, fügen Sie in das Wort ein weiches Trennzeichen ein (*soft hyphen*, Unicode U+00AD). Unter Windows gelingt das zumeist mit der Tastenkombination `AltGr + -`.

Pandoc belässt das Zeichen unverändert, die meisten HTML-Browser und manche E-Book-Viewer können auch korrekt damit umgehen. Auch aktuelle LaTeX-Versionen interpretieren das Zeichen richtig. (Die Fehlermeldung *unicode char u8 not set up for use with LaTeX* deutet auf eine zu alte LaTeX-Version hin.)

### Anführungszeichen

Markdown und Pandoc belassen Anführungszeichen grundsätzlich unverändert. Wenn Sie Pandoc mit der Option `--from markdown+smart` ausführen (diese Option kommt auch bei diesem E-Book zum Einsatz), werden zusammengehörende einfache bzw. doppelte Anführungszeichen jeweils nach oben bzw. nach unten gerichtet, wie dies im englischen Sprachraum üblich ist. Zu sehen ist dies im folgenden Beispiel bei den Wörtern *amet* und *sadipscing*.

Wenn Sie wie im folgenden Beispiel bei den Wörtern *diam* und *eirmod* die im deutschen Sprachraum gebräuchlichen Anführungszeichen wünschen, müssen Sie die entsprechenden Sonderzeichen im Editor direkt eingeben (siehe auch hier in der [Wikipedia](#)).

| Zeichen | Unicode | Bedeutung                           |
|---------|---------|-------------------------------------|
| ,       | U+201A  | einfaches deutsches Apostroph unten |
| ‘       | U+2018  | einfaches deutsches Apostroph oben  |
| „       | U+201E  | doppelte deutsche Apostrophe unten  |
| “       | U+201C  | doppelte deutsche Apostrophe oben   |

Tabelle 3.3: Deutsche Apostrophe

```
Lorem ipsum »dolor« sit “amet”,
consetetur ‘sadipscing’ elitr,
sed „diam“ nonumy ,eirmod’ tempor ...
```

### 3.3 Kommentare

Markdown sieht keine direkte Möglichkeit vor, Kommentare in den Text zu integrieren – also Text, der im endgültigen Dokument nicht sichtbar ist. Da Markdown aber das Einbetten von HTML-Code in den Text erlaubt (siehe auch den Abschnitt [HTML- und LaTeX-Code](#)), können Sie auf HTML-Kommentare zurückgreifen.

```
<!-- Kommentar in einem
Markdown-Dokument -->
```

#### **Vorsicht**

*Beachten Sie, dass diese Kommentare im resultierenden HTML-Dokument zwar unsichtbar, aber sehr wohl enthalten sind und z. B. im HTML-Quelltext gelesen werden können! Wenn Sie sich Passwörter oder andere vertrauliche Informationen merken müssen, sind HTML-Kommentare also nicht der ideale Ort! Verwenden Sie gegebenenfalls beim Aufruf von Pandoc die Option `--strip-comments`, um alle Kommentare zu eliminieren.*

Falls Sie vorhaben, Ihr Dokument später in ein E-Book im EPUB-Format umzuwandeln, vermeiden Sie mehrere aufeinanderfolgende Minuszeichen innerhalb des Kommentars! EPUB basiert auf XHTML, und die XHTML-Syntax verbietet mehrere aneinandergereihte Minuszeichen in Kommentaren; der Kommentar `<!-- Option --abc -->` ist somit ein Syntaxfehler und führt dazu, dass die resultierende EPUB-Datei nicht dargestellt werden kann!

## 3.4 Dokumentstruktur und Überschriften

Es gibt zwei Möglichkeiten, Überschriften zu kennzeichnen.

- #-Syntax: Hier stellen Sie dem Überschriftentext ein bis sechs #-Zeichen voran und kennzeichnen auf diese Weise Überschriften erster bis sechster Ordnung. Optional dürfen nach dem Überschriftentext weitere #-Zeichen folgen; diese haben aber nur kosmetische Funktion und werden von Markdown ignoriert. Deswegen spielt ihre Anzahl auch keine Rolle (d. h., eine Überschrift dritter Ordnung wird mit drei #-Zeichen eingeleitet, kann aber am Ende ohne weiteres sechs #-Zeichen aufweisen).
- Die andere Variante sieht ein Unterstreichen der Überschrift durch die Zeichen = oder - vor; sie funktioniert nur für Überschriften erster und zweiter Ordnung. Es ist gleichgültig, wie weit sie die Überschrift unterstreichen. Für die Markdown-Syntax reicht sogar ein einziges Zeichen aus (= oder -).

```
Kapitelüberschrift (entspricht <h1> in HTML)
Abschnittsüberschrift (<h2>)
überschrift dritter Ordnung (<h3>)
überschrift vierter Ordnung (<h4>)
überschrift fünfter Ordnung (<h5>)
überschrift sechster Ordnung (<h6>)
```



```
Noch eine Kapitelüberschrift

Noch eine überschrift sechster Ordnung

Noch eine Kapitelüberschrift (<h1>)
=====

Noch eine Abschnittsüberschrift (<h2>)

Noch eine Abschnittsüberschrift (<h2>)
-
```

### Zusatzinformationen in Überschriften (Pandoc)

Überschriften werden normalerweise nicht nummeriert. Wenn Sie das wünschen, geben Sie beim Aufruf von Pandoc die Option `--number-sections` oder kurz `-N` an.

In Pandoc kann im Anschluss an den Überschriftentext mit `{#label}` ein Sprunglabel definiert werden. Außerdem können mit `{.class key=value}` Zusatzinformationen übergeben werden. Die beiden Syntaxerweiterungen sind kombinierbar.

```
überschrift {#abc .efg mykey=myvalue}
```

Pandoc macht daraus diesen HTML- bzw. LaTeX-Code:

```
<h1 id="abc" class="efg" data-mykey="myvalue">überschrift</h1>

\hypertarget{abc}{%
\section{überschrift}\label{abc}}
```

Pandoc kennt eine vordefinierte Klasse `.unnumbered`, die für Überschriften gedacht ist, die trotz der Pandoc-Option `--number-sections` nicht nummeriert werden sollen. Anstelle von `{.unnumbered}` ist auch die Kurzschreibweise `{-}` erlaubt.

```
Vorwort {-}
```

Pandoc macht daraus diesen HTML- bzw. LaTeX-Code:

```
<h1 id="vorwort" class="unnumbered">Vorwort</h1>

\hypertarget{vorwort}{%
\section*{Vorwort}\label{vorwort}}
\addcontentsline{toc}{section}{Vorwort}
```

Weitere Informationen zum Umgang mit Links im Text finden Sie im Abschnitt [Querverweise innerhalb von Markdown-Dokumenten](#).

### 3.5 Textformatierung (fett, kursiv ...)

#### Fetter und kursiver Text

Um Text kursiv oder fett hervorzuheben, stellen Sie ihn zwischen die Zeichen \* bzw. deren Verdoppelung \*\*. Anstelle von \* können Sie auch \_ verwenden. \* und \_ funktionieren auch innerhalb von Wörtern.

```
normal, *kursiv*, _auch kursiv_, **fett**, __auch fett__
```

Sonderfälle: abc\*def\*ghi, pdf\*write\*file

```
normal, kursiv, auch kursiv, fett, auch fett
```

Sonderfälle: abcdefghi, pdfwritefile

Wenn die Zeichen \* oder \_ von Leerzeichen umgeben werden, verlieren sie ihre besondere Bedeutung. Wenn für Markdown nicht erkennbar ist, ob \* zur Kennzeichnung kursiven Texts oder als normales Zeichen zu verwenden ist, müssen Sie \ voranstellen.

```
Mehr Sonderfälle: a*b*c = d, a * b * c = d, a\b*c = d
```

```
Mehr Sonderfälle: abc = d, a * b * c = d, a*b*c = d
```

## Pandoc-Eigenheiten

Im originalen Markdown ist es nicht vorgesehen, ein Wort gleichzeitig fett und kursiv hervorzuheben. In Pandoc verwenden Sie dazu einfach dreimal das Zeichen `*`.

```
Nur in Pandoc: ***fett und kursiv***
fett *und kursiv* weiter fett
```

```
Nur in Pandoc: fett und kursiv
fett und kursiv weiter fett
```

Außerdem führt die originale Markdown-Logik in der Praxis häufig zu ungewünschten Auszeichnungen – etwa bei `pdf_write_file`, das standardmäßig zu `pdfwritefile` wird. Aus diesem Grund gilt `_` in Pandoc und vielen anderen Markdown-Implementationen nur für ganze Worte, nicht aber für Wortteile. `pdf_write_file` ergibt somit `pdf_write_file`, `pdf*write*file` hingegen `pdfwritefile`.

## Hochstellen, Tiefstellen, Durchstreichen und Kapitälchen (Pandoc)

Die Zeichen `^` und `~` sind zum Hoch- und Tiefstellen vorgesehen:

- `H~2~0` ergibt  $H_2O$
- `I^2^S-Bus` ergibt  $I^2S$ -Bus
- `E=m*c^2^` ergibt  $E=m*c^2$

Der hoch- oder tiefgestellte Ausdruck darf keine Leerzeichen enthalten. Ist das jedoch erforderlich, muss den Leerzeichen `\` vorangestellt werden:

- `n^a + b^` liefert das vermutlich nicht beabsichtigte Ergebnis  $n^a + b^$
- `n^a\ +\ b^` liefert korrekt  $n^{a+b}$

Beachten Sie, dass Sie in Pandoc auch komplexere mathematische Formeln in LaTeX-Schreibweise formulieren können (siehe den Abschnitt [HTML- und LaTeX-Code](#)).

Zum Durchstreichen markieren Sie Text mit zwei Tilden:

- `~~durchgestrichen~~` ergibt ~~durchgestrichen~~

Zu guter Letzt bietet Pandoc die Möglichkeit, Text in Kapitälchen (*small caps*) darzustellen:

- `[Datei/Öffnen]{.smallcaps}` ergibt DATEI/ÖFFNEN

## 3.6 Aufzählungen und Listen

Simple Aufzählungen ohne Nummerierung erzeugen Sie, indem Sie den Einträgen die Zeichen `,` `+` oder `-` voranstellen. Die Zeichen sind gleichbedeutend, Sie können sie sogar innerhalb einer Aufzählung mischen (ein Punkt mit `,` der nächste mit `+`).

Kurze Aufzählungspunkte geben Sie einfach zeilenweise an. Es ist also nicht erforderlich, zwischen den Punkten leere Zeilen anzugeben.

```
* Deutschland
- Italien
+ Schweiz
```

- Deutschland
- Italien
- Schweiz

Bei umfangreicheren Aufzählungspunkten ist es zweckmäßig, den Text einzurücken und die Aufzählungspunkte durch Leerzeilen zu trennen. Das hat auch Konsequenzen für den resultierenden HTML-Code. Markdown verpackt nun jeden Aufzählungspunkt in einen Absatz (also `<li><p>text ...</p></li>`).

```
* Der erste Aufzählungspunkt
 reicht über zwei Zeilen.

* Jetzt kommt der zweite Punkt.
```

- Der erste Aufzählungspunkt reicht über zwei Zeilen.
- Jetzt kommt der zweite Punkt.

Wenn der Text *eines* Aufzählungspunkts über mehrere Absätze reicht, muss ab dem zweiten Absatz die jeweils erste Textzeile um vier Leerzeichen oder um ein Tabulatorzeichen eingerückt werden. Im Quelltext sieht das am schönsten aus, wenn Sie den gesamten Text um vier Zeichen einrücken, wie das folgende Beispiel zeigt:

Normaler Text.

- ```
* Der erste Aufzählungspunkt
  reicht über zwei Zeilen.

  Dieser Absatz setzt den ersten
  Aufzählungspunkt fort.

* Erst hier beginnt der zweite Aufzählungspunkt.
```

Normaler Text.

- Der erste Aufzählungspunkt reicht über zwei Zeilen.
Dieser Absatz setzt den ersten Aufzählungspunkt fort.
- Erst hier beginnt der zweite Aufzählungspunkt.

Die Einrückung von vier Zeichen für Folgeabsätze innerhalb eines Listenpunktes steht im Konflikt mit einer anderen Regel, die besagt, dass auch Listings von Programmen um vier Zeichen eingerückt sein sollen (siehe den Abschnitt [Listings](#)). Wenn Sie möchten, dass einem Aufzählungspunkt ein Listing und nicht ein weiterer Absatz folgt, müssen Sie die beiden Elemente durch einen HTML-Kommentar trennen:

```
* erstens

* zweitens

<!-- -->
```

```
class HelloWorld {  
    print("Hello World");  
}
```

- erstens
- zweitens

```
class HelloWorld {  
    print("Hello World");  
}
```

Verschachtelte Listen

John Gruber geht auf seiner Markdown-Syntaxseite zwar nicht auf diesen Fall ein, aber bereits das originale Markdown sieht verschachtelte Listen vor. Die Syntaxregeln dafür sind wie immer simpel: Der Text jeder Ebene muss um vier Zeichen, um ein Vielfaches von vier Zeichen bzw. um je ein Tabulatorzeichen eingerückt werden. Es bleibt Ihnen weiterhin freigestellt, welche Markierungszeichen Sie verwenden – aber es ist naheliegend, für jede Ebene eine andere Markierung zu verwenden.

```
* Belletristik  
* Kinder- und Jugendbücher  
* IT-Bücher  
  + Programmierung  
  + Datenbanken  
    - MySQL  
    - SQL Server  
    - IBM DB/2  
  + SAP
```

- Belletristik
- Kinder- und Jugendbücher
- IT-Bücher
 - Programmierung
 - Datenbanken
 - MySQL
 - SQL Server
 - IBM DB/2
 - SAP

Nummerierte Listen

Bei der Formulierung von Listen können Sie jedem Punkt auch eine Zahl voranstellen. Bei der Umwandlung zu HTML-Code macht Markdown daraus eine *ordered list* (<o1>).

Beachten Sie aber, dass Markdown die Zahlen selbst nicht interpretiert. Es ist also gleichgültig, wie Sie die Listenpunkte nummerieren. Anstelle von 1., 2., 3. liefert auch 3., 2., 1. oder 17., 12., 28. dasselbe Ergebnis!

1. Windows
2. macOS
3. Linux

1. Windows
2. macOS
3. Linux

Tipp

Sollte ein Absatz mit einer Zahl und einem Punkt beginnen, der Absatz aber kein Aufzählungspunkt sein, müssen Sie dem Punkt einen Backslash voranstellen, also z. B. 1\. Mai -- der Tag der Arbeit!.

In Pandoc gibt es bei nummerierten Listen zwei Verbesserungen gegenüber dem originalen Markdown:

- **Startnummer:** Pandoc wertet die *erste* Nummer einer Liste aus und beginnt die Aufzählung damit. Der Wert der weiteren Nummern ist dann egal.
- **Nummerierungsschema:** Pandoc erkennt auch Aufzählungspunkte, die so aussehen:
 a., b., c. ...
 1), 2), 3) ...
 (1), (2), (3) ...

Die resultierende Liste wird dann auch in HTML entsprechend formatiert.

Beispiellisten (Pandoc)

Pandoc sieht eine eigene Aufzählungsvariante speziell für durchnummerierte Beispiele vor. Im Markdown-Text werden die Punkte mit (@) oder mit (@label) gekennzeichnet. Pandoc kümmert sich um die richtige Nummerierung, wobei diese auch nach anderen Markdown-Elementen fortgesetzt wird. Mit @label können Sie auf einen bestimmten Punkt verweisen.

```
(@slides) Folien mit Pandoc gestalten
(@latex) Von Pandoc nach LaTeX

Ein normaler Absatz.

(@pdf) PDFs erzeugen

Wie ich in Punkt @latex erklärt habe, können Sie ...
```

```
(1)Folien mit Pandoc gestalten
(2)Von Pandoc nach LaTeX

Ein normaler Absatz.

(3)PDFs erzeugen

Wie ich in Punkt 2 erklärt habe, können Sie ...
```


Definitionslisten (Pandoc)

In Pandoc und einigen anderen Markdown-Dialekten können Sie auch sogenannte Definitionslisten erstellen: Dabei wird jeder Listeneintrag mit einer Überschrift ausgestattet. Definitionslisten können beispielsweise für ein Glossar, ein Abkürzungsverzeichnis oder für die Referenz der Optionen eines Kommandos verwendet werden.

Die zu definierenden Ausdrücke müssen in Markdown in einer Zeile Platz finden. Danach folgt üblicherweise eine leere Zeile (nicht zwingend erforderlich) und die Definition des Ausdrucks, die mit einem Doppelpunkt eingeleitet wird. Der weitere Text muss vier Zeichen eingerückt sein. Die Definition darf über mehrere Absätze reichen, die ebenfalls alle um vier Zeichen eingerückt sind.

MTA

```
: Der *Mail Transfer Agent* wird umgangssprachlich als
  E-Mail-Server bezeichnet. Der MTA kümmert sich
  darum, E-Mails über das Internet zu versenden bzw.
  zu empfangen, wobei das Protokoll SMTP eingesetzt wird.
```

MDA

```
: Der *Mail Delivery Agent* kümmert sich um die lokale
  Zustellung von E-Mails, also um die Speicherung der
  beim MTA eintreffenden E-Mails in lokalen
  Postfächern.
```

```
In viele MTAs ist ein MDA bereits integriert bzw. wird
mitgeliefert, z.\,B.\ das Kommando `local` bei Postfix.
```

MTA

Der *Mail Transfer Agent* wird umgangssprachlich als E-Mail-Server bezeichnet. Der MTA kümmert sich darum, E-Mails über das Internet zu versenden bzw. zu empfangen, wobei das Protokoll SMTP eingesetzt wird.

MDA

Der *Mail Delivery Agent* kümmert sich um die lokale Zustellung von E-Mails, also um die Speicherung der beim MTA eintreffenden E-Mails in lokalen Postfächern.

In viele MTAs ist ein MDA bereits integriert bzw. wird mitgeliefert, z. B. das Kommando `local` bei Postfix.

Auf HTML-Ebene werden Definitionslisten durch die Tags `<dl>` (Definition List), `<dt>` (Definition Term) und `<dd>` (Definition Definition) abgebildet. Standardmäßig sind die Definitionen in HTML eingerückt. Durch eine CSS-Datei können Sie eine optisch ansprechendere Formatierung erzielen.

Todo-Listen (GFM)

Github Flavored Markdown kennt eine eigene Syntax für Todo-Listen. Auf der resultierenden HTML-Seite werden erledigte Punkte durch ein Auswahlhäkchen gekennzeichnet.

- [x] Fehler 73234 beheben
- [x] Fehler 88234 beheben
- [] Fehler 43455 beheben
- [] Dokumentation fertigstellen

3.7 Einrückungen bzw. Zitate

Das Konzept ist vor allem von E-Mails vertraut: Beim Antworten wird vorhandener Text eingerückt, um so eine einfache Differenzierung zwischen Frage und Antwort zu ermöglichen. Markdown übernimmt das in Text-Mails übliche Einrückzeichen `>` zur Kennzeichnung von eingerücktem Text.

```
> Eingerückter Text beginnt mit dem  
> Zeichen >. Der Text kann über mehrere  
> Zeilen reichen.
```

```
> Es reicht aus, wenn nur die erste  
Zeile durch > gekennzeichnet ist.
```

Eingerückter Text beginnt mit dem Zeichen `>`. Der Text kann über mehrere Zeilen reichen.

Es reicht aus, wenn nur die erste Zeile durch `>` gekennzeichnet ist.

Pandoc unterstützt sogar mehrere Einrückebenen.

```
> Normale Einrückung  
  
>> Dieser Absatz ist zweifach eingerückt.  
  
>>> Und dieser sogar dreifach.
```

Normale Einrückung

Dieser Absatz ist zweifach eingerückt.

Und dieser sogar dreifach.

Im HTML-Code werden Einrückungen durch `<blockquote>`-Elemente abgebildet. Wie üblich kann die tatsächliche optische Gestaltung durch CSS-Einstellungen gesteuert werden. Innerhalb von Einrückungen sind andere Markdown-Textelemente zulässig, z. B. Überschriften, Listen oder Quellcode-Passagen.

3.8 Listings und Programmcode

Kurze Passagen

Kurze Textpassagen mit Programmcode oder Sonderzeichen, die von Markdown unverändert weitergegeben werden sollen, werden mit `~` eingeleitet und wieder abgeschlossen, d. h., `~@{}~` liefert `@{}`.

Damit auch das Zeichen ` selbst dargestellt werden kann, sieht Pandoc für solche Fälle eine Verdoppelung vor, d. h., der Ausdruck wird mit zwei `-Zeichen und einem Leerzeichen eingeleitet und mit einem Leerzeichen und zwei `-Zeichen beendet.

```
`` `abc` `` --> ergibt `abc`
```

Mehrzeilige Listings

Längere Listings, die über mehrere Zeilen reichen, werden in der Markdown-Syntax um vier Zeichen eingerückt. Im resultierenden HTML-Dokument wird der Text zwischen `<pre>` und `</pre>` verpackt. Sämtliche Sonderzeichen, Leerzeichen, Einrückungen und feste Zeilenumbrüche bleiben erhalten, außerdem wird der Text in einer nichtproportionalen Schrift dargestellt (z. B. Courier). Das Listing endet mit der ersten Zeile, die nicht eingerückt ist. Leere Zeilen innerhalb des Listings sind erlaubt; diese müssen nicht eingerückt sein.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Innerhalb des Listings sind alle sonst üblichen Markdown-Regeln deaktiviert. Damit fehlt aber auch die Möglichkeit, Passagen des Listings besonders zu kennzeichnen (z. B. fett hervorzuheben).

Wenn Listings innerhalb anderer Markdown-Elemente verwendet werden sollen, muss die Einrückung entsprechend um ein vier Zeichen bzw. um ein Vielfaches von vier Zeichen vergrößert werden. Wenn ein Listing also z. B. zusammen mit einem Aufzählungspunkt dargestellt werden soll, muss der Code um acht Zeichen eingerückt werden:

- * Punkt 1

- * Punkt 2

```
Listing zu Punkt 2
Fortsetzung des Listings
```

Noch ein Absatz zu Punkt 2.

- Punkt 1
- Punkt 2

```
Listing zu Punkt 2
Fortsetzung des Listings
```

Noch ein Absatz zu Punkt 2.

Listings ohne Einrückungen (Pandoc, GFM)

Bei einigen Markdown-Dialekten können Listings mit besonderen Zeichenkombinationen eingeleitet und wieder abgeschlossen werden. Diese Syntaxformen haben den Vorteil, dass der Code nicht eingerückt werden muss.

- **Pandoc:** Listings werden durch ````` oder durch `~~~` eingeleitet und abgeschlossen.
- **GitHub Flavored Markdown:** Listings werden durch ````` eingeleitet und abgeschlossen.

Bei allen angeführten Markdown-Dialekten sind auch mehr als drei Markierungszeichen erlaubt, also z. B. eine ganze `--`-Linie (`-----`).

```
~~~
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
~~~
```

Syntax-Hervorhebung (Pandoc, GFM)

Die originale Markdown-Syntax sieht keine Hervorhebung von Code-Bestandteilen vor. Listings werden unverändert weitergegeben. Auf vielen Websites wird Programmcode hingegen optisch ansprechender dargestellt: mit Kommentaren, Schlüsselwörtern etc. in unterschiedlichen Farben. Diesen Effekt (also das Syntax-Highlighting) können Sie auch in vielen Markdown-Dialekten erzielen. Dazu müssen Sie beim Listing die Programmiersprache angeben, wobei die Syntax variiert:

- **Pandoc:** ````sprache` oder `~~~sprache` oder ````{.sprache}` oder `~~~{.sprache}`
- **GitHub Flavored Markdown:** ````${sprache}`

Das folgende Beispiel verwendet die Pandoc-Syntax:

```
```java
// Hello World in Java
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello World!");
 }
}

```

```
// Hello World in Java
public class HelloWorld {
 public static void main(String[] args) {
 System.out.println("Hello World!");
 }
}
```

In Pandoc können Sie das Listing auch mit Zeilennummern und mit einem Sprunglabel für Querverweise ausstatten:

```
``` {.java .numberLines startFrom="10" #helloworld}
// Hello World in Java
public class HelloWorld {
```

```
public static void main(String[] args) {
    System.out.println("Hello World!");
}
}
---
```

```
10 // Hello World in Java
11 public class HelloWorld {
12     public static void main(String[] args) {
13         System.out.println("Hello World!");
14     }
15 }
```

Pandoc erlaubt außerdem auch die Syntaxhervorhebung für kurze Codepassagen zwischen ```-Zeichen. Dazu muss dem zweiten ```-Zeichen die Programmiersprache folgen. ``SELECT * FROM customers`{.sql}` liefert **SELECT** * **FROM** customers.

3.9 Links und Querverweise

Die Markdown-Syntax unterscheidet zwischen verschiedenen Link-Varianten:

- einfache Links, bei denen die Link-Adresse und der angezeigte Text übereinstimmen
- Inline-Links, bei denen der angezeigte Text und die Link-Adresse unterschiedlich sind
- Referenzen, bei denen die Link-Adresse getrennt definiert wird (oft in einem Link-Block am Ende des Dokuments)

Einfache Links

In Markdown können Sie einen einfachen Link in der Form `<https://google.com/>` schreiben. Im HTML-Dokument wird daraus der anklickbare Link <https://google.com/>.

In manchen Markdown-Dialekten können Links sogar ohne jede Kennzeichnung verwendet werden. Vorausgesetzt wird dabei zumeist, dass der Link korrekt mit `http://` oder `https://` beginnt oder zumindest mit `www.` anfängt.

Inline-Links

Die Syntax `[linktext] (https://www.xyz.com/)` für Inline-Links trennt den angezeigten Text von der Adresse. Aus `[Wikipedia] (https://de.wikipedia.org/)` wird im resultierenden HTML-Dokument [Wikipedia](https://de.wikipedia.org/), wobei das Anklicken des Links zur Seite <https://de.wikipedia.org/> führt.

Alternativ können Sie Inline-Links auch mit `[linktext] (http://www.xyz.com/ "Title")` formulieren. Das ändert nichts am Erscheinungsbild des Links im HTML-Dokument; allerdings wird nun zusätzlich der Link-Titel angezeigt, wenn die Maus über den Link bewegt wird (also ohne diesen noch anzuklicken).

Aus dem Markdown-Text `[Wikipedia] (https://de.wikipedia.org/ "Die Online-Enzyklopädie")` wird der folgende HTML-Code:

```
<a href="https://de.wikipedia.org/" title="Die Online-Enzyklopädie">
  Wikipedia</a>`
```

Links innerhalb der lokalen Website bzw. innerhalb des aktuellen Texts können auch in der Form `[Impressum] (impressum.html)` oder siehe `[hier] (#label)` formuliert werden. Sie verweisen dann auf den Beginn der betreffenden Datei bzw. auf die Überschrift mit dem angegebenen Label.

Referenzen

Der Nachteil von Inline-Links besteht darin, dass diese oft recht lang sind und die Lesbarkeit des Markdown-Texts stark beeinträchtigen. Deswegen bietet Markdown die Möglichkeit, die Link-Adressen im Anschluss an den eigentlichen Text zu formulieren. Der Fließtext enthält dann nur die Kurzform `[linktext] [label]`. Am einfachsten kann die Syntax anhand eines Beispiels verdeutlicht werden:

```
Aktuelle Informationen zur Programmiersprache Swift finden Sie
auf den folgenden Seiten:
```

```
* [Swift-Homepage] [1]: Das ist die offizielle Swift-Projektseite
von Apple.
```



```
* [Swift-Forum][2]: Hier wird über die Anwendung und
  Weiterentwicklung von Swift diskutiert.
```

```
[1]: https://swift.org
```

```
[2]: https://forums.swift.org
```

Das fertige Dokument sieht dann so aus:

Aktuelle Informationen zur Programmiersprache Swift finden Sie auf den folgenden Seiten:

- [Swift-Homepage](#): Das ist die offizielle Swift-Projektseite von Apple.
- [Swift-Forum](#): Hier wird über die Anwendung und Weiterentwicklung von Swift diskutiert.

Die Link-Adressen können an einer beliebigen Stelle im Markdown-Dokument in der Form `[label]: https://xxx.com/yyy "title"` definiert werden. Die Definition ist im resultierenden HTML-Dokument nicht sichtbar.

Alle derart definierten Links können in Markdown mit der Syntax `[linktext][label]` verwendet werden. Der Label verrät Markdown, welche Link-Adresse Sie verwenden möchten. Der Label kann aus Buchstaben und Zahlen zusammengesetzt werden. Dabei wird zwischen Groß- und Kleinschreibung nicht unterschieden, wie das folgende Beispiel demonstriert.

Wenn der Linktext mit dem Label übereinstimmt, können Sie auf den Label verzichten. Das zweite eckige Klammernpaar muss aber erhalten bleiben:

```
Auf der englischen Wikipedia finden Sie eine kompakte
Zusammenfassung der [Java-Syntax][].
```

```
[java-syntax]: https://en.wikipedia.org/wiki/Java_syntax
```

Auf der englischen Wikipedia finden Sie eine kompakte Zusammenfassung der [Java-Syntax](#).

Querverweise innerhalb von Markdown-Dokumenten (Pandoc)

Sobald Sie Markdown zum Verfassen längerer Texte verwenden, entsteht der Wunsch nach Querverweisen (also z. B. *siehe das Kapitel [Die Markdown-Syntax](#)*). Die originale Markdown-Syntax sieht diese Möglichkeit leider nicht vor. Sie können sich damit behelfen, dass Sie Sprungziele im Dokument als HTML-Code angeben: ``; dann können Sie an einer anderen Stelle einen Link auf dieses Ziel setzen, also `siehe [Zieltext] (#label)`.

Pandoc bietet die Möglichkeit, Überschriften manuell mit einem fixen Sprungziel auszustatten. Dazu wird der Überschriftentext durch `{#label}` erweitert:

```
# Die Markdown-Syntax {#syntax}
```

Querverweise auf das Kapitel erfolgen nun mit `siehe das Kapitel [Die Markdown-Syntax] (#syntax)`.

Die explizite Nennung von Sprunglabeln können Sie sich oft sparen. In Pandoc gelten nämlich alle Überschriften automatisch als Sprungziele. Bei der Konvertierung des Dokuments wird der gesamte Überschriftentext in Kleinbuchstaben zur Identifizierung verwendet (`id="..."` in HTML). Sonder- und Leerzeichen in der Überschrift werden durch das Zeichen `-` ersetzt. Die Überschrift `# Die Markdown-Syntax` wird somit automatisch durch `die-markdown-syntax` identifiziert. Ein Verweis auf diese Überschrift erfolgt in der Form `siehe das Kapitel [Die Markdown-Syntax] (#die-markdown-syntax)`. Noch komfortabler werden derartige Querverweise dadurch, dass Pandoc hierfür die Kurzsyntax `[Die Markdown-Syntax]` zulässt, sofern der Zieltext und der Abschnittstitel übereinstimmen.

Pandocs automatische Sprungziele funktionieren nicht nur in HTML-Ergebnissen, sondern auch für eBooks und LaTeX-Dokumente. Vergessen Sie aber nicht, dass Sie Querverweise ändern müssen, wenn Sie die Überschrift eines Abschnitts neu formulieren. Bei mehreren gleichlautenden Abschnitten kann nur die erste Überschrift für Querverweise verwendet werden; Abhilfe schafft dann die manuelle Definition von Sprungzielen.

3.10 Bilder

Die Syntax zum Einbetten von Bildern in einen Markdown-Text ist verwandt mit der Link-Syntax (siehe den Abschnitt [Links und Querverweise](#)). Der Ort des Bilds kann wahlweise *inline* oder in einer eigenen, durch eine ID gekennzeichnete Zeile angegeben werden:

```
![Alternativer Text](../bilder/xxx.eps)

![Alternativer Text](https://abc.com/xxx.png "Titel")

![Alternativer Text][id]

[id]: https://abc.com/images/xxx.png "Titel"
```

Wenn `` in einer eigenen Zeile angegeben wird, betrachtet Markdown das Bild als einen eigenen Absatz; andernfalls wird das Bild an der aktuellen Stelle in den Text integriert, was vor allem für sehr kleine Bilder zweckmäßig ist.

Beschriftung von Bildern (Pandoc)

Pandoc macht aus `![alt](bild.png "titel")` den folgenden HTML-Code:

```
<figure>
<figcaption>alt</
  figcaption>
</figure>
```

Beachten Sie, dass der Titeltext zwar für das Attribut `title` verwendet wird; als Bildunterschrift (`<figcaption>`) kommt aber der `alt`-Text zur Anwendung. Das ist insofern praktisch, weil damit in der Praxis zumeist auf "title" verzichtet werden kann, also:

```
![Bildunterschrift](bild.png)
```

Das Aussehen der Beschriftung (z. B. kursive Schrift) sowie eine eventuelle Nummerierung der Abbildungen können Sie per CSS-Code steuern. Ein entsprechendes Beispiel finden Sie im Kapitel [HTML-Dokumente](#).

Bildgröße und andere Parameter (Pandoc)

Pandoc versucht je nach Auflösung und DPI-Wert die Größe des Bilds selbst zu bestimmen. In aller Regel scheitert das: Manche Bilder erscheinen im Dokument zu groß, andere zu klein.

Abhilfe schafft eine manuelle Steuerung der Bildgröße. Dabei geben Sie in einem nachgestellten geschwungenen Klammernpaar die Parameter `width` und/oder `height` an. (Meist ist es sinnvoll, nur einen Parameter anzugeben; Pandoc behält dann die Proportionen des Bilds bei.) Die Angabe kann wahlweise absolut (z. B. `width=100pt` oder `height=2cm`) oder relativ zur Breite des Dokuments erfolgen (`width=25%`). Weitere erlaubte Maßeinheiten sind hier dokumentiert:

<http://pandoc.org/MANUAL.html#images>

Ebenfalls in geschwungenen Klammern können Sie einen Klassennamen sowie ein ID-Label angeben. Das hilft bei der CSS-Formatierung und ermöglicht Links in der Form `[Abbildung] (#idcode)`.

Pandoc wandelt den folgenden Markdown-Code

```
![Bildunterschrift](bild.png "titletext"){#idcode .classname width=30%}
```

in diesen HTML-Code um:

```
<figure>

<figcaption>Bildunterschrift</figcaption>
</figure>
```

Falls Sie LaTeX als Zielformat verwenden, sieht der resultierende Code wie folgt aus:

```
\begin{figure}
  \hypertarget{idcode}{%
  \centering
  \includegraphics[width=0.3\textwidth,height=\textheight]{bild.png}
  \caption{Bildunterschrift}
  \label{idcode}
}
\end{figure}
```

Beachten Sie, dass "titletext" in der LaTeX-Ausgabe nicht berücksichtigt wird. Das Bild wird trotz der irreführenden Anweisung `height=\textheight` korrekt mit 30% der Seitenbreite angezeigt. (Verantwortlich dafür sind einige Zeilen im LaTeX-Template von Pandoc. Wenn Sie von Pandoc erzeugten LaTeX-Code selbst verarbeiten, führen Sie `pandoc -D latex > default.latex` aus und werfen Sie einen Blick auf die Zeilen nach `\usepackage{graphicx}!`)

Wenn Sie Ihren Markdown-Text in ein PDF-Dokument umwandeln, werden Abbildungen automatisch nummeriert. In HTML-Dokumenten können Sie das durch CSS-Code erreichen. Leider fehlt in Pandoc eine plattformübergreifende Möglichkeit, um Verweise auf nummerierte Abbildungen (»siehe Abbildung 12.4«) durchführen zu können.

Sämtliche Parameter in den geschwungenen Klammern sind optional. Wenn Sie nur die Bildbreite einstellen möchten, reicht also dieser Markdown-Code:

```
![Bildunterschrift](bild.png){width=30%}
```

3.11 Tabellen

Das originale Markdown beinhaltet keine Möglichkeit zur Gestaltung von Tabellen. Einige Markdown-Dialekte füllen diese offensichtliche Lücke – leider auf unterschiedliche Art und Weise. Für alle Tabellenvarianten gilt, dass die optische Gestaltung der Tabelle durch eine CSS-Datei gesteuert werden kann. Verzichten Sie auf diese Option, sehen die resultierenden Tabellen eintönig aus.

Linientabellen (PHP Markdown Extra, Pandoc, GFM)

Die am meisten verbreitete Tabellensyntax stammt von PHP Markdown. Sie steht in identischer Form auch unter Pandoc (dort unter der Bezeichnung *pipe tables*) sowie in GitHub Flavored Markdown zur Verfügung.

Die Kennzeichnung der Spalten erfolgt durch das Zeichen |. Die Doppelpunkte in der Linie zwischen dem Spaltenkopf und den weiteren Zeilen geben an, ob der Spalteninhalt linksbündig (gilt per Default), rechtsbündig oder mittig ausgerichtet werden soll.

Die Syntax funktioniert selbst dann noch, wenn die Spalten nicht exakt übereinander ausgerichtet sind. Entscheidend sind ausschließlich die |-Zeichen zur Spaltentrennung.

```
| Rechts | Links | Mittig | Default |
|-----:|:-----:|:-----:|-----|
|      123 | abcdefg |      x | 1234 |
|      12 | fgh |     xxxx | 123 |
|      1 | i |      xx | 1 |
```

Rechts	Links	Mittig	Default
123	abcdefg	x	1234
12	fgh	xxxx	123
1	i	xx	1

Bei Pandoc können Sie die Tabelle mit `Table:` Beschriftung oder in der Kurzform `:` Beschriftung benennen.

```
| Rechts | Links | Mittig | Default |
|-----:|:-----:|:-----:|-----|
|      123 | abcdefg |      x | 1234 |
|      ... | | | |
```

Table: Eine Linientabelle

Rechts	Links	Mittig	Default
123	abcdefg	x	1234
...			

Tabelle 3.5: Eine Linientabelle

Einfache Tabellen (Pandoc)

Pandoc kennt neben den von PHP Markdown Extra übernommenen Linientabellen drei weitere Syntaxvarianten zur Darstellung von Tabellen: einfache Tabellen, Tabellen mit mehrzeiligen Einträgen sowie Gittertabellen.

Bei einfachen Tabellen wird die Tabellenstruktur nicht durch das Trennzeichen | vorgegeben, sondern durch die Begrenzungslinien ober- und unterhalb der Tabellenzeilen. Jede Zeile der Tabelle wird durch eine Markdown-Zeile abgebildet (ohne leere Zeilen dazwischen).

Die Ausrichtung der Spalten geht aus der Textausrichtung hervor. Die Beschriftung der Spalten sowie der gesamten Tabelle ist optional.

Deutsch	Englisch	Schwedisch	Hexadezimal
rot	red	röd	~ff0000~
grün	green	grön	~00ff00~
blau	blue	blå	~0000ff~
weiß	white	vit	~ffffff~
schwarz	black	svart	~000000~

Table: Farben in deutsch, englisch, schwedisch sowie hexadezimal

Deutsch	Englisch	Schwedisch	Hexadezimal
rot	red	röd	ff0000
grün	green	grön	00ff00
blau	blue	blå	0000ff
weiß	white	vit	ffffff
schwarz	black	svart	000000

Tabelle 3.6: Farben in deutsch, englisch, schwedisch sowie hexadezimal

Tabellen mit mehrzeiligen Einträgen (Pandoc)

Zur Formulierung längerer Tabelleneinträge reicht die Zeilenbreite irgendwann nicht mehr aus. Für solche Fälle sieht Pandoc eine Syntaxvariante für mehrzeilige Einträge vor, wobei die Zeilen der Tabelle jeweils durch Leerzeilen getrennt werden. Bei Tabellen ohne Kopfzeile bleibt die Syntax unverändert. Allerdings ist die Länge der Striche zur Kennzeichnung der Spalten nun ein prozentueller Maßstab für die Spaltenbreiten im HTML-Code.

```

-----
`-u u` oder `--user=u`   gibt den Benutzernamen an. Standardmäßig
                          gilt der aktuelle Benutzername
                          bzw. unter Windows `ODBC`.

`-h h` oder `--host=h`   gibt den Hostnamen des MySQL-Servers an
                          (standardmäßig *localhost*).

`-p` oder `--password`   zeigt eine Passwortabfrage an.
-----
    
```


-u u oder --user=u	gibt den Benutzernamen an. Standardmäßig gilt der aktuelle Benutzername bzw. unter Windows ODBC.
-h h oder --host=h	gibt den Hostnamen des MySQL-Servers an (standardmäßig <i>localhost</i>).
-p oder --password	zeigt eine Passwortabfrage an.

Wenn Sie die Spalten einer mehrzeiligen Tabelle beschriften möchten, müssen Sie oberhalb der Spaltenbeschriftung eine durchgängige Linie angeben.

```

-----
Option                Bedeutung
-----
`-u u` oder `--user=u`  gibt den Benutzernamen an. Standardmäßig
                        gilt der aktuelle Benutzername
                        bzw. unter Windows `ODBC`.

...
-----

```

Table: Mehrzeilige Tabelle mit Spaltenbeschriftung

Gittertabellen (Pandoc)

Bei *grid tables* werden die Spaltenformen durch vertikale und horizontale Linien nachgebildet. Das gelingt am besten mit einem Editor, der einen entsprechenden Tabellenmodus anbietet (z. B. *table-mode* in Emacs).

Im Vergleich zu den beiden anderen Tabellenvarianten sind bei Gittertabellen alle Zellen linksbündig ausgerichtet. Dafür kann jede Blockzelle beliebige Markdown-Elemente enthalten (z. B. mehrere Absätze, Aufzählungen, Einrückungen oder Listings); das hat bei meinen Tests allerdings nicht immer zuverlässig funktioniert. Nicht erlaubt sind Zellen über mehrere Zeilen oder Spalten.

```

+-----+-----+
| Kommando      | Funktion      |
+=====+=====+
| `create dbname` | erzeugt eine neue Datenbank. |
+-----+-----+
| `drop dbname`  | löscht eine vorhandene      |
|                | Datenbank unwiderruflich.  |
|                | * Punkt 1                  |
|                | * Punkt 2                  |
+-----+-----+

```

Table: Beispiel für eine Gittertabelle

Horizontale Linien

Das originale Markdown kennt zwar keine Tabellen, aber immerhin horizontale Linien. Dazu müssen in einer isolierten Zeile mindestens drei -, _ oder *-Zeichen angegeben werden. Das folgende Listing zeigt einige Syntaxvarianten, um eine horizontale Linie auszudrücken. Im resultierenden HTML-Ergebnis sieht die Linie in allen Fällen gleich aus.

```

---

* * * *

-----

- - - - -

```

3.12 Fußnoten

Das originale Markdown sieht keine Fußnoten vor. Einige Markdown-Dialekte bieten aber entsprechende Erweiterungen.

Eingebettete Fußnoten (Pandoc)

In den Text eingebettete Fußnoten haben die Form `^[Fußnote]`. Bei der HTML-Konvertierung wird der Fußnotentext durch eine hochgestellte, anklickbare Zahl ersetzt. Sämtliche Fußnoten werden in einem Block am Ende des Dokuments angeordnet.

```
Unix^[UNIX ist eine eingetragene Marke der Open Group.] wird in  
diesem Buch als Oberbegriff für diverse vom ursprünglichen Unix  
abgeleitete Betriebssysteme verwendet. Linux ist eine Unix-Variante,  
bei der aber der Quelltext frei verfügbar ist.
```

Unix¹ wird in diesem Buch als Oberbegriff für diverse vom ursprünglichen Unix abgeleitete Betriebssysteme verwendet. Linux ist eine Unix-Variante, bei der aber der Quelltext frei verfügbar ist.

1. UNIX ist eine eingetragene Marke der Open Group.

Referenzierte Fußnoten (Pandoc)

Alternativ besteht die Möglichkeit, mit `[^1label]` auf eine Fußnote zu verweisen, die an einem anderen Ort innerhalb des Markdown-Dokuments definiert ist. Die Fußnote selbst wird mit `[^1label]`: eingeleitet. Der Text darf über mehrere Absätze reichen und auch andere Markdown-Elemente (Aufzählungen, Listings etc.) enthalten. Damit klar ist, wie weit die Fußnote reicht, müssen alle Elemente der Fußnote um vier Zeichen eingerückt werden. Sämtliche Fußnoten werden am Ende des HTML-Dokuments angezeigt, unabhängig davon, an welcher Stelle sie im Markdown-Text definiert wurden.

Unix^[^unix] wird in diesem Buch als Oberbegriff für diverse vom ursprünglichen Unix abgeleitete Betriebssysteme verwendet. Linux ist eine Unix-Variante, bei der aber der Quelltext frei verfügbar ist.

[^unix]: UNIX ist eine eingetragene Marke der Open Group.

Häufig wird UNIX (in Großbuchstaben) zur Kennzeichnung von Unix-Systemen verwendet, die von der Open Group zertifiziert sind. Unix (in normaler Schreibweise) ist hingegen die übliche Bezeichnung für Unix-ähnliche Betriebssysteme ohne Zertifizierung.

3.13 HTML- und LaTeX-Code

HTML-Code

Was machen Sie, wenn Sie HTML-Funktionen nutzen möchten, die Markdown nicht unterstützt? Kein Problem, Sie können den erforderlichen HTML-Code einfach *direkt* in Ihr Markdown-Dokument einbetten. Diese Möglichkeit besteht in allen Markdown-Dialekten, auch im originalen Markdown von John Gruber.

HTML-Blockelemente (Absätze, Tabellen etc.) müssen von anderen Markdown-Elementen durch eine Leerzeile getrennt werden. Die Start- und End-Tags sollten zudem nicht eingerückt werden. Innerhalb des HTML-Blocks gilt die Markdown-Syntax nicht!

Normaler Markdown-Text.

```
<table>
  <tr><th>Spalte 1</th><th>Spalte 2</th></tr>
  <tr><td>Zeile 1</td><td>abc *efg* **hijk**</td></tr>
  <tr><td>Zeile 2</td><td>Drag&Drop, &euro;</td></tr>
  <tr><td>Zeile 3</td><td>a + b < c </td></tr>
</table>
```

Normaler Markdown-Text.

Spalte 1

Spalte 2

Zeile 1

abc *efg* **hijk**

Zeile 2

Drag & Drop, €

Zeile 3

a + b < c

HTML-Code ist nicht LaTeX/PDF-kompatibel!

Wenn Sie dieses E-Book in der EPUB- oder Mobi-Variante lesen, ist das obige Beispiel korrekt als Tabelle zu sehen. Wenn Sie hingegen die PDF-Variante des E-Books nutzen, bei der der Markdown-Code zuerst mit Pandoc in LaTeX-Code umgewandelt wurde, sehen Sie hier nur unformatierten Text. Pandoc hat die HTML-Tags entfernt, der restliche Text bleibt übrig.

Für Sonderzeichen gelten folgende Regeln:

- Die Zeichen < und > werden, wenn sie alleine stehen, korrekt durch < und > ersetzt. In HTML-Tags bleiben die Zeichen hingegen erhalten. Daher wird in der obigen Beispieltabelle a + b < c korrekt abgebildet.
- Auch über das &-Zeichen müssen Sie sich in der Regel keine Gedanken machen – Markdown ersetzt es automatisch durch &. Daher wird Drag,&,Drop in der obigen Tabelle wunschgemäß wiedergegeben. Wenn Markdown aber erkennt, dass Sie mit dem Zeichen & ein HTML-Sonderzeichen auszudrücken beabsichtigen (z. B. €), bleibt das Zeichen als solches erhalten. (Das gilt interessanterweise auch, wenn Pandoc aus dem Markdown-Text eine PDF-Variante oder LaTeX-Code generiert.)

HTML-Tags, die nur für Textpassagen gedacht sind (, <cite> oder) können auch *innerhalb* von Markdown-Text eingesetzt werden. In diesem Fall bleibt die Markdown-Syntax gültig:

Normaler Markdown-Text. ``Ab hier
in der EPUB/Mobi-Variante in roter Farbe.`` Fortsetzung.

Normaler Markdown-Text. Ab hier in der EPUB/Mobi-Variante in roter Farbe.
Fortsetzung.

Markdown-Formatierung innerhalb des HTML-Codes (Pandoc)

Was den Umgang mit eingebettetem HTML-Code betrifft, weicht Pandoc in einem Punkt von der originalen Markdown-Syntax ab: Markdown-Auszeichnungen können auch innerhalb von HTML-Blöcken verwendet werden. Wenn Sie das nicht möchten, müssen Sie die Erweiterung `markdown_in_html_blocks` explizit deaktivieren (Option `--from markdown-markdown_in_html_blocks`).

```
<table>
  <tr><td>Zeile 1</td>
    <td>Textformatierung: normal *kursiv* **fett**</td></tr>
  <tr><td>Zeile 2</td>
    <td>Links: [Wikipedia](https://de.wikipedia.org/)</td></tr>
  <tr><td>Zeile 3</td><td>Lorem ipsum ...</td></tr>
</table>
```

LaTeX-Formeln (Pandoc)

Pandoc wandelt Markdown-Dokumente auf Wunsch auch in das LaTeX-Format um. Damit Sie dabei auch LaTeX-Funktionen nutzen können, die Markdown nicht direkt unterstützt, kann TeX-Code (ab Pandoc 2.3) bzw. LaTeX-Code (bis Pandoc 2.2) ähnlich wie HTML-Code direkt weitergegeben werden. Weitere Tipps zur Umwandlung von Markdown-Dokumenten in das LaTeX-Format folgen im Kapitel [LaTeX und PDF](#).

Um mathematische Formeln in gewöhnlichen Markdown-Text einzubetten, stellen Sie die Formel wie unter TeX/LaTeX zwischen zwei `$`-Zeichen. Wichtig ist, dass dem ersten `$`-Zeichen unmittelbar das erste Zeichen der Formel zu folgen hat (kein Leerzeichen!) und

das zweite $\$$ -Zeichen direkt an das Ende des Ausdrucks gestellt werden muss (wieder ohne Leerzeichen). Beim LaTeX-Export wird der LaTeX-Code direkt weitergeleitet.

Die Formel $E = mc^2$ beschreibt den Zusammenhang zwischen Masse und Energie. Die Länge der Hypotenuse eines rechtwinkligen Dreiecks wird mit $c = \sqrt{a^2 + b^2}$ berechnet.

Die Formel $E = mc^2$ beschreibt den Zusammenhang zwischen Masse und Energie. Die Länge der Hypotenuse eines rechtwinkligen Dreiecks wird mit $c = \sqrt{a^2 + b^2}$ berechnet.

LaTeX-Formeln in HTML/EPUB-Dokumenten

Wenn Sie dieses E-Book in der PDF-Variante lesen, sind die Formeln im obigen Beispiel korrekt dargestellt. Wenn Sie sich hingegen für die EPUB- oder Mobi-Variante des E-Books entschieden haben, wird in Ihrem Reader die Pythagoras-Formel optisch unschön als LaTeX-Quelltext angezeigt. (Nur in ganz einfachen Fällen gelingt Pandoc die richtige Interpretation von LaTeX-Code sogar für HTML. Einsteins Formel wird deswegen korrekt angegeben, ebenso π für die Kreisteilungszahl π .)

Zur Lösung dieses Dilemmas stellt Pandoc gleich fünf (!) verschiedene Verfahren zur Auswahl, um LaTeX-Formeln in HTML- und EPUB-Formaten zu integrieren. Leider ist keines dieser Verfahren perfekt, funktioniert also gleichermaßen für alle gängigen Webbrowser und E-Book-Reader. Manche Verfahren basieren auf JavaScript, andere übertragen den Code der Formel zu einem externen Server, der ein Bild der Formel zurückliefert. Einen Überblick über alle fünf Varianten finden Sie in der Pandoc-Dokumentation:

<https://pandoc.org/MANUAL.html#math-rendering-in-html>

Einzelne stehende Formeln müssen Sie mit `\begin{displaymath}` oder `\begin{equation}` einleiten und entsprechend abschließen. Die Kurzschreibweise `\[formel \]` ist nicht zulässig. Beachten Sie, dass Pandoc derartigen Code als LaTeX-Code und *nicht* als Formel betrachtet. Wenn Sie den Markdown-Text in ein anderes Format als LaTeX/PDF umwandeln, verschwindet die Formel ersatzlos!

```
\begin{displaymath}
\sum_{n=0}^{1000} n^2
\end{displaymath}
```

$$\sum_{n=0}^{1000} n^2$$

Formeln LaTeX-Makros (Pandoc)

Eine Besonderheit von Pandoc, die es in vergleichbarer Form in anderen Markdown-Dialekten nicht gibt, ist der Umgang mit LaTeX-Makros: Pandoc versucht, innerhalb des Markdown-Texts neu definierte LaTeX-Kommandos bei der Konvertierung von Formeln innerhalb des Markdown-Texts in einem anderen Format als LaTeX auszuführen.

Das folgende Beispiel stammt direkt aus der Pandoc-Syntaxbeschreibung. Dort wird zuerst das LaTeX-Kommando `\tuple` definiert, das den an das Kommando übergebenen Parameter zwischen die Zeichen `<` und `>` stellt. Aus `$(\tuple{x, y, z})$` wird also `<x, y, z>`. Bemerkenswert ist, dass dies nicht nur bei der Übersetzung des Dokuments in das LaTeX-Format funktioniert, sondern auch für HTML-Ergebnisse.

```
\newcommand{\tuple}[1]{\langle #1 \rangle}
```

Normaler Text mit einer Formel: `$(\tuple{x, y, z})$`

Normaler Text mit einer Formel: `<x, y, z>`

Wenn Sie dieses Verhalten nicht wünschen, führen Sie Pandoc ohne die standardmäßig aktive Erweiterung `latex_macros` aus, also mit der Option `--from markdown-latex_macros`.

LaTeX-Code (Pandoc)

Neben mathematischen Formeln können Sie auch beliebigen LaTeX-Code direkt in den Markdown-Text eingeben. Pandoc erkennt LaTeX-Kommandos an der Syntax `\kommando{abc}`. Längere LaTeX-Passagen können mit `\begin{abc}` eingeleitet und mit `\end{abc}` abgeschlossen werden. Dazwischen dürfen sich keine leeren Zeilen befinden. Der LaTeX-Code wird beim LaTeX-Export unverändert weitergegeben, verschwindet aber beim

Export in alle anderen Formate und fehlt somit bei der HTML-Fassung Ihres Dokuments. Das gilt auch für die EPUB- und Mobi-Variante dieses E-Books, wo im folgenden Beispiel das in einem Rechteck eingerahmte Wort »LaTeX-Code« fehlt.

```
Gewöhnlicher Markdown-Text mit eingebettetem \fbox{LaTeX-Code}.
```

Gewöhnlicher Markdown-Text mit eingebettetem LaTeX-Code.

Wenn Sie mit Pandoc ein Buch verfassen, können Sie z. B. mit `\index{xxx}` Einträge für das Stichwortverzeichnis einbauen. Die mit LaTeX erzeugte PDF-Version Ihres Buchs wird dann ein Stichwortverzeichnis enthalten, die HTML- bzw. E-Book-Version (EPUB, Mobi) hingegen nicht.

HTML- und LaTeX-Code kombinieren (Pandoc)

Um die Ausgabeformate HTML und LaTeX optimal zu unterstützen, können Sie bestimmte Ausgaben parallel als HTML- und LaTeX-Code durchführen. Im folgenden Beispiel soll ein kurzer Text in einem rechteckigen Kästchen ausgegeben werden. Die LaTeX-Syntax hierfür lautet `\fbox{xxx}`. In HTML erreichen Sie denselben Effekt, indem Sie den Text zwischen ``-Tags mit einer speziellen CSS-Formatierung einschließen. Bei der Umwandlung in ein LaTeX-Format eliminiert Pandoc den HTML-Code, bei der Umwandlung in ein HTML-Format (inklusive EPUB) den LaTeX-Code. So wird das Kästchen in beiden Fällen korrekt angezeigt.

```
Gewöhnlicher Markdown-Text. \fbox{Text in einer Box.}
<span style="border:1px; border-style:solid;">Text in einer
Box.</span> Wieder normal weiter.
```

Gewöhnlicher Markdown-Text. Text in einer Box. Text in einer Box. Wieder normal weiter.

Für dieses E-Book habe ich die Darstellung der Beispiele folgendermaßen realisiert:

```
\samplestart<div class="sample">
```

Markdown-Beispielcode

```
</div>\sampleend
```

In der HTML-Fassung des E-Books habe ich durch CSS-Code für die Klasse `sample` die Beispiele besonders hervorgehoben. In der mit LaTeX erzeugten PDF-Fassung habe ich hingegen die selbst definierten LaTeX-Kommandos `\samplestart` und `-end` verwendet, um die Beispiele zu formatieren.

3.14 Weitere Pandoc-spezifische Funktionen

Alle in diesem Abschnitt beschriebenen Funktionen sind Markdown-Erweiterungen. Sie stehen also nur in Pandoc sowie in manchen anderen Markdown-Dialekten zur Verfügung, nicht aber im originalen Markdown.

Literaturangaben und Literaturverzeichnisse

Mitunter gewinnt man den Eindruck, dass in wissenschaftlichen Texten die korrekte Angabe von Quellen wichtiger ist als der eigentliche Inhalt. Da die ursprüngliche Markdown-Syntax nicht zum Verfassen wissenschaftlicher Arbeiten konzipiert worden ist, fehlt dort die Möglichkeit, systematisch Literaturangaben zu formulieren.

In Pandoc zitieren Sie in eckigen Klammern, wobei Sie mit dem `@`-Zeichen auf die eigentliche Quelle verweisen:

```
Der Himmel ist tagsüber meistens blau [siehe auch @arnold2003,
Seite 33-35], in den Nachtstunden dagegen schwarz [@huber2013;
@eder2012]. In den Morgen- und Abendstunden kann der Himmel
orange, rot oder purpur leuchten [siehe @weiss2007].
```

Damit die Quellenverweise aufgelöst werden können, müssen Sie beim Aufruf von Pandoc den Dateinamen einer Literaturdatenbank mit der Option `--bibliography` angeben. Die Literaturdatenbank müssen Sie mit eigenen Werkzeugen erstellen, wobei Pandoc mit einer ganzen Reihe gängiger Formate zurechtkommt (MODS, BibTeX, MEDLINE etc.). Tipps zur Verarbeitung von Quellenangaben finden Sie in der Pandoc-Dokumentation:

<https://pandoc.org/MANUAL.html#citation-rendering>

Dokument-Metadaten

In Pandoc können Sie am Beginn des Dokuments einen Titel-Block mit Metainformationen über das Dokument formulieren. Der Block kann drei Informationen umfassen: den Titel, den oder die Namen der Autoren sowie ein Datum. Im einfachsten Fall geben Sie die Metadaten in drei Zeilen an, die mit `%` beginnen und den Dokumenttitel, den Autor und das Datum enthalten.

```
% Markdown & Pandoc (2. Auflage)
% Michael Kofler
% 2018-10-15
```

Hier beginnt der eigentliche Markdown-Text.

Wenn Sie die Markdown-Datei nun mit `pandoc -s` in ein selbstständiges HTML-Dokument umwandeln, beginnt dieses mit folgenden Informationen:

```
...
<head>
  <meta name="author" content="Michael Kofler" />
  <meta name="dcterms.date" content="2018-10-15" />
  <title>Markdown & Pandoc (2. Auflage)</title>
  ...
</head>
```

Erzeugen Sie hingegen ein LaTeX-Dokument, enthält dieses die folgenden Zeilen:

```
\title{Markdown \& Pandoc (2. Auflage)}  
\author{Michael Kofler}  
\date{2018-10-15}
```

Sie können im Titelblock wie bereits erwähnt auch mehrere Autoren angeben; dazu trennen Sie diese durch Semikola oder Zeilenumbrüche. Pandoc erlaubt auch einen mehrzeiligen Titel; in diesem Fall darf den Folgezeilen aber kein %-Zeichen vorangestellt werden.

```
% Das ist ein außergewöhnlich langer Titel  
  für ein außergewöhnlich tolles Buch  
% Autor 1  
  Autor 2  
  Autor 3  
% 2018
```

Sollten Sie Pandoc dazu verwenden, um man-Seiten zu verfassen (also Hilfetexte für Unix/Linux-Systeme), geben runde Klammern im Titel die man-Gruppe an (z. B. LS(1) für den man-Text zum Kommando ls in der man-Gruppe 1). man-Texte sehen neben der Überschrift auch eine Kopf- und Fußzeile vor. Diese können ebenfalls im Pandoc-Titelblock angegeben werden. Die Kopfzeile beginnt nach den runden Klammern der man-Gruppe, die Fußzeile nach dem Zeichen |.

```
% überschrift(1) Kopfzeile | Fußzeile
```

Metadaten zwingend erforderlich

Je nachdem, welches Ausgabeformat bzw. welche Optionen Sie beim Aufruf von Pandoc verwenden, fordert Pandoc die Metadaten mit einer Warnung ein (»This document format requires a nonempty title element«) und verwendet als Defaulttitel den Dateinamen.

Dokument-Metadaten im YAML-Format

Pandoc bietet auch die Möglichkeit, die Metadaten in einer Datei im YAML-Format ([Wikipedia](#)) anzugeben. Dazu leiten Sie den Block mit den Metadaten innerhalb der Markdown-Datei mit drei Minuszeichen ein (---) und beenden ihn mit drei Punkten (...).

Diese Syntax hat zwei Vorteile: Erstens ist die Syntax klarer, weil alle Metadaten explizit benannt werden, und zweitens können Sie damit diverse andere (Template-)Variablen einstellen (siehe auch den Abschnitt [Templates](#)), die Pandoc dann bei der Verarbeitung der Markdown-Datei berücksichtigt:

```
Bis hierher normaler Markdown-Text. Jetzt folgt der
Metadaten-Block im YAML-Format.
```

```
---
title:  Markdown & Pandoc
author: Michael Kofler
abstract: >
  Dieses E-Book gibt
  eine Einführung ...
variablexy: 123
...
```

```
Ab hier wieder Markdown.
```

Wenn Sie eine Variable mit mehrzeiligem Text einstellen und dabei den Zeilenumbruch erhalten möchten, leiten Sie den Block mit | ein:

```
---
...
header-includes: |
  \usepackage[margins=raggedright]{floatrow}
  \usepackage{caption}
  \captionsetup{labelsep=colon,justification=justified,
    singlelinecheck=off}
  \renewcommand{\captionfont}{\small\it}
...
```

Tipp

Wenn Sie umfangreiche Veränderungen am LaTeX-Header vornehmen, ist es zumeist besser, die Kommandos in eine eigene Datei zu schreiben und diese beim Pandoc-Aufruf mit `-H headerdatei` zu berücksichtigen. Beachten Sie, dass die beiden Verfahren nicht kombinierbar sind. Sie müssen sich für eines der beiden Verfahren (also YAML-Metadaten oder externe Datei mit `-H`) entscheiden.

Welche zusätzlichen Metadaten bzw. Variablen Pandoc berücksichtigt, hängt davon ab, welches Ein- und Ausgabeformat zum Einsatz kommt. Das Attribut `abstract` wird beispielsweise vom EPUB-Writer ausgewertet.

Wenn Sie möchten, können Sie alle Metadaten auch in einer eigenen Datei speichern und diese zusätzlich an Pandoc übergeben, z. B.:

```
pandoc kap1.md kap2.md kap3.md meta.yaml -o .html
```

Weitere Details zu dieser Variante finden Sie im Pandoc-Handbuch:

<https://pandoc.org/MANUAL.html#metadata-blocks>

Import anderer Dateien

Leider sehen weder das originale Markdown noch Pandoc eine Möglichkeit vor, um weitere Textdateien einzulesen. Wenn längere Dokumente aus mehreren Dateien zusammengesetzt sind, müssen sämtliche Dateien an den Markdown-Konverter übergeben werden (z. B. `pandoc kap1.md kap2.md kap3.md -o ergebnis.html`).

Eine Ausnahme ist das LaTeX-Kommando `\input{filename}`, das von Pandoc ausgeführt wird – aber nur für das Zielformat PDF oder LaTeX.

Immerhin existieren fertige Filter-Scripts, um Dateien mit Programmcode oder auch ganze Markdown-Dateien in ein Markdown-Dokument zu integrieren. Die Verwendung eines derartigen Scripts zeige ich Ihnen im Abschnitt [Filter](#).

4 Das Pandoc-Kommando

Im einfachsten Fall rufen Sie das Pandoc-Kommando (also `pandoc`) wie folgt auf:

```
pandoc input.md -o output.html
```

Wenn Sie Einfluss darauf nehmen möchten, wie Pandoc die Eingabedatei verarbeitet und wie das Kommando die Ausgabedatei erzeugt, dann bedarf es weiterer Steuerungsmechanismen, die ich Ihnen in diesem Kapitel vorstelle:

- **Optionen:** Durch Optionen wie `--section-divs` steuern Sie Details des Verhaltens von Pandoc.
- **Erweiterungen:** Die Funktionsweise von Pandoc hängt davon ab, welche der unzähligen eingebauten Erweiterungen aktiv sind. Mit den Optionen `--to` und `--from` geben Sie nicht nur das Ein- und Ausgabeformat an, sondern können die Liste der in Pandoc vorgesehenen Default-Erweiterungen erweitern bzw. reduzieren.
- **Templates und Template-Variablen:** Beim Aufbau neuer Dokumente verwendet Pandoc Musterdateien (eben *Templates*), die eine Menge Variablen vorsehen. Einige Variablen sind durch Pandoc vorgegeben, andere können Sie mit der Option `-v varname=va1ue` oder durch Metadaten im YAML-Format einstellen. Pandoc-Profis erhalten noch mehr Konfigurationsmöglichkeiten, wenn sie vordefinierte Templates durch eigene ersetzen.
- **Filter:** Bei der Umwandlung vom Input- ins Output-Format stellt Pandoc das Dokument zwischenzeitlich in einer eigenen Datenstruktur dar. In dieser Phase können Sie das Dokument durch Filter (also eigene Scripts oder Programme) manipulieren. Selbst

wenn Sie sich nicht die Mühe machen möchten, eigene Filter zu programmieren, können Sie auf eine ganze Palette im Internet verfügbarer Filter zurückgreifen.

Leider gibt es dabei eine Hürde: Viele Filter setzen voraus, dass Sie außer Pandoc auch die Programmiersprache Haskell mit diversen Bibliotheken installiert haben.

Ich betrachte Pandoc in diesem Buch primär aus der Markdown-Perspektive, d. h. unter der Voraussetzung, dass das Quelldokument im Markdown-Format vorliegt. Beachten Sie aber, dass der Pandoc-Erfinder und -Entwickler John MacFarlane Pandoc nicht ausschließlich als Markdown-Tool betrachtet, sondern als universellen Dokument-Konverter!

4.1 Input- und Output-Formate

In der einfachsten Form wird Pandoc so aufgerufen:

```
pandoc input.md > output.html  
pandoc input.md -o output.html (gleichwertig)
```

Pandoc erzeugt damit aus einer Markdown-Datei die entsprechende HTML-Datei.

Neben Markdown und HTML unterstützt Pandoc unzählige weitere Ein- und Ausgabeformate, die mit den Optionen `-f` (oder `--from`) und `-t` (oder `--to`) angegeben werden, beispielsweise so:

```
pandoc -f markdown -t html      input.md      > output.html  
pandoc -f html      -t latex    input.html    > output.latex  
pandoc -f latex     -t markdown input.latex  > output.md
```

Oft können Sie auf die Optionen `-f` und `-t` verzichten. Pandoc bestimmt das gewünschte Format dann anhand der Dateikennung.

Die folgende Tabelle fasst in alphabetischer Reihenfolge zusammen, welche Formate Pandoc lesend und schreibend unterstützt (Option `-f` bzw. `-t formatname`):

Format	Formatname	lesen	schreiben
AsciiDoc	asciidoc		*
LaTeX Beamer	beamer		*
CommonMark Markdown	commonmark	*	*
ConTeXt	context		*
Creole	crealoe	*	
DocBook XML	docbook	*	*
DocBook 5	docbook5		*
DokuWiki	docuwiki		*
Word	docx	*	*
DZSlides HTML	dzslides		*
EPUB	epub2, epub3	*	*
FictionBook2	fb2		*
GitHub Flavored Markdown	gfm	*	*
Haddock Markup	haddock	*	*
LaTeX	latex	*	*
HTML	html	*	*
InDesign ICML	icml		*
JATS XML	jats		*
JSON AST	json	*	*
LaTeX	latex	*	*
man-Seiten	man		*
Markdown (Pandoc)	markdown	*	*
Markdown (MultiMarkdown)	markdown_mmd	*	*
Markdown (PHP Extra)	markdown_phpextra	*	*

Markdown (Original)	markdown_strict	*	*
MediaWiki	mediawiki	*	*
Groff	ms		*
Muse	muse	*	*
Haskell AST	native	*	*
LibreOffice Writer	odt	*	*
OpenDocument CMLS	opendocument		*
OPML	opml		*
Emacs Org-Mode	org	*	*
PDF	—		*
Plain Text	plain		*
Power Point	pptx		*
reStructuredText	rst	*	*
Rich Text Format	rtf		*
S5 HTML	s5		*
Slideous HTML	slideous		*
Slidy HTML	slidy		*
txt2tags	t2t	*	
TEI Simple	tei		*
Texinfo	texinfo		*
Textile	textile	*	*
TikiWiki	tikiwiki	*	
TWiki	twiki	*	
Vimwiki	vimwiki	*	

Tabelle 4.1: Von Pandoc unterstützte Formate

Es fehlt hier der Platz, auf alle Formate im Detail einzugehen, aber einige Anmerkungen sind doch angebracht.

- `native` ist ein Pandoc-internes Format (*Abstract Syntax Tree*, kurz AST) zur Darstellung formatierten Texts. Mit diesem Format müssen Sie sich auseinandersetzen, wenn Sie eigene [Filter](#) in der Programmiersprache Haskell programmieren möchten.
- Auch das Format `json` ist nur für Programmierer relevant. Es hat den gleichen Aufbau wie AST, verwendet allerdings die JSON-Syntax zur baumförmigen Darstellung des Dokuments.
- Für das Format PDF gibt es keinen Formatnamen, d. h., `-t pdf` ist nicht zulässig. Das liegt daran, dass PDF nicht unmittelbar unterstützt wird, sondern nur via LaTeX. Pandoc erzeugt also zuerst LaTeX-Quellcode und generiert daraus eine PDF-Datei (siehe auch das Kapitel [LaTeX und PDF](#)). Alternativ können Sie mit Pandoc auch ein LibreOffice-Dokument erzeugen und dieses dann manuell in eine PDF-Datei exportieren. Ein weiterer Weg besteht darin, ein HTML-Dokument zu erzeugen und dieses in einem Browser auszudrucken.

Ausführliche Informationen zum Erstellen von HTML-Seiten, LaTeX-Dokumenten, PDFs, E-Books (EPUB und Mobi) sowie Vortragsfolien (HTML und PDF) folgen in den weiteren Kapiteln dieses Buchs.

Einschränkungen

Erwarten Sie nicht, dass Pandoc alle Formate gleichermaßen perfekt unterstützt! Auf der Pandoc-Website gibt es zwar eine eindrucksvolle Sammlung von [Konvertierungsbeispielen](#), in der Praxis müssen Sie aber bei den meisten Formaten sowohl beim Lesen (Import) als auch beim Schreiben (Export) Abstriche machen. Dafür gibt es zwei Gründe:

- Zum einen sind manche Import- und Export-Module von Pandoc nicht ganz ausgereift. Die Pandoc-Dokumentation weist explizit darauf hin, dass die Import-Module für die Formate `html`, `latex`, `rst` und `textile` unvollständig sind.
- Zum anderen ist eine verlustfreie Umwandlung zwischen zwei beliebigen Formaten prinzipbedingt unmöglich. Viele der aufgelisteten Formate unterstützen Gestal-

tungsmöglichkeiten, die Pandoc nicht kennt, etwa frei platzierte Elemente (*floats*), mehrspaltigen Text, Marginalien oder Farben. Andere Formate kennen hingegen deutlich weniger Layoutfunktionen, z. B. keine für Bilder oder Tabellen.

Pandoc ist somit ein tolles Hilfsmittel im babylonischen Durcheinander der Textformate, kann aber natürlich keine Wunder vollbringen. Die besten Erfahrungen mit Pandoc werden Sie machen, wenn Sie das Programm primär zum Erstellen von HTML- und LaTeX-Dokumenten und als Ausgangsformat die Markdown-Syntax verwenden. Auch die meisten anderen Export-Formate funktionieren gut, sofern Sie mit den Einschränkungen des jeweiligen Formats vertraut sind.

Wenig begeistert hat mich der Import von LaTeX-Dokumenten. Einigermaßen passable Resultate werden Sie nur bei LaTeX-Dokumenten erzielen, deren Autoren sich auf ein Minimum der LaTeX-Möglichkeiten beschränkt haben. Je mehr Zusatzpakete zum Einsatz kommen, je mehr individueller LaTeX-Code verwendet wird, desto unbrauchbarer ist das Import-Ergebnis. Pandoc ist in dieser Hinsicht leider in guter Gesellschaft: Auch die diversen im Internet verfügbaren LaTeX-Konverter versagen bei der Umwandlung *realer* LaTeX-Quellen kläglich.

Markdown-Kompatibilität

Im Kapitel [Die Markdown-Syntax](#) bin ich schon auf die Syntaxunterschiede zwischen den verschiedenen Markdown-Dialekten eingegangen. Pandoc erwartet die Markdown-Dateien standardmäßig in einer stark erweiterten Markdown-Syntax. Darüber hinaus ist Pandoc mit der originalen Markdown-Syntax laut John Gruber sowie mit drei weiteren Markdown-Dialekten kompatibel. Um derartige Dateien zu verarbeiten, müssen Sie das Eingabeformat mit der Option `--from` explizit angeben:

- `--from markdown_strict`: originales Markdown
- `--from markdown_phpextra`: PHP Markdown Extra
- `--from markdown_mmd`: MultiMarkdown
- `--from gfm`: GitHub Flavored Markdown

4.2 Pandoc-Erweiterungen

Sämtliche Markdown-Erweiterungen sowie viele Spezialfunktionen sind in Pandoc in Form von Erweiterungsmodulen realisiert. Sie können beim Aufruf des Pandoc-Kommandos explizit angeben, welche Erweiterungsmodule abweichend vom Standardverhalten für ein bestimmtes Eingabeformat aktiviert (+) bzw. deaktiviert (-) werden sollen. Die allgemeine Syntax lautet:

```
name+a+b-c-d
```

Pandoc soll also das Format `name` verwenden, die Erweiterungen `a` und `b` zusätzlich aktivieren, dafür `c` und `d` deaktivieren.

Zwei Beispiele:

- Die Option `--from markdown_strict+footnotes` bedeutet, dass Pandoc eine Datei in der originalen Markdown-Syntax erwartet (`--from markdown_strict`), ergänzt lediglich durch die Pandoc-eigene Fußnoten-Syntax (`+footnotes`).
- Die Option `--from markdown-smart` bedeutet, dass Ihre Markdown-Datei grundsätzlich die Pandoc-spezifische Syntax verwendet (`--from markdown`), dass aber die üblicherweise aktive `smart`-Erweiterung nicht genutzt werden soll (`-smart`). Diese Erweiterung bewirkt unter anderem, dass Pandoc `--` als Gedankenstrich interpretiert.

Alle Pandoc-Erweiterungen sind hier dokumentiert:

<https://pandoc.org/MANUAL.html#extensions>

Wenn Sie wissen möchten, welche Erweiterungen standardmäßig aktiv sind, führen Sie `pandoc --list-extensions` aus:

```
pandoc --list-extensions
-abbreviations
+all_symbols_escapable
-amuse
-angle_brackets_escapable
-ascii_identifiers
+auto_identifiers
...
```

In Version 2.3 kennt das Programm 71 Extensions, von denen normalerweise 45 aktiv sind.

4.3 Pandoc-Optionen

Dieser Abschnitt fasst die wichtigsten Optionen für den Aufruf des Kommandos `pandoc` zusammen. Eine vollständige Referenz finden Sie im Pandoc-Handbuch:

<https://pandoc.org/MANUAL.html#options>

Details zu formatspezifischen Optionen folgen in den weiteren Kapiteln, die speziell auf die Erzeugung von HTML- und LaTeX-Dokumenten sowie auf das Erstellen von E-Books und Vortragsfolien eingehen.

Die allgemeine Syntax des Pandoc-Kommandos sieht folgendermaßen aus:

```
pandoc [optionen] [eingabedateien]
```

Die Eingabedateien werden der Reihe nach zu einem Ausgabedokument verarbeitet.

Anstelle von Eingabedateien sind auch Webadressen erlaubt, also z. B. `https://host.com/datei`.

Wenn keine Eingabedateien angegeben werden, erwartet `pandoc` den Markdown-Quelltext aus der Standardeingabe. Das resultierende Dokument wird an die Standardausgabe geschrieben. Es ist auch zulässig, diese Syntax durch die Zeichen `<` und `>` zu verdeutlichen.

```
pandoc [optionen] < input > output
```

Schließlich können Sie die Ausgabedatei mit der Option `-o` explizit festlegen. Bei binären Ausgabeformaten ist diese Option zwingend erforderlich.

```
pandoc [optionen] input -o output.pdf
```

Pandoc erkennt das Ein- und Ausgabeformat häufig anhand der Dateikennung. Gelingt dies nicht, müssen Sie das Format mit den Optionen `-f = --from` bzw. `-t = --to` explizit angeben.

Allgemeine Optionen

--data-dir=verzeichnis

gibt den Ort des Benutzerdatenverzeichnisses an. In diesem Verzeichnis können eigene Default-Layouts für verschiedene Formate gespeichert werden (z. B. CSS- oder Template-Dateien), um die resultierenden Dokumente abweichend von den Pandoc-Vorgaben zu gestalten. Wenn diese Option nicht verwendet wird, berücksichtigt Pandoc automatisch das Verzeichnis `.pandoc` (Linux, macOS) bzw. `C:\Users\\AppData\Roaming\pandoc`. Den bei Ihrer Installation gültigen Pfad können Sie mit `pandoc -v` ermitteln.

-D format

gibt die Template-Datei für das angegebene Format aus (siehe den Abschnitt [Templates](#)).

-f xxx oder --from=xxx

gibt das Ausgangsformat an, z. B. `latex`. Wenn die Option fehlt, versucht Pandoc das Format anhand der Dateikennung zu erraten. Der Formatname kann um Module erweitert werden, die Pandoc zusätzlich verwenden soll (+) bzw. die es nicht nutzen soll (-). Daraus ergibt sich dann z. B. der Formatname `-f markdown+mmd_title_block-implicit_figures` (siehe auch den Abschnitt [Markdown-Kompatibilität](#)).

-F programm oder --filter=programm

gibt ein Programm oder Script an, das die interne Darstellung des Dokuments verändert, nachdem Pandoc das Dokument gelesen hat, aber noch bevor es das Dokument schreibt. Der Filter erhält das Dokument in einem JSON-Format und muss es im selben Format wieder zurückgeben. Die Option kann mehrfach verwendet werden, um mehrere Filter hintereinander aufzurufen. Weitere Details folgen im Abschnitt [Filter](#).

-h oder --help

listet alle Optionen sowie alle Ein- und Ausgabeformate auf.

--lua-filter=programm

wie `--filter`, allerdings muss der Filter als Script in der Programmiersprache Lua vorliegen.

-o datei

gibt an, in welche Datei das Ergebnis geschrieben wird. Wenn diese Option fehlt, schreibt Pandoc die resultierende Textdatei in die Standardausgabe.

-t xxx oder --to=xxx

gibt das Zielformat an, z. B. epub. Standardmäßig erzeugt Pandoc HTML-Dateien.

-v oder --version

gibt die Pandoc-Version an, listet alle für die Syntaxhervorhebung unterstützten Sprachen auf und gibt den Ort des Benutzerdatenverzeichnisses an.

Allgemeine Reader- und Writer-Optionen

Die folgenden Optionen steuern die Verarbeitung der Eingabedateien bzw. beeinflussen die Ausgabe des resultierenden Dokuments durch Pandoc.

-A datei oder --include-after-body=datei

fügt die Datei bei HTML-Dateien vor `</body>` bzw. bei LaTeX-Dateien vor `\end{document}` ein. Die Option kann mehrfach verwendet werden und aktiviert `--standalone`.

-B datei oder --include-before-body=datei

fügt die Datei bei HTML-Dateien vor `<body>` bzw. bei LaTeX-Dateien vor dem Markdown-Text ein. Die Option kann mehrfach verwendet werden und aktiviert `--standalone`.

--base-header-level=n

gibt die Startnummer für Überschriften erster Ebene an (normalerweise 1).

--default-image-extension=xxx

gibt an, welche Kennung an die Dateinamen von Bildern hinzugefügt werden soll (z. B. png oder eps), wenn die Kennung fehlt.

-H datei oder --include-in-header=datei

fügt die Datei am Ende des HTML- oder LaTeX-Headers ein. Die Option kann mehrfach verwendet werden und eignet sich bei HTML-Dokumenten für JavaScript- und CSS-Dateien. Mit der Option wird automatisch auch `--standalone` aktiviert.

--highlight-style=xxx

legt den Stil für die Syntaxhervorhebung fest. Mögliche Einstellungen sind `pygments` (gilt standardmäßig), `kate`, `monochrome`, `espresso`, `zenburn`, `haddock` und `tango`. Auf der Pandoc-Webseite können Sie sich [Demos](#) ansehen, welche Farbmuster sich damit ergeben. Beispielsweise sind `espresso` und `zenburn` für einen dunklen Hintergrund optimiert.

--no-highlight

deaktiviert die Syntaxhervorhebung.

--number-sections

nummeriert Kapitel und Abschnitte bei allen Formaten, die dies unterstützen. Pandoc sieht keine Möglichkeit vor, die Anzahl der Ebenen zu steuern, die nummeriert werden sollen – also z. B. nur Kapitelüberschriften. Entweder werden alle Überschriften nummeriert, oder gar keine. Eigene Nummerierungskonzepte bei HTML-Dokumenten können Sie mit CSS realisieren – siehe auch den Abschnitt [Nummerierte Überschriften](#) im nächsten Kapitel.

--number-offset=n

gibt den Offset für die Kapitel-Nummerierung an (standardmäßig 0). Die Option aktiviert zugleich `--number-sections`. Mit `--number-offset=n,m,o` können die Offsets für drei Überschriftsebenen angegeben werden.

-s oder --standalone

erzeugt ein eigenständiges Dokument. Diese Option ist vor allem bei den Ausgabeformaten HTML, LaTeX und RTF relevant. Ohne diese Option erzeugt Pandoc Dateien ohne Header-Informationen; solche Dateien können nicht für sich verwendet werden, sondern müssen in ein anderes Dokument eingebettet werden.

--template=datei

verwendet die angegebene Datei als Vorlage. Die Template-Datei kann neben Layout-Vorgaben auch Anweisungen zur Strukturierung der Datei enthalten, z. B. zur Einbettung des Inhaltsverzeichnisses. Beispiele zur Verwendung von Template-Dateien folgen in den weiteren Kapiteln.

--toc

erzeugt ein Inhaltsverzeichnis.

--toc-depth=n

legt die Anzahl der Überschriftsebenen für das Inhaltsverzeichnis fest (standardmäßig 3).

-V key[=value] oder --variable=key[:value]

stellt eine Template-Variable ein.

Writer-Optionen für HTML

Die folgenden Optionen gelten nicht nur für das HTML-Format, sondern überwiegend auch für andere auf HTML basierende Formate. Dazu zählen epub2 und epub3 sowie alle auf HTML basierenden Folienformate, z. B. slidy, slideous, dzslides sowie s5.

--ascii

verwendet ausschließlich ASCII-Zeichen. Alle HTML-Zeichen mit Codes größer 127 werden durch HTML-Entities dargestellt (also z. B. ü statt ä).

-c datei oder --css=datei

gibt den Ort einer CSS-Datei an. Dabei kann auch eine Webadresse angegeben werden.

--email-obfuscation=javascript|references

verwendet JavaScript bzw. hexadezimal codierte Zeichen zur Darstellung von E-Mail-Adressen (zum Spam-Schutz).

--id-prefix=xxx

stattet alle HTML-Identifizierer mit dem angegebenen Präfix aus. Das ist dann

zweckmäßig, wenn Sie ein Dokument in mehreren Teilen übersetzen und dabei ID-Konflikte vermeiden möchten.

--self-contained

bettet alle externen Dateien (Bilder, Videos, JavaScript- und CSS-Dateien etc.) in Form von Daten-URLs ([Wikipedia](#)) direkt in den HTML-Code ein. Die resultierende HTML-Datei kann dann offline gelesen werden.

--section-divs

verpackt Abschnitte in `<section>`-Tags.

Writer-Optionen für LaTeX und PDF

--listings

verwendet das LaTeX-Paket `listings` zur Darstellung von Quellcode. Ohne diese Option verwendet Pandoc das `fancyvrb`-Paket.

--pdf-engine=lualatex|xelatex|wkhtmltopdf|weasyprint|prince|context|pdfroff

gibt an, wie PDF-Dokumente erzeugt werden sollen. Standardmäßig kommt PdfLaTeX zum Einsatz. Je nach Option müssen die entsprechenden Programme installiert werden, z. B. TeX Live, die HTML/PDF-Konverter `wkhtmltopdf` oder PrinceXML.

Writer-Optionen für Folien

-i oder --incremental

lässt Listenpunkte auf Folien Schritt für Schritt erscheinen. Ohne diese Option werden alle Punkte sofort angezeigt.

--slide-level=n

gibt an, bei welcher Überschriftenebene der Markdown-Text in Folien zerlegt wird. Pandoc versucht, die richtige Ebene anhand der Strukturierung des Inhalts selbstständig zu erkennen. Die Option ist nur erforderlich, wenn das nicht gelingt.

Writer-Optionen für E-Books im EPUB-Format

-c datei oder --css=datei

gibt den Ort einer CSS-Datei an. Dabei kann auch eine Webadresse angegeben werden. Wenn die Option fehlt, verwendet Pandoc `epub.css` im lokalen Verzeichnis.

--epub-chapter-level=n

gibt an, auf welcher Ebene das E-Book in Kapitel-Dateien zerlegt werden soll. Standardmäßig gilt $n=1$, d. h., die Trennung erfolgt bei Überschriften erster Ordnung. Mit $n=2$ oder $n=3$ wird das E-Book in kleinere Einheiten zerlegt.

--epub-cover-image=datei

verwendet die angegebene Datei als E-Book-Cover.

--epub-embed-font=datei

bettet die angegebene Schriftdatei in die EPUB-Datei. Die Option kann mehrfach verwendet werden, um mehrere Fonts einzubetten (siehe auch den Abschnitt [Eigene Schriften](#)).

--epub-metadata=datei

liest die E-Book-Metadaten (Titel, Verlag etc.) aus der angegebenen XML-Datei. Das Format der Datei muss dem [Dublin Core Metadata Element Set](#) entsprechen. Ohne diese Option verwendet Pandoc die [Metadaten](#) der Markdown-Datei.

4.4 Templates

Der Grundaufbau der durch Pandoc erzeugten Dokumente wird durch sogenannte Templates gesteuert. Diese Musterdateien werden zusammen mit Pandoc installiert und haben Namen wie `default.html` oder `default.epub3`. Mit `pandoc -D format > default.format` können Sie die gewünschte Template-Datei ansehen:

```
pandoc -D epub3 > default.epub3
```

Außerdem finden Sie die gerade aktuelle Version der Templates auf GitHub:

<https://github.com/jgm/pandoc-templates>

Zum Teil weichen die Template-Namen von den Formatnamen ab:

- **DOCX und PowerPoint:** Für diese Binärformate gibt es kein Template. Sie können aber mit `--reference-doc` eine Musterdatei im DOCX- bzw. im PPTX- oder POTX-Format angeben. Bei PowerPoint-Musterdateien sind die ersten vier Folien relevant, die als Muster für die Titel-Seite, für das Inhaltsverzeichnis, für eine Abschnittsüberschrift (*section header*) sowie für normale Folien gelten.
- **ODT:** Hier lautet der Template-Name `default.opendocument`. Zusätzlich kann mit `--reference-doc` eine Musterdatei im ODT-Format angegeben werden.
- **PDF:** Je nach Engine Template kommt das Template `default.latex` oder `default.context` oder `default.ms` oder `default.html` zum Einsatz.

Template-Syntax

Template-Dateien liegen einfach im Textformat vor und werden im Wesentlichen 1:1 übernommen. Ausgenommen sind Variablen, Bedingungen und Schleifen:

- **Variablen:** Variablen haben die Form `$variablename$`. Sie werden durch ihren Inhalt ersetzt.
- **Bedingungen:** Um Teile des Templates abhängig vom Inhalt einer Variablen zu machen, gibt es die Syntax `$if(variable)$ x $else$ y $endif$`. Sofern die angegebene Variable nicht den Wert Null enthält, wird der Text `x` aus der Schablone übernommen, andernfalls `y`. Wie in den meisten Programmiersprachen ist der `$else$`-Teil optional.
- **Schleifen:** Manche Variablen dürfen Wertelisten enthalten, z. B. `$author$`. Um alle Inhalte dieser Variable zu verarbeiten, bilden Sie die Schleife in der Form `$for(variable)$ x $variable$ y $endfor$`. Im Schleifenkörper wird anstelle von `$variable$` das gerade aktuelle Listenelement eingesetzt.

Die folgende Tabelle beschreibt die wichtigsten Template-Variablen. Eine vollständige Referenz gibt der [Pandoc User Guide](#).

Variable	Bedeutung
abstract	Zusammenfassung des Texts (optional)
author	der bzw. die Autoren des Dokuments (aus den Metadaten)
body	der eigentliche Inhalt des Texts
date	das Datum des Dokuments (aus den Metadaten)
headers-include	im Header zu inkludierende Dateien (Option -H / --include-in-header)
include-before	vor dem Text zu inkludierende Dateien (Option -B / --include-before)
include-after	nach dem Text zu inkludierende Dateien (Option -A / --include-after)
lang	die Sprache des Dokuments
subtitle	der Untertitel des Dokuments (optional)
title	der Titel des Dokuments (wird den Metadaten entnommen)
toc	das Inhaltsverzeichnis

Tabelle 4.2: Template-Variablen

Die meisten Variablen werden von Pandoc gesetzt – und zwar abhängig vom Ausgabeformat. Wenn Sie ein HTML-Dokument erzeugen, enthält `$body$` den HTML-Code zur Darstellung des Textinhalts, wenn Sie ein LaTeX-Dokument erzeugen, dementsprechenden LaTeX-Code.

Einigen Variablen können Sie mit der Option `-v varname=value` selbst einen Wert zuweisen. Deutlich übersichtlicher ist es oft, derartige Zuweisungen im Metadatenblock im YAML-Format durchzuführen (siehe auch den Abschnitt [Dokument-Metadaten im YAML-Format](#)).

Template-Beispiel

Die folgenden Zeilen zeigen eine gekürzte Fassung der Template-Datei `default.html` für HTML-Dokumente. Die Einrückungen der Zeilen entsprechen nicht dem tatsächlichen Inhalt der Datei, sondern wurden hier nur zur Verbesserung der Lesbarkeit durchgeführt.

Gleich in der zweiten Zeile wird die Variable `$lang$` ausgewertet. Wenn diese Variable nicht gesetzt ist, lautet die Ausgabe einfach `<html>`; hat die Variable dagegen den Inhalt `de`, enthält das resultierende HTML-Dokument die Zeile `<html lang="de">`.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      lang="$lang$" xml:lang="$lang$"
      $if(dir)$ dir="$dir"$endif$>
<head>
  <meta charset="utf-8" />
  ...
  $for(author-meta)$
    <meta name="author" content="$author-meta" />
  $endfor$
  ...
  <title>$if(title-prefix)$$title-prefix$ - $endif$$pagetitle$</title>
  ... (diverse CSS-Importe)
  $for(header-includes)$
    $header-includes$
  $endfor$
</head>
<body>
  $for(include-before)$
    $include-before$
  $endfor$
  $if(title)$
    ... (Titel, Subtitel, Autorenliste usw.)
  </header>
$endif$
$if(toc)$
  <nav id="$idprefix$TOC">
```

```
    $table-of-contents$  
  </nav>  
$endif$  
$body$  
$for(include-after)$  
  $include-after$  
$endfor$  
</body>  
</html>
```

Eigene Templates

Die Template-Syntax ist einfach zu verstehen. Wenn Sie den Aufbau für ein bestimmtes Dokumentformat verändern möchten, erstellen Sie einfach eine entsprechende Template-Datei. Dazu speichern Sie das aktuelle Template mit `pandoc -D format > default.format` in einer eigenen Datei und verändern diese mit einem Editor.

Damit Pandoc Ihre Template-Datei automatisch berücksichtigt, speichern Sie diese im Pandoc-Benutzerdatenverzeichnis, dessen Ort Sie mit `pandoc -v` ermitteln können (unter macOS und Linux einfach `.pandoc`). Wenn Sie eine veränderte Template-Datei nur für ein bestimmtes Projekt verwenden möchten, speichern Sie die Template-Datei im Projektverzeichnis und rufen Pandoc mit der Option `--template=datei` auf.

Vorsicht

Der Aufbau der Template-Dateien kann sich mit neuen Pandoc-Versionen ändern. In der Vergangenheit war das schon mehrfach der Fall. In solchen Fällen müssen Sie Ihre eigenen Änderungen und jene der neuen Pandoc-Version manuell zusammenführen, damit alle Features von Pandoc weiterhin funktionieren.

Um Probleme zu vermeiden, empfiehlt der Pandoc-Entwickler John MacFarlane, die [originalen Templates](#) mittels `git` in ein lokales Verzeichnis zu synchronisieren und die eigenen Änderungen nach Pandoc-Updates neuerlich mit den Originalen zusammenzuführen. Die Idee ist gut, aber leider nur für Pandoc-Anwender geeignet, die sich mit dem Versionsverwaltungssystem `git` auskennen.

4.5 Filter

Wenn Pandoc aus einer Input-Datei oder mehreren Input-Dateien eine Output-Datei erzeugt, generiert es zuerst eine vom Ein- und Ausgabeformat unabhängige interne Darstellung. Bevor die Output-Datei geschrieben wird, kann die Dokumentstruktur durch Filter verändert werden:

```
pandoc in.md --filter ./myscript -o out.html
pandoc in.md --lua-filter ./luascript -o out.html
```

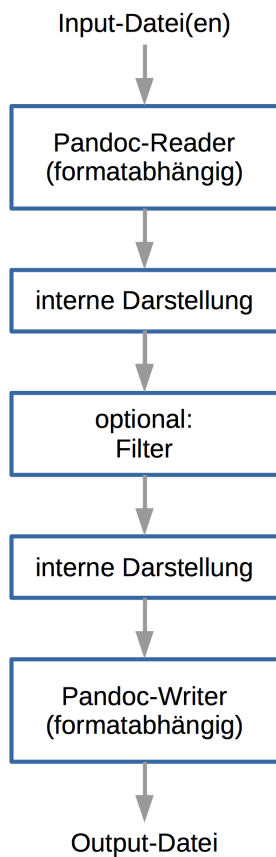


Abbildung 4.1: Filter in den Pandoc-Arbeitsablauf einbetten

Filter sind also Programme oder Scripts, die die Pandoc-interne Darstellung des Dokuments, den sogenannten *Abstract Syntax Tree* (AST), verarbeiten und manipulieren können.

In Haskell, also der Programmiersprache, in der Pandoc entwickelt wurde, ist für die Abbildung des AST das Modul `Text.Pandoc.Definition` zuständig. Es ist hier dokumentiert:

<http://hackage.haskell.org/package/pandoc-types>

AST-Aufbau

Der beste Weg, die AST-Logik zu begreifen, ist es, einfache Markdown-Dokumente in das Ausgabeformat `native` zu übersetzen. Das Kommando

```
pandoc in.md -o out.native
```

wandelt diese Markdown-Datei

```
# überschrift

Lorem ipsum dolor *sit amet,* consetetur
s adipscing **elitr, sed** diam nonumy
`eirmod tempor` invidunt ut

---

labore et dolore magna
aliquyam erat, sed diam
---

* voluptua. At
* vero eos
* et accusam et
```

in die folgende interne Darstellung um. (Ich habe einige Einrückungen vorgenommen und Leerzeichen bzw. leere Zeilen eingefügt, damit die Struktur besser erkennbar ist.)

```
[
  Header 1 ("\252berschrift", [], []) [Str "\220berschrift"],

  Para[Str "Lorem", Space, Str "ipsum", Space, Str "dolor",
    Space, Emph [Str "sit", Space, Str "amet,"], Space,
    Str "consetetur", Space, Str "sadipscing", Space,
    Strong [Str "elit", Space, Str "sed"], SoftBreak,
    Str "diam", Space, Str "nonummy", Space,
    Code ("", [], []) "eirmod tempor",
    Space, Str "invidunt", Space, Str "ut"],

  CodeBlock ("", [], [])
    "labore et dolore magna \nAliquam erat, sed diam ",

  BulletList[
    [Plain [Str "voluptua.", Space, Str "At"]],
    [Plain [Str "vero", Space, Str "eos"]],
    [Plain [Str "et", Space, Str "accusam", Space, Str "et"]]]
]
```

Traditionelle JSON-Filter

»Traditionelle« Pandoc-Filter, die mit der Option `-F` bzw. `--filter` aufgerufen werden, erhalten das Dokument freilich nicht direkt in der obigen AST-Darstellung, sondern in einer für Menschen noch unleserlichen JSON-Zeichenkette:

```
pandoc in.md -o out.json
```

Das folgende Listing ist wiederum neu formatiert und aus Platzgründen gekürzt:

```
{"blocks":[
  {"t":"Header", "c":[1,["überschrift", [], []],
    [{"t":"Str", "c":"überschrift"}]}],
  {"t":"Para", "c":[{"t":"Str", "c":"Lorem"},
    {"t":"Space"},
```

```
{ "t": "Str", "c": "ipsum" },
{ "t": "Space" },
{ "t": "Str", "c": "dolor" },
{ "t": "Space" },
{ "t": "Emph", "c": [ { "t": "Str", "c": "sit" },
                    { "t": "Space" },
                    { "t": "Str", "c": "amet," } ] },
... ] }, ... ],
"pandoc-api-version": [1,17,5,1],
"meta": {} }
```

Der Aufruf eines Filters mit

```
pandoc in.md --filter myfilter out.html
```

entspricht also:

```
pandoc in.md --to json | filter | pandoc --from json -o out.html
```

- Schritt 1: Pandoc verarbeitet die Input-Datei(en) und übergibt als Ergebnis eine JSON-Darstellung des AST an die Standardausgabe.
- Schritt 2: Dank des Pipe-Operators verarbeitet das Filter-Script die JSON-Daten aus der Standardeingabe und schreibt die veränderten JSON-Daten ebenfalls an die Standardausgabe.
- Schritt 3: Das zweite Pandoc-Kommando verarbeitet die Standardeingabe (zweites Pipe-Zeichen) und erzeugt das endgültige Dokument im gewünschten Ausgabeformat.

Traditionelle Filter werden oft in der Programmiersprache Haskell entwickelt. Grundsätzlich können Sie aber jede beliebige Sprache verwenden. Die einzige Voraussetzung besteht darin, dass Ihr Script das JSON-Dokument »versteht« und daraus wieder ein korrektes JSON-Dokument generiert. Für die Programmiersprachen Python, PHP, Perl, JavaScript/Node.js, Groovy und Ruby gibt es Bibliotheken, die dabei helfen.

Lua-Filter (ab Pandoc 2)

Das Verwandeln und Zurückverwandeln des AST in das JSON-Format kosten Zeit. Außerdem setzt der Aufruf eines externen Scripts voraus, dass die vom Script benötigte Programmiersprache samt allen Bibliotheken installiert ist (siehe auch den folgenden Abschnitt).

Um diese Nachteile zu umgehen, bietet Pandoc ab Version 2 die Möglichkeit, Scripts auch in der Sprache Lua ([Wikipedia](#)) zu formulieren. In Pandoc sind sowohl der Lua-Interpreter als auch alle Bibliotheken für den Lua-Zugriff auf den AST inkludiert. Lua-Filter sind deswegen portabler und effizienter als herkömmliche Filter.

Voraussetzungen und Einschränkungen

Viele für Pandoc erhältliche traditionelle Filter sind in der Programmiersprache Haskell entwickelt. Deren Ausführung setzt voraus, dass außer Pandoc auch Haskell installiert ist. Wenn Sie Pandoc als fertiges Paket installiert haben, ist das aber nicht der Fall!

Sie müssen vielmehr zuerst die Haskell-Plattform mit dem Paketverwaltungssystem `cabal` installieren und dann `cabal install pandoc` ausführen. Die erforderlichen Schritte für Ubuntu Linux habe ich im Abschnitt [Pandoc installieren](#) kurz zusammengefasst. Eine Anleitung, wie Sie die Haskell-Plattform unter Windows bzw. macOS installieren, finden Sie hier:

<https://www.haskell.org/platform/windows.html>

<https://www.haskell.org/platform/mac.html>

Ein generelles Problem von Filtern besteht darin, dass ihr Code mitunter abhängig von der Pandoc-Version ist. So gibt es im Internet diverse Filter, die unter Pandoc 1.n funktioniert haben, unter Pandoc 2.n aber zu Fehlern führen.

Fertige Filter

Ein Blick auf die AST-Darstellung eines Dokuments macht klar, dass die Programmierung eigener Filter kein Kinderspiel ist. Die Präferenz des Pandoc-Entwicklers John MacFarlane für exotische Programmiersprachen (Haskell, Lua) macht die Sache auch nicht gerade leichter. Eine Einführung in die Entwicklung eigener Filter geben die beiden folgenden Seiten:

<https://pandoc.org/filters.htm>

<https://pandoc.org/lua-filters.html>

Wenn Sie Glück haben, gibt es Filter, die genau Ihre Ansprüche erfüllen. Auf diesen Webseiten finden Sie Sammlungen fertiger Pandoc-Filter:

<https://github.com/jgm/pandoc/wiki/Pandoc-Filters>

<https://github.com/jgm/pandocfilters> (Filter in Python)

<https://github.com/pandoc/lua-filters> (Filter in Lua)

Beispiel 1: Code-Dateien einfügen

Oft wäre es praktisch, lange Listings nicht direkt in ein Markdown-Dokument zu übernehmen, sondern den betreffenden Code erst beim Erzeugen des Dokuments zu importieren. Die Markdown-Syntax bietet dazu leider keine Möglichkeit.

Im [Pandoc-Handbuch](#) finden Sie ein nur 10-zeiliges Script in der Programmiersprache Haskell, das genau diese Aufgabe übernimmt:

```
#!/usr/bin/env runhaskell
# Quelle https://pandoc.org/filters.html#include-files

import Text.Pandoc.JSON

doInclude :: Block -> IO Block
doInclude cb@(CodeBlock (id, classes, namevals) contents) =
  case lookup "include" namevals of
    Just f -> return . (CodeBlock (id, classes, namevals)) =<< readFile
      f
```

```
Nothing -> return cb
doInclude x = return x

main :: IO ()
main = toJSONFilter doInclude
```

Den Code des Scripts müssen Sie in einer lokalen Datei mit dem Namen `includes.hs` speichern. Unter macOS und Linux müssen Sie diese Datei auch ausführbar machen (`chmod +x includes.hs`). Wenn Sie nun in Ihrem Markdown-Dokument den folgenden Code einbauen

```
``` {.java include="HelloWorld.java"}
ersetzt diesen Text durch den Java-Code
```
```

und Pandoc auf diese Weise aufrufen

```
pandoc input.md -s --filter ./includes.hs -o output.html
```

dann wird an dieser Stelle `HelloWorld.java` als Listing eingebaut und ein Java-spezifisches Syntax-Highlighting durchgeführt.

Beispiel 2: Abkürzungen typografisch korrekt darstellen

Bei Abkürzungen wie »d. h.« sollte nach den typografischen Regeln zwischen »d.« und »h.« ein schmales, nicht trennbares Leerzeichen platziert werden. Der Abstand zwischen »d.« und »h.« soll also schmaler sein als ein normales Leerzeichen. Gleichzeitig muss ausgeschlossen werden, dass »d.« und »h.« beim Zeilenumbruch voneinander getrennt werden. Viele Autoren schreiben »d. h.« aber ganz einfach ohne Leerzeichen dazwischen.

Pandoc unterstützt das im Unicode-Standard für solche Fälle vorgesehe Zeichen U+202F (*narrow no-break space*). In HTML-Code bleibt dieses Zeichen erhalten, in LaTeX-Code wird es durch `\,` ersetzt.

Das folgende Mini-Script tauscht im gesamten Text (aber nicht in Listings bzw. in `code`-Passagen) die ohne trennendes Leerzeichen eingegebenen Abkürzungen durch eine entsprechende Zeichenkette mit dem schmalen, nicht trennbaren Leerzeichen aus.

```
function Str(elem)
  if string.find(elem.text, "d.h.") then
    return pandoc.Str(string.gsub(elem.text, "d.h.", "d.\u{202f}h.))
  else
    return elem
  end
end
```

Das Script ist die Adaption des Lua-Hello-World-Scripts im [Pandoc-Handbuch](#). Die Funktion `Str` wird auf alle gleichnamigen Elemente des AST angewendet.

Nachdem Sie das Script in einer Datei (z. B. `dh.lua`) gespeichert haben, kann es wie folgt verwendet werden:

```
pandoc in.md --lua-filter dh.lua -o out.html
```

Wesentlich schwieriger wäre es übrigens, im AST bei allen »d. h.«-Abkürzungen ein gewöhnliches Leerzeichen durch ein schmales, nicht trennbares Leerzeichen zu ersetzen. Das Problem besteht darin, dass sich »d.« und »h.« nun in zwei getrennten Elementen des AST befinden. Das Lua-Script müsste ganz anders formuliert werden, um »d. h.« in einer Sequenz aus je einem `Str`-, `Space`- und `Str`-Element zu entdecken.

5 HTML-Dokumente

Die häufigste Anwendung von Markdown besteht darin, HTML-Dokumente zu erstellen – von kurzen Blog-Beiträgen bis hin zu umfassenden Handbüchern. Im Prinzip sind alle dafür erforderlichen Grundlagen in den vorangegangenen beiden Kapiteln bereits zusammengefasst: Sie müssen also einen Text in der Markdown-Syntax verfassen und diesen dann mit einem Markdown-Konverter in das HTML-Format umwandeln. Ich gehe im Weiteren davon aus, dass Sie dazu Pandoc verwenden.

Dieses Kapitel fasst einige HTML-spezifische Tipps zusammen, die Ihnen dabei helfen, das Layout und die optische Gestaltung Ihrer Dokumente zu optimieren. Zum Verständnis dieses Kapitel ist es hilfreich, wenn Sie über ein Grundwissen von CSS (*Cascading Style Sheets*, [Wikipedia](#)) verfügen. CSS ist – vollkommen losgelöst von Markdown und Pandoc – die aktuelle Basis für die Gestaltung von HTML-Seiten. Eine CSS-Einführung ist im Rahmen dieses E-Books aber unmöglich.

5.1 Der Aufbau von HTML-Dokumenten

Grundsätzlich unterscheidet Pandoc zwischen zwei unterschiedlichen Varianten beim Erzeugen von HTML-Seiten:

- HTML-Fragmente ohne Header
- selbstständige HTML-Dokumente inklusive Header-Bereich, CSS-Links etc.

Standardmäßig, also mit `pandoc input.md > output.html` oder mit dem gleichwertigen Aufruf `pandoc input.md -o output.html`, liefert Pandoc eine unvollständige HTML-Datei. Die Datei beginnt direkt mit dem ersten Tag für einen Absatz, einer Überschrift oder womit immer Ihr Text beginnt. Derartige HTML-Dateien sind dafür gedacht, dass sie in eine

größere HTML-Seite eingebettet werden (z. B. in einem Blog oder Content Management System), die die fehlenden Header-Informationen und in der Regel auch CSS-Dateien für die Formatierung beisteuern.

Wenn Sie Pandoc hingegen mit der Option `-s` bzw. `--standalone` aufrufen, sieht das resultierende HTML-Dokument folgendermaßen aus:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="" xml:lang="">
<head>
  <meta charset="utf-8" />
  <meta name="generator" content="pandoc" />
  <meta name="viewport" content="width=device-width,
    initial-scale=1.0, user-scalable=yes" />
  <title>Dokumenttitel</title>
  <style type="text/css">
    code{white-space: pre-wrap;}
    span.smallcaps{font-variant: small-caps;}
    span.underline{text-decoration: underline;}
    div.column{display: inline-block;
      vertical-align: top; width: 50%;}
  </style>
</head>
<body>
<p>Der Inhalt Ihres Dokuments ...
</body>
</html>
```

Je nach Inhalt des Texts und je nachdem, welche Optionen Sie beim Aufruf von Pandoc verwendet haben, enthält der `<style>`-Abschnitt weitere CSS-Formatanweisungen, die beispielsweise zur korrekten Darstellung von Code-Listings oder für die Nummerierung von Überschriften (Pandoc-Option `--number-sections`) verantwortlich sind.

Hinweis

Der Aufbau des HTML-Dokuments wird durch Template-Dateien vorgegeben. Eine Einführung in die Syntax dieser Dateien sowie eine kurze Anleitung, wie Sie Templates selbst an eigene Wünsche anpassen können, finden Sie im Abschnitt [Templates](#).

Eigene CSS- und JavaScript-Dateien

Mit vier Optionen, die alle auch mehrfach verwendbar sind, können Sie in das resultierende HTML-Dokument eigene CSS-, JavaScript- und sonstige Dateien einbetten. Alle vier Optionen sind in der Referenz der Pandoc-Optionen im Abschnitt [Pandoc-Optionen](#) näher beschrieben.

- `-c datei` oder `--css=datei`
- `-H datei` oder `--include-in-header=datei`
- `-B datei` oder `--include-before-body=datei`
- `-A datei` oder `--include-after-body=datei`

Sonderzeichen und äöüß werden falsch dargestellt

Grundsätzlich erzeugt Pandoc UTF-8-codierte HTML-Dateien. Wenn in einem Webbrowser Sonderzeichen oder internationale Buchstaben wie ä, ö, ü oder ß falsch dargestellt werden, liegt dies immer daran, dass der Webbrowser den Zeichensatz nicht erkennt.

Eine mögliche Lösung des Problems ist die Pandoc-Option `-s`: Damit bettet Pandoc am Beginn der HTML-Datei den Header-Block mit den Zeichensatzinformationen ein. Die Option `-s` ist aber ungeeignet, wenn Sie das Pandoc-Ergebnis in einem Content Management System, auf einem Blog etc. nutzen möchten, also generell, wenn der von Pandoc erzeugte Text mit anderem HTML-Code verbunden wird.

Sollte Ihre Website *nicht* UTF-8, sondern eine andere Codierung verwenden, lösen Sie die Zeichensatzprobleme am besten mit der Option `--asci i`. Diese bewirkt, dass alle Zeichen mit Codes größer als 127 durch HTML-Entities dargestellt werden (also z. B. `ä`; statt ä). Das macht zwar die HTML-Datei größer als notwendig und den HTML-Code entsprechend unleserlich, geht aber allen Zeichensatzproblemen aus dem Weg.

5.2 Die Formatierung von HTML-Dokumenten

Grundsätzlich kümmert sich Pandoc nur um den Inhalt und die Strukturierung, aber nicht um die Formatierung Ihrer Dokumente. Eine Überschrift wird also z. B. zwischen `<h1>` und `</h1>` gestellt – aber es bleibt dem Webbrowser überlassen, in welcher Schrift, in welcher Schriftgröße, in welcher Farbe etc. die Überschrift dargestellt wird. Soviel sei vorweg gesagt: Besonders ansprechend wird Ihr Dokument nicht aussehen.

Abhilfe schafft eine CSS-Datei mit Formatvorgaben für alle im Dokument vorkommenden HTML-Elemente. Dort legen Sie z. B. fest, welche Grundschrift Ihr Dokument verwenden soll, wie Überschriften formatiert werden, wie Tabellen aussehen oder ob Code-Blöcke in graue Boxen gestellt werden sollen. CSS bietet hierfür nahezu unbegrenzte Formatierungsmöglichkeiten, weswegen zu diesem Thema schon viele Bücher verfasst wurden.

Eine simple CSS-Datei, die aus der hässlichen Defaultformatierung schon ein einigermaßen erträgliches Layout macht, kann z. B. so aussehen:

```
/* globale Default-Einstellungen */
body {
    padding: 0 1em;
    font-family: sans-serif;
    line-height: 140%;
}

/* überschriften */
h1 {
    color: #dd0000;
    font-size: 200%;
    margin: 3ex 0 2ex 0;
}
h2 {
    color: #dd0000;
    font-size: 120%;
    margin: 2ex 0 1ex 0;
}
```

```
/* Zitate */
blockquote {
  margin: 0 3em;
  font-style: italic;
}
```

Damit Pandoc einen Link auf die CSS-Datei in das HTML-Dokument einbaut, rufen Sie das Kommando folgendermaßen auf:

```
pandoc input.md -s -c mein.css > out.html
```

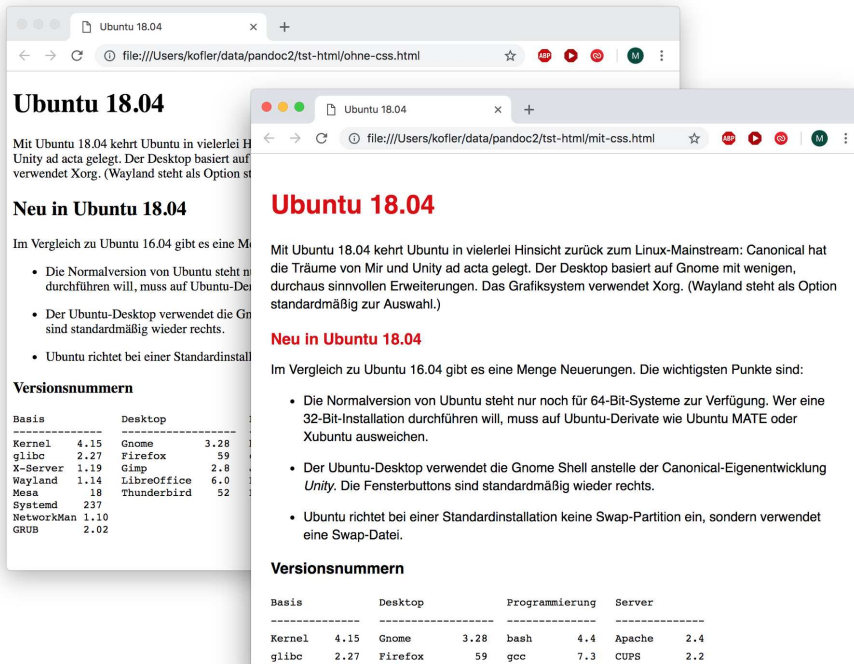


Abbildung 5.1: HTML-Dokument mit und ohne CSS-Formatierung

Die folgenden Abschnitte fassen einige Tipps zur Formatierung der wichtigsten HTML-Elemente zusammen.

Überschriften und Inhaltsverzeichnis

Überschriften werden in `<h1>`- bis `<h6>`-Tags verpackt. Das Aussehen der Überschriften kann dementsprechend durch CSS-Einstellungen für diese Tags verändert werden, wie das obige Einführungsbeispiel gezeigt hat.

Eine Pandoc-Eigenheit ist der Umgang mit Überschriften in Kombination mit der Option `--toc`. Diese Option bewirkt nicht nur, dass am Beginn des Dokuments ein Inhaltsverzeichnis eingebettet wird, sondern auch, dass nun alle Überschriften im Dokument als Links auf sich selbst ausgeführt werden. Anstelle von `<h1>text</h1>` produziert Pandoc nun den folgenden HTML-Code:

```
<h1 id="labeltext">überschrift</h1>
```

Das eigentliche Inhaltsverzeichnis wird standardmäßig am Beginn des Dokuments angezeigt. Es wird mit einem `<nav id="TOC">`-Tag eingeleitet und ist aus simplen ``-Listen zusammengesetzt:

```
...
<body>
<nav id="TOC">
<ul>
<li><a href="#kap-html">HTML-Dokumente</a><ul>
<li><a href="#der-aufbau-von-html-dokumenten">Der
    Aufbau von HTML-Dokumenten</a><ul>
...
</ul>
</li>
</ul>...
</nav>...
```

Um nur Überschriften bis zu einer bestimmten Ebene in das Inhaltsverzeichnis aufzunehmen, verwenden Sie die Option `--toc-depth n`.

Die Gestaltung des Inhaltsverzeichnisses können Sie durch CSS-Code beeinflussen. Mit den folgenden Zeilen erreichen Sie, dass das Inhaltsverzeichnis am rechten Rand mit verkleinerter Schrift dargestellt wird. (Falls Sie den eigentlichen Inhalt mit Linien

oder Boxen gestalten, müssen Sie dort zusätzlich die CSS-Anweisung `overflow:hidden`; einbauen, um Überlappungen mit dem Inhaltsverzeichnis zu vermeiden.)

```
nav {
  float:right;
  width:30%;
  padding: 0.2em;
  margin: 0.5em;
  border: 1px dotted gray;
  font-size: 80%;
}
```

Nummerierte Überschriften

Mit der Pandoc-Option `--number-sections` erreichen Sie, dass sämtliche Überschriften nummeriert werden: Überschriften der ersten Ebene mit 1, 2, 3, Überschriften der zweiten Ebene mit 2.1, 2.2, 2.3, Überschriften der dritten Ebene mit 2.2.1, 2.2.2, 2.2.3 etc. Bei meinen Tests trat dabei das Problem auf, dass die erste Kapitelüberschrift statt 1 die Nummer 2 hatte. Abhilfe schuf `--number-offset 0`.

Hinter den Kulissen kümmert sich Pandoc mit der Option `--number-sections` bei HTML-Dokumenten selbst um die Nummerierung und fügt die Kapitel- und Abschnittsnummern mit `` in den HTML-Code ein:

```
<h1 id="label1">
  <span class="header-section-number">1</span>
  überschrift</h1>
<h2 id="label2">
  <span class="header-section-number">1.1</span>
  Der Aufbau von HTML-Dokumenten</h2>
```

Somit können Sie durch CSS-Code für die Klasse `header-section-number` Einfluss auf die optische Gestaltung aller Nummern nehmen.

Individuelle Nummerierungskonzepte unterstützt Pandoc allerdings nicht. Es ist beispielsweise unmöglich, *Kapitel 1: xxx* für Überschriften der ersten Ordnung zu verwenden,

1.2 yyy für Überschrift der zweiten Ordnung und gar keine Nummerierung für Überschriften ab der dritten Ordnung.

Derartige Wünsche können Sie realisieren, wenn Sie auf die Option `--numbered-sections` verzichten und die Nummerierung per CSS-Code realisieren. Im folgenden Beispiel zeigt entsprechenden Code für die ersten beiden Überschriftsebenen. Der Selektor `h1:not(.title)` ist erforderlich, dass die H1-Überschrift für das Dokument als ganzes nicht mitgezählt wird. (Ohne `not` bekommt das erste Kapitel die Nummer 2.)

```
body {
    counter-reset: chapcnt;
    ...
}
h1::before {
    content: "Kapitel " counter(chapcnt) ": ";
}
h1:not(.title) {
    counter-increment: chapcnt;
    counter-reset: seccnt;
    ...
}
h2 {
    counter-increment: seccnt;
}
h2::before {
    content: counter(chapcnt) "." counter(secnt) "\00a0\00a0\00a0";
}
```

Leider ist die oben skizzierte Vorgehensweise inkompatibel mit der Option `--toc`, die ohne `--number-sections` ein Inhaltsverzeichnis ohne Nummern erstellt.

Tabellen

Handlungsbedarf besteht in aller Regel auch bei der Formatierung von Tabellen. Ein paar Zeilen CSS-Code reichen schon aus, um Tabellen aufzupeppen. Eine Web-Suche nach *html table css* liefert Ihnen hunderte weitere Gestaltungsideen.

```
table {
  border-collapse: collapse;
  border-width: 2px 0 2px 0;
  border-color: #888;
  border-style: solid;
}
th {
  padding: 0.5ex 1ex;
  border-width: 2px 0;
  border-color: #888;
  border-style: solid;
  background-color: #e0e0e0;
}
td {
  padding: 0.4ex 1ex;
  vertical-align: top;
}
tr:nth-child(even) {
  background-color: #f8f8f8;
}
tr:nth-child(odd) {
  background-color: #f0f0f0;
}
```

Falls Sie die Tabelle beschriften, baut Pandoc hierfür in den HTML-Code ein `<caption>`-Element ein. Die folgenden CSS-Zeilen kümmern sich darum, dass die Beschriftung unterhalb in einer etwas kleineren, kursiven Schrift erfolgt. Falls Sie nummerierte Tabellenunterschriften wünschen, gehen Sie einfach wie bei den Kapitelüberschriften vor.

```
caption {
  padding-top: 2px;
  margin-top: 1px;
  margin-bottom: 1ex;
  caption-side: bottom;
  text-align: left;
  font-size: 90%;
  font-style: italic;
}
```

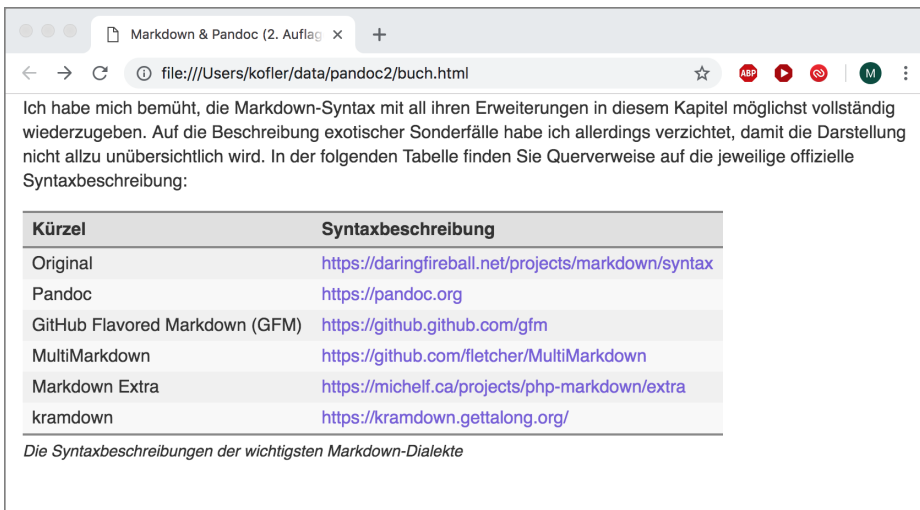


Abbildung 5.2: HTML-Tabelle mit CSS-Gestaltung

Abbildungen

Ein paar CSS-Zeilen bewirken, dass alle Bilder linksbündig im Dokument erscheinen und in kursiver, kleiner Schrift bezeichnet werden.

```
figure {
  align: left;
  margin-left: 0;
  margin-top: 3ex;
```

```
}  
figure figcaption {  
  font-size: 90%;  
  font-style: italic;  
}
```

Wenn Sie einen Blog-Beitrag mit überwiegend kleinen Bildern verfassen, ist es oft ansprechender, die Bilder mit `float` vom Fließtext umlaufen zu lassen. Überschriften stören dabei – `display: none;` eliminiert sie daher.

```
figure {  
  float: right;  
  margin: 0.5em;  
}  
figure figcaption {  
  display: none;  
}
```

6 E-Books im EPUB- und Moby-Format

Wenn von E-Book die Rede ist, sind zumeist Dateien in drei verschiedenen Formaten gemeint:

- PDF
- EPUB
- Moby

Dieses Kapitel zeigt, wie Sie mit Pandoc E-Books im freien EPUB-Format sowie im Amazon-spezifischen Moby-Format erzeugen können. Beide Formate basieren intern auf XHTML bzw. HTML. Dabei ist zu beachten, dass EPUB und Moby nur eine Teilmenge von HTML 5 und CSS unterstützen. Oft ist es zweckmäßig, für die E-Book-Variante eigene CSS-Dateien vorzusehen, die den Einschränkungen des E-Book-Formats gerecht werden. Empfehlenswert ist es auch, bereits von Beginn an EPUB-Versionen des Dokuments zu erzeugen, um allfällige Probleme gleich zu erkennen.

Tipps zum Erstellen von PDF-Dateien, unabhängig davon, ob diese nun als E-Books gedacht sind oder nicht, finden Sie im Kapitel [LaTeX](#). Der Grund für diese möglicherweise willkürlich erscheinende Trennung liegt darin, dass Pandoc hinter den Kulissen keinen direkten PDF-Export vorsieht. Der Weg zu einer PDF-Datei führt vielmehr über das LaTeX-Format und birgt einige Tücken.

6.1 EPUB

Das EPUB-Format ([Wikipedia](#)) ist ein offener Standard für E-Books. Das E-Book wird dabei aus einer Menge Einzeldaten zusammengesetzt: XML-Dateien mit Metadaten, XHTML-Dateien mit dem eigentlichen Buchinhalt, eine CSS-Datei für die Formatierung, Bilddateien für Abbildungen etc. Alle Dateien werden schließlich zu einem ZIP-Archiv verpackt. Eine EPUB-Datei ist letztlich also nichts anderes als eine ZIP-Datei, deren Inhalt bestimmten Regeln folgt.

EPUB-Reader und -Viewer

EPUB-Dateien können auf den meisten Smartphones und Tablets sowie auf vielen E-Book-Readern gelesen werden. Ein Problemkandidat für EPUB-Liebhaber sind aber ausgerechnet die zur Zeit populärsten E-Book-Reader, nämlich die Kindle-Geräte von Amazon: Diese können nur E-Books im Amazon-eigenen Mobi-Format darstellen.

Natürlich gibt es auch für herkömmliche PCs EPUB-Viewer. Unter macOS ist das Programm *iBooks* standardmäßig installiert. Für Windows-Rechner gibt es diverse kostenlose und kommerzielle Programme. Populär ist auch das Open-Source-Programm [Calibre](#): Es glänzt durch zahllose Funktionen, vor allem zur Verwaltung von E-Book-Sammlungen und zur Umwandlung von E-Books in unterschiedliche Formate. Weniger geglückt ist leider die Benutzeroberfläche. Calibre kann keine DRM-geschützten E-Books lesen.

Darstellungsabweichungen

Es ist leider nicht vorhersehbar, wie Ihr E-Book in verschiedenen E-Book-Readern aussehen wird. Zwar ist das EPUB-Format durch einen Standard sehr genau festgeschrieben, aber nicht jeder Reader erfüllt diesen Standard vollständig.

EPUB-Dateien mit Pandoc erzeugen

Grundsätzlich gibt es zwei Wege vom Markdown-Text zur EPUB-Datei:

- Wenn Sie mit Pandoc arbeiten, können Sie dessen EPUB-Export-Funktionen nutzen.
- Unabhängig vom Markdown-Dialekt besteht die Möglichkeit, zuerst eine HTML-Fassung Ihres Dokuments zu erzeugen und daraus dann eine EPUB-Datei mit einem HTML-zu-EPUB-Konverter zu generieren. Geeignete Programme sind das schon erwähnte Programm Calibre oder der EPUB-Editor [Sigil](#).

Dieses E-Book konzentriert sich primär auf die erste Variante, also das Erstellen von EPUBs direkt mit Pandoc. Im einfachsten Fall erzeugen Sie mit dem folgenden Kommando ein E-Book im Format EPUB 3:

```
pandoc kap1.md kap2.md kap3.md -t epub3 -o buch.epub
```

Darüber hinaus können Sie mit einigen EPUB-spezifischen Pandoc-Optionen eine CSS-Datei angeben, das Cover Ihres E-Books festlegen etc. Einen Überblick über die EPUB-spezifischen Optionen finden Sie im Abschnitt [Export-Optionen für E-Books](#), Anwendungsbeispiele in den folgenden Abschnitten.

Metadaten und Cover

Das EPUB-Format schreibt vor, dass jedes E-Book zumindest einen Titel aufweist. Fügen Sie also an den Beginn der ersten Markdown-Datei Ihres E-Books einen dreizeiligen Metadaten-Block ein, der den Titel, den oder die Autoren sowie das Erscheinungsdatum enthält (siehe auch den Abschnitt [Metadaten](#)):

```
% Buchtitel  
% Autor  
% Erscheinungsdatum
```

Außerdem sollte jede EPUB-Datei Copyright-Informationen sowie Angaben über die Sprache enthalten. Diese Angaben speichern Sie in einer XML-Datei, die dem [Dublin Core Metadata Element Set](#) entsprechen muss. Die Datei kann beispielsweise so aussehen:

```
<dc:rights>Copyright 2018 Michael Kofler</dc:rights>
<dc:description>Beschreibt die Markdown-Syntax und Pandoc</
  dc:description>
<dc:language>de</dc:language>
<dc:publisher>ebooks.kofler</dc:publisher>
<dc:identifier id="ISBN">1234567890123</dc:identifier>
```

Damit die Informationen aus dieser Datei berücksichtigt werden, geben Sie deren Namen beim Aufruf von Pandoc die Option `--epub-metadata` an.

Es ist üblich, jedes E-Book mit einem Titelbild (Cover) auszustatten. Die meisten E-Book-Programme erwarten, dass die Bildproportionen zwischen 3:4 und 2:3 betragen. Die Cover-Abbildung sollte im PNG- oder JPEG-Format vorliegen. Sie wird mit der Option `--epub-cover-image` an Pandoc übergeben.

Ein typisches Pandoc-Kommando zum Erstellen einer EPUB-Datei sieht damit so aus:

```
pandoc kap1.md kap2.md kap3.md -t epub3 \
  --epub-metadata mymeta.xml --epub-cover-image mycover.png \
  --css meine.css -o buch.epub
```

Blocksatz und Formatierung

Viele EPUB-Reader versuchen partout, jeden Text im Blocksatz darzustellen; dass die resultierenden Löcher zwischen den Wörtern hässlich aussehen und den Lesefluss massiv beeinträchtigen, scheint niemanden zu stören.

Die meisten EPUB-Reader (leider nicht alle) akzeptieren aber immerhin die CSS-Formatierung `text-align: left;`. Es sollte reichen, diese einfach für das `<body>`-Element anzugeben, nach meinen Erfahrungen ist es aber sicherer, die Anweisung explizit auch für `<p>` und `<blockquote>` anzugeben.

```
body, p, blockquote p {
    text-align: left;
}
```

Damit die EPUB-Datei diese und weitere CSS-Einstellungen berücksichtigt, geben Sie beim Aufruf von Pandoc die Option `--css meine.css` an. Diese Option kann auch wiederholt angegeben werden, um mehrere CSS-Dateien zu übergeben.

Wenn Sie sowohl eine HTML- als auch eine EPUB-Version Ihres Dokuments erstellen möchten, ist es sinnvoll, für beide Varianten je eine eigene CSS-Datei vorzubereiten. Die meisten CSS-Einstellungen werden für beide Formate identisch sein, aber fallweise ist es notwendig, die Darstellung speziell für das eine oder andere Format zu optimieren.

Sollten Sie keine eigene CSS-Datei angeben, kommt eine von Pandoc vorgegebene Defaultdatei zum Einsatz. Deren Code können Sie auf [GitHub](#) ansehen. Dieser Code ist gleichzeitig ein guter Ausgangspunkt für eigene Anpassungen.

Bilder und Listings

Abbildungen in EPUBs verhalten sich weitgehend gleich wie in HTML-Dokumenten. Zulässige Bildformate sind GIF, PNG, JPEG und SVG.

Code-Listings sind ein großes Problem für viele E-Book-Reader: Wenn die Display-Größe nicht ausreicht, um eine Codezeile vollständig darzustellen (und das kann durchaus schon bei 40 Zeichen der Fall sein), wird diese umbrochen. Das resultierende Listing ist damit zwar vollständig, aber praktisch unlesbar.

Eine vernünftige Lösung für dieses Problem gibt es nicht. Sie können die Häufigkeit hässlicher Listings mindern, indem Sie per CSS die Schriftgröße in Listingblöcken stark reduzieren. Alle Leser werden Sie damit natürlich nicht glücklich machen, aber persönlich ist mir ein Listing in kleiner Schrift immer noch lieber als eines, in der jede Zeile umbrochen ist.

```
code {
    font-family: monospace;
    font-size: 80%;
}
```


Inhaltsverzeichnis

Beim Erzeugen einer EPUB-Datei wird aus allen Überschriften des Dokuments automatisch ein Inhaltsverzeichnis generiert. Alle gängigen EPUB-Reader nutzen dieses Inhaltsverzeichnis und blenden es bei Bedarf ein (z. B. wenn ein spezieller Button gedrückt wird).

Wenn Sie möchten, können Sie dennoch auch für EPUBs die Option `--toc` verwenden. Sie erreichen damit, dass das Inhaltsverzeichnis nach der Titelseite in den EPUB-Text eingebettet wird. Aus unerfindlichen Gründen verwendet Pandoc in der XHTML-Seite `nav.xhtml` zur Strukturierung des Inhaltsverzeichnisses ``-Elemente. Die Überschriften werden daher nummeriert.

```
<nav epub:type="toc" id="toc">
  <h1 id="toc-title">Markdown & Pandoc (2. Auflage)</h1>
  <ol class="toc">
    <li id="toc-li-1">
      <a href="text/ch001.xhtml#impressum">
        Impressum</a>
    </li>
    <li id="toc-li-3">
      <a href="text/ch003.xhtml#intro">
        Einführung</a>
      <ol class="toc">
        <li id="toc-li-4">
          <a href="text/ch003.xhtml#hello-world">
            Hello World!</a>
        </li>
        ...
      </ol></li>
    </ol></li>
  </nav>
```

Bei Bedarf können Sie die Nummerierung der Einträge im Inhaltsverzeichnis per CSS-Code abstellen:

```
nav ol, nav ol ol, nav ol ol ol {  
  font-size: 85%;  
  list-style-type: none;  
  text-align: left;  
}
```

Mathematische Formeln und HTML-Code

Wenn Ihr Dokument mathematische Formeln enthält, ist es sinnvoll, die EPUB-Datei mit der Pandoc-Option `-t epub3` zu erstellen. Das resultierende E-Book enthält die Formeln dann in der MathML-Schreibweise. Moderne EPUB-Reader sind in der Lage, die Formeln richtig wiederzugeben (wenn auch selten so schön wie mit LaTeX).

Verzichten Sie auf diese Option, versucht Pandoc einfache Formeln direkt mit HTML-Code darzustellen. Bei komplizierteren Formeln wird hingegen der LaTeX-Quellcode im HTML-Dokument angezeigt. Das mag für LaTeX-kundige Leser korrekt sein, sieht aber natürlich nicht besonders schön aus.

Wenn Sie in Ihrem Markdown-Text direkt HTML-Code einbetten, müssen Sie akribisch darauf achten, dass Sie alle geöffneten HTML-Elemente wieder schließen. Vergessen Sie darauf, erzeugt Pandoc zwar ohne Fehlermeldungen eine EPUB-Datei; sobald diese jedoch in einem ePub-Leseprogramm geöffnet wird, treten aber Fehlermeldungen auf, und im schlimmsten Fall kann das E-Book überhaupt nicht gelesen werden.

Eigene Schriften

Das EPUB-Format erlaubt die Integration eigener Schriften in die EPUB-Datei. Dazu benötigen Sie zuerst Font-Dateien, die Sie weitergeben dürfen. Gut geeignet sind freie Schriften wie [DejaVu](#) oder [Source Sans Pro](#).

Die EPUB-Spezifikation erwartet Font-Dateien als OTF-Dateien. TTF-Dateien werden von vielen EPUB-Viewern ebenfalls auch akzeptiert, entsprechen aber nicht dem Standard. Falls Sie TTF-Dateien in das OTF-Format umwandeln möchten, können Sie dazu das Kommando `fontforge` verwenden. Unter Debian oder Ubuntu kann das Kommando

einfach mit `apt install fontforge` installiert werden. Unter macOS müssen Sie zuerst [Homebrew](#) einrichten. Danach führen Sie `brew install fontforge` aus.

Das folgende Shell-Script durchläuft alle *.ttf-Dateien, bildet zuerst den entsprechenden *.otf-Dateinamen und setzt dann ein Kommando über drei Zeilen zusammen. Dieses Kommando wird mit `-c` an `fontforge` übergeben.

```
#!/bin/bash
for ttfname in *.ttf; do
    otfname=${ttfname%.ttf}.otf;
    echo "$ttfname -> $otfname"
    cmd="import fontforge; from sys import argv; "
    cmd="$cmd f = fontforge.open('$ttfname'); "
    cmd="$cmd f.generate('$otfname')"
    fontforge -c "$cmd"
done
```

Der zweite Schritt besteht darin, die CSS-Datei so anzupassen, dass Ihre eigenen Font-Dateien genutzt werden. Das folgende Beispiel bezieht sich auf die DejaVu-Schriften:

```
@font-face {
    font-family: DejaVuSans;
    font-style: normal;
    font-weight: normal;
    src:url("../fonts/DejaVuSans.otf");
}
@font-face {
    font-family: DejaVuSans;
    font-style: normal;
    font-weight: bold;
    src:url("../fonts/DejaVuSans-Bold.otf");
}
@font-face {
    font-family: DejaVuSans;
    font-style: italic;
    font-weight: normal;
    src:url("../fonts/DejaVuSans-Oblique.otf");
}
```

```
}
@font-face {
  font-family: DejaVuSans;
  font-style: italic;
  font-weight: bold;
  src:url("../fonts/DejaVuSans-BoldOblique.otf");
}
@font-face {
  font-family: DejaVuSansMono;
  font-style: normal;
  font-weight: normal;
  src:url("../fonts/DejaVuSansMono.otf");
}
@font-face {
  font-family: DejaVuSansMono;
  font-style: normal;
  font-weight: bold;
  src:url("../fonts/DejaVuSansMono-Bold.otf");
}
body {
  font-family: "DejaVuSans";
}
code {
  font-family: DejaVuSansMono;
}
```

Die merkwürdigen Pfade (also ../font/name.otf) ergeben sich aus der Dateistruktur innerhalb eines EPUB-Dokuments. Die CSS-Datei landet nämlich im Unterverzeichnis `styles`, während die Font-Dateien im Unterverzeichnis `fonts` gespeichert werden.

Und zu guter Letzt müssen Sie für *jede* der eingesetzten Font-Dateien eine `--epub-embed-font`-Option beim Aufruf von Pandoc angeben. Da das Pandoc-Kommando nun recht umfangreich wird, empfiehlt es sich, dieses in eine kleine Script-Datei zu verpacken:

```
#!/bin/bash
pandoc vorwort.md kapitel1.md kapitel2.md kapitel3.md \
-t epub3 \
```

```
--css epub.css \  
--epub-metadata epubmeta.xml \  
--epub-cover-image cover.jpg \  
--epub-embed-font DejaVuSans-Bold.ttf \  
--epub-embed-font DejaVuSans-Oblique.ttf \  
--epub-embed-font DejaVuSansMono-Bold.ttf \  
--epub-embed-font DejaVuSans-BoldOblique.ttf \  
--epub-embed-font DejaVuSans.ttf \  
--epub-embed-font DejaVuSansMono.ttf \  
-o buch.epub
```

EPUB-Dateien validieren

Wenn Ihr EPUB-Reader (oder jener Ihrer Kunden) Probleme mit der richtigen Darstellung des von Pandoc erzeugten E-Books hat, kann das viele Ursachen haben – von mangelnder EPUB-Kompatibilität bis hin zu Fehlern, die Pandoc verursacht hat. Bei der Fehlersuche helfen Werkzeuge, die EPUB-Dateien validieren. Gut geeignet sind dafür das schon erwähnte Programm [Sigil](#) (Menükommando *Werkzeuge/Validiere EPUB*) oder das leider aktuell nicht mehr gewartete Open-Source-Programm [EpubCheck](#).

Bei [EpubCheck](#) handelt es sich um ein Java-Programm. Damit Sie das Programm ausführen können, muss auf Ihrem Rechner eine Java-Laufzeitumgebung installiert sein ([JRE-Download](#)). Anschließend laden Sie [EpubCheck](#) herunter und packen es aus.

Sie müssen das Programm in einem Terminalfenster starten, d. h., es gibt keine grafische Benutzeroberfläche. Die Positionsangaben der Fehlermeldungen geben die Zeile und die Spalte an.

```
java -jar ~/Downloads/epubcheck-4.0.2/epubcheck.jar buch.epub
```

Verwendung der EPUB 3.0.1 Prüfungen

```
ERROR(RSC-005): buch.epub/EPUB/text/ch005.xhtml(135,13):
```

```
Validierungsfehler: Element "blockquote" ist an dieser Stelle  
nicht erlaubt. ...
```

```
FATAL(RSC-016): buch.epub/EPUB/text/ch005.xhtml(186,3):
```

```
Schwerer Fehler beim Parsen der Datei 'Elementtyp "p"
```

```
muss mit dem entsprechenden Endtag "</p>" beendet werden.'.
FATAL(RSC-016): buch.epub/EPUB/text/ch009.xhtml(123,4):
  Schwerer Fehler beim Parsen der Datei 'Zeichenfolge "--" ist
  in Kommentaren nicht zulässig.'.
...
```

Oftmals werden Fehler durch eigenen, in den Markdown-Code eingebetteten HTML-Code verursacht. Achten Sie darauf, dass alle HTML-Tags wieder geschlossen werden müssen. Mitunter ist es erforderlich, HTML-Code und Markdown-Code durch je eine leere Zeile zu trennen.

Falsche Querverweise

Unbedingt auf den Grund gehen sollten Sie dem Fehler »Fragmentbezeichner ist nicht angegeben« (oder in englischer Sprache »fragment identifier is not defined«)! Diese Fehlermeldung bedeutet, dass es innerhalb Ihres E-Books einen ungültigen Link gibt, also beispielsweise einen Querverweis auf ein Kapitel, dessen Namen Sie geändert haben.

EPUB-Interna

Eine EPUB-Datei ist in Wirklichkeit einfach ein ZIP-Archiv. Wenn Sie den internen Aufbau einer EPUB-Datei ansehen möchten, etwa um einer Fehlermeldung von epubcheck auf den Grund zu gehen, zerlegen Sie das E-Book einfach mit einem ZIP-Programm in seine Einzeldateien. Diesen Vorgang sollten Sie in einem leeren Verzeichnis durchführen, damit Sie den Überblick über die Dateien behalten und nicht versehentlich Dateien überschreiben. Unter Linux bzw. macOS gehen Sie folgendermaßen vor:

```
mkdir mybook
cd mybook
unzip ../mybook.zip

Archive:  ../mybook.epub
  extracting: mimetype
   inflating: META-INF/container.xml
   inflating: META-INF/com.apple.ibooks.display-options.xml
   inflating: stylesheet.css
```

```

inflating: title_page.xhtml
inflating: content.opf
inflating: toc.ncx
inflating: nav.xhtml
inflating: images/img0.png
inflating: ch001.xhtml
    
```

Noch komfortabler erhalten Sie Zugriff auf die EPUB-Einzeldateien, wenn Sie das E-Book mit dem Programm Sigil öffnen. Dort können Sie die Dateien auch gleich verändern und eine neue Version des E-Books speichern. Das ist allerdings nur für allerletzte Korrekturen und Feineinstellungen zu empfehlen, weil alle derart durchgeführten Änderungen mit dem nächsten Pandoc-Aufruf verloren gehen.

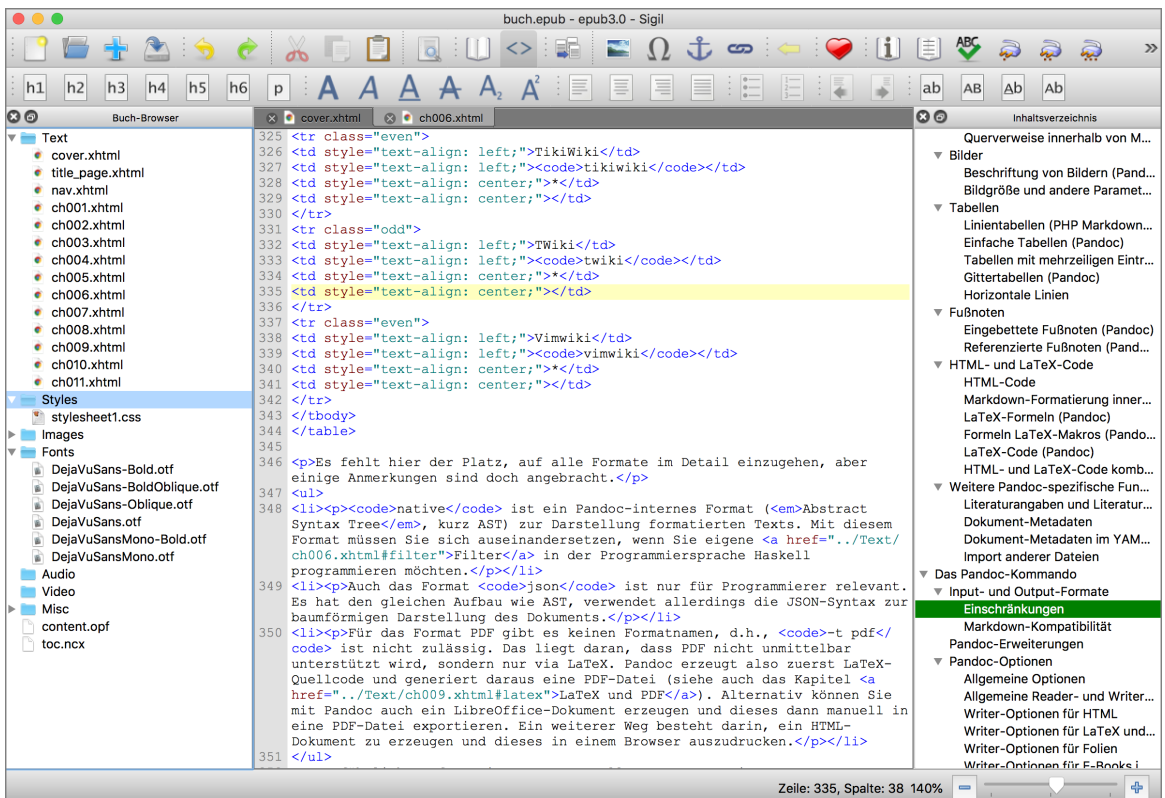


Abbildung 6.1: EPUB-Dateien mit Sigil bearbeiten

EPUB-Dateien aus HTML-Dokumenten erzeugen

Anstatt die EPUB-Datei direkt mit Pandoc zu erzeugen, können Sie auch einen anderen Weg beschreiten: Sie erzeugen zuerst eine HTML-Version Ihres Dokuments und wandeln diese dann mit anderen Werkzeugen in eine EPUB-Datei um.

Gut geeignet ist für diesen Zweck die grafische Benutzeroberfläche [Sigil](#). In diesem Programm erzeugen Sie zuerst mit *Datei/Neu* ein neues E-Book und fügen dann mit *Datei/Hinzufügen/Vorhandene Dateien* die betreffenden HTML-Datei(en) ein. Sigil importiert automatisch auch alle Bilder und CSS-Dateien, auf die die HTML-Dateien verweisen.

Mit *Werkzeuge/Inhaltsverzeichnis* erzeugen Sie dann das EPUB-Inhaltsverzeichnis. Bei langen E-Books ist es zudem zweckmäßig, das E-Book in einzelne Kapitel zu trennen. Dazu bewegen Sie den Cursor jeweils an den Beginn eines Kapitels und führen *Bearbeiten/Teilung am Cursor* durch.

Nach meinen Erfahrungen ist das manuelle Zusammenstellen von EPUBs allerdings wesentlich fehleranfälliger als Pandoc. Die Vorgehensweise empfiehlt sich nur dann, wenn Sie sehr gut mit den EPUB-Internas vertraut sind und Anforderungen an das E-Book stellen, die Pandoc nicht erfüllen kann.

6.2 Mobi

Amazon unterstützt das EPUB-Format leider nicht. Im Amazon-Kindle-Store können ausschließlich Mobi-Dateien verkauft werden, und auch die diversen Kindle-Lesegeräte erfordern Mobi-Dateien. Eine geschlossene Welt also, wie sie sich nur ein Marktführer leisten kann ...

Der interne Aufbau des Mobi-Formats hat Ähnlichkeiten mit dem EPUB-Format, im Detail gibt es aber viele Unterschiede, insbesondere was die Möglichkeiten zur Formatierung und zur Gestaltung des Layouts betrifft. Immerhin ist die aktuelle Mobi-Version nun weitgehend kompatibel zu HTML 5 und CSS 3, wenngleich viele Spezialfunktionen nicht unterstützt werden. Auch auf JavaScript sowie auf MathML zur Darstellung mathematischer Formeln müssen Sie verzichten. Im Detail können Sie die Spezifikation auf den Seiten [Kindle Format 8](#) und [Supported HTML5 / CSS3 Tags](#) sowie in den im PDF-Format vorliegenden, sehr umfassenden [Kindle Publishing Guidelines](#) nachlesen.

KindleGen

Pandoc unterstützt das Mobi-Format nicht. Um Mobi-E-Books zu erzeugen, müssen Sie auf das kostenlose Kommando `kindlegen` zurückgreifen. Dieses Programm laden Sie von der [Amazon-Website](#) herunter, wobei Versionen für Windows, macOS und Linux zur Auswahl stehen.

Ist dieses Programm einmal installiert, führen zwei Wege von Ihrem Markdown-Text zur Mobi-Datei:

- **EPUB -> Mobi:** Bei dieser Variante verwenden Sie Pandoc, um eine möglichst perfekte EPUB-Variante Ihres E-Books zu erzeugen. Daraus machen Sie dann mit einem simplen Aufruf von `kindlegen name.epub` die entsprechende Datei `name.mobi`. Die Metadaten der EPUB-Datei (Titel, Autor, Cover, Inhaltsverzeichnis etc.) werden automatisch in die Mobi-Datei übernommen. Wenn Sie also ohnedies auch eine EPUB-Datei benötigen, z. B. weil Sie Ihr E-Book in beiden Formaten verkaufen möchten, sparen Sie eine Menge Zeit.
- **HTML -> Mobi:** Ein zweiter Weg zur Mobi-Datei besteht darin, dass Sie mit Pandoc eine HTML-Datei Ihres gesamten Dokuments erstellen. Diese HTML-Datei verarbeiten Sie dann mit `kindlegen` zu einem Mobi-E-Book. Allerdings müssen Sie nun manuell die Dateien `name.opf` und `name.ncx` erstellen, bevor Sie die Mobi-Konvertierung durchführen können. Die OPF-Datei beschreibt im XML-Format die Eckdaten des E-Books. Die NCX-Datei enthält ein Inhaltsverzeichnis des Buchs, das in Kindle-Lesegeräten zur Navigation verwendet wird.

Empfehlung

In den folgenden Abschnitten folgen weitere Details zu beiden Varianten. Als ich vor fünf Jahren die erste Auflage dieses E-Books verfasste, ging meine Empfehlung zur HTML-Variante.

Mittlerweile ist es genau umgekehrt. Aktuelle Versionen von `kindlegen` (ich habe zuletzt mit Version 2.9 gearbeitet) verarbeiten EPUB-Dateien unkompliziert und weitgehend fehlerfrei. Der Weg über HTML ist nicht nur umständlicher (Sie müssen die Dateien `name.opf` und `name.ncx` verfassen), sondern verursacht häufig auch Probleme mit der Darstellung von Sonderzeichen, mit dem Layout (viele CSS-Einstellungen werden einfach ignoriert etc.).

Langer Rede kurzer Sinn: Erzeugen Sie zuerst eine möglichst perfekte EPUB-Datei und machen Sie dann daraus eine Mobi-Datei!

EPUB zu Mobi konvertieren

Wenn Sie aus Ihren Markdown-Quellen bereits eine funktionierende EPUB-Datei erzeugt haben, ist die Konvertierung in das Mobi-Format denkbar einfach: Das Kommando `kindlegen name.epub` erzeugt die entsprechende Datei `name.mobi`. Fertig!

Jetzt müssen Sie nur noch kontrollieren, ob das Layout des E-Books passt (siehe den Abschnitt [Vorschau](#)). Wenn Sie zufrieden sind, können Sie Ihr Mobi-E-Book nun über [Kindle Direct Publishing](#) auf der Amazon-Website zum Verkauf anbieten.

HTML zu Mobi konvertieren

Um eine Mobi-Datei aus der HTML-Version Ihres Dokuments zu erzeugen, benötigen Sie zumindest *vier* Dateien:

- die HTML-Datei Ihres Dokuments
- eine Bilddatei mit dem Cover Ihres E-Books
- eine OPF-Datei mit den Eckdaten Ihres E-Books (Titel, Autor etc.)
- eine NCX-Datei mit dem Inhaltsverzeichnis

Probleme mit Sonderzeichen

Die HTML-Datei muss mit `pandoc --ascii` erzeugt werden – andernfalls werden deutsche Sonderzeichen (äöüß) und andere Nicht-ASCII-Zeichen falsch dargestellt.

Falls die HTML-Datei auf CSS-Dateien oder Bilder verweist, müssen natürlich auch diese Dateien zur Verfügung stehen. Sind alle Vorbereitungsarbeiten abgeschlossen, rufen Sie `kindlegen` auf und übergeben die `*.opf`-Datei. Alle weiteren Dateien findet `kindlegen` aufgrund der Angaben in der `*.opf`-Datei.

```
kindlegen name.opf
```

Der erste Durchlauf von `kindlegen` endet häufig mit Fehlermeldungen: fehlende Angaben in der `*.opf`-Datei, falsche Querverweise in der `*.html`-Datei etc. Erst wenn all diese Probleme behoben sind, erzeugt `kindlegen` die Ergebnisdatei `name.mobi`.

Die folgenden Zeilen zeigen ein Muster für die Datei name.opf. Sie müssen auf jeden Fall die Metadaten am Beginn des Dokuments anpassen, außerdem die <item>-Elemente, die auf die restlichen Dateien verweisen (in diesem Beispiel buch.html, buch.ncx und cover.jpg).

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://www.idpf.org/2007/opf" version="2.0"
  unique-identifier="BookId">
  <metadata xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:opf="http://www.idpf.org/2007/opf">
    <meta name="cover" content="cover" />
    <dc:title>Buchtitel ...</dc:title>
    <dc:language>de</dc:language>
    <dc:identifier id="BookId" opf:scheme="ISBN">1234567890
      </dc:identifier>
    <dc:creator>Autor ...</dc:creator>
    <dc:publisher>Verlag ...</dc:publisher>
    <dc:subject>Thema ...</dc:subject>
    <dc:date>2018-12-31</dc:date>
    <dc:description>Kurze Zusammenfassung des
      Inhalts</dc:description>
  </metadata>

  <manifest>
    <!-- HTML Datei mit dem eigentlichen E-Book-Inhalt -->
    <item id="content" media-type="application/xhtml+xml"
      href="buch.html"></item>

    <!-- Inhaltsverzeichnis -->
    <item id="toc" media-type="application/x-dtbnex+xml"
      href="buch.ncx"/>

    <!-- Cover-Abbildung -->
    <item id="cover" media-type="image/gif" href="cover.jpg"/>
  </manifest>
```

```
<spine toc="toc">
  <itemref idref="toc"/>
</spine>

<!-- Der toc-Eintrag ist erforderlich, damit Kindle das
      Inhaltsverzeichnis findet. Der letzte type="text"-Eintrag
      definiert die Startseite des E-Books. -->
<guide>
  <reference type="toc" title="Inhaltsverzeichnis"
    href="buch.html#TOC"></reference>
  <reference type="text" title="Start"
    href="buch.html#impressum"></reference>
</guide>
</package>
```

Innerhalb der OPF-Datei müssen Sie auf eine NCX-Datei mit dem Inhaltsverzeichnis verweisen. Die folgenden Zeilen zeigen ein Beispiel für ein derartiges Inhaltsverzeichnis. Nach der Wiederholung der Titel- und Autorangaben müssen Sie für jeden Eintrag des Inhaltsverzeichnisses ein `<navPoint>`-Element definieren. Achten Sie darauf, dass es zu allen Querverweisen, die Sie hier verwenden (z. B. `src="buch.html#vorwort"`) entsprechende Sprungziele in Ihrem Dokument gibt. In Pandoc erreichen Sie das am einfachsten, indem Sie der betreffenden Überschrift `{#1abe1}` hinzufügen.

Das manuelle Erstellen der NCX-Datei ist natürlich mühsam. Ich beschränke mich deswegen normalerweise auf die Kapitelüberschriften und biete ein vollständiges Inhaltsverzeichnis in das Dokument ein (Pandoc-Option `--toc`). Dieses Inhaltsverzeichnis erscheint dann nur im Text, nicht als Kindle-Menü.

Grundsätzlich ist es auch möglich, die `<navPoint>`-Elemente zu verschachteln und so Abschnittsüberschriften in das Kindle-Inhaltsverzeichnis aufzunehmen. Schleierhaft bleibt, warum kindlegen die erforderlichen Daten nicht selbst aus den `<h1>`- und `<h2>`-Elementen des HTML-Texts extrahiert.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ncx PUBLIC "-//NISO//DTD ncx 2005-1//EN"
  "http://www.daisy.org/z3986/2005/ncx-2005-1.dtd">
```

```
<ncx xmlns="http://www.daisy.org/z3986/2005/ncx/"
  version="2005-1" xml:lang="en-US">
<head>
<meta name="dtb:uid" content="BookId"/>
<meta name="dtb:depth" content="2"/>
<meta name="dtb:totalPageCount" content="190"/>
<meta name="dtb:maxPageNumber" content="190"/>
</head>
<docTitle><text>Titel ...</text></docTitle>
<docAuthor><text>Autor ...</text></docAuthor>
  <navMap>
    <navPoint class="toc" id="toc" >
      <navLabel>
        <text>Impressum</text>
      </navLabel>
      <content src="buch.html#impressum"/>
    </navPoint>

    <navPoint class="toc" id="toc" >
      <navLabel>
        <text>Vorwort</text>
      </navLabel>
      <content src="buch.html#vorwort"/>
    </navPoint>

    <navPoint class="toc" id="toc" >
      <navLabel>
        <text>Einführung</text>
      </navLabel>
      <content src="buch.html#intro"/>
    </navPoint>

    ...

  </navMap>
</ncx>
```

Weitere Details zum Aufbau der OPF- und NCX-Dateien können Sie in den [Kindle Publishing Guidelines](#) nachlesen. Der Link *KF8 example* auf der [KindleGen-Downloadseite](#) führt zu einem ZIP-Archiv mit vollständigen Beispieldateien für ein kleines E-Book.

Vorschau

Um zu kontrollieren, ob die Mobi-Datei tatsächlich wunschgemäß aussieht, brauchen Sie den Kindle-Previewer. Auch dieses Programm stellt [Amazon](#) in einer Windows- und in einer macOS-Version kostenlos zur Verfügung. Linux-Fans benötigen zur Vorschau einen Windows-Rechner oder eine entsprechende virtuelle Maschine.

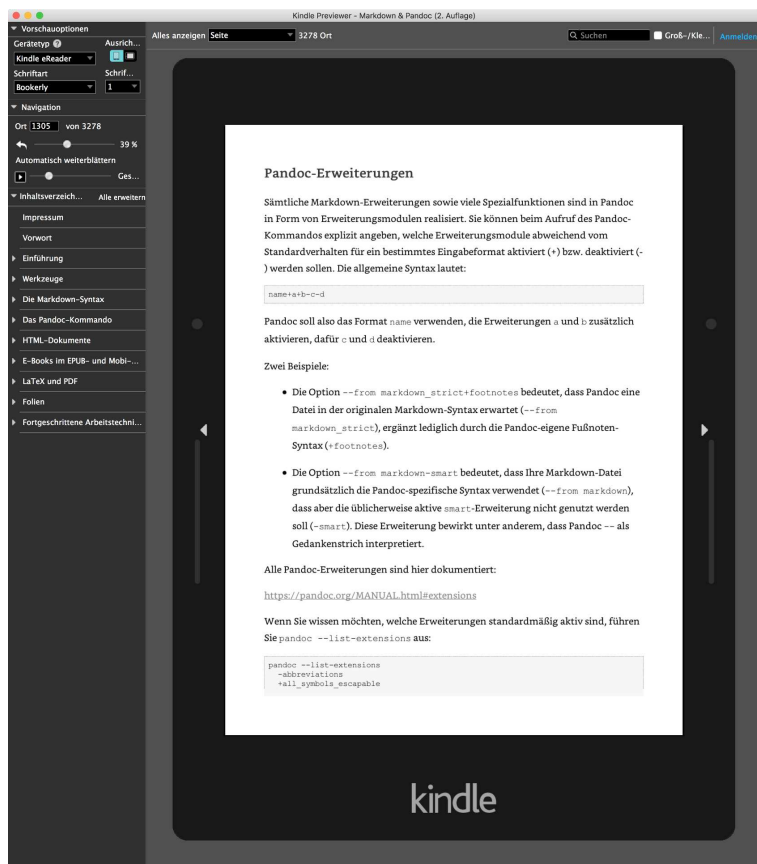


Abbildung 6.2: Vorschau von Mobi-Dateien mit dem Kindle-Previewer

Mobi-Internas ansehen

Während Sie den Inhalt einer EPUB-Datei wie bei einem ZIP-Archiv mühelos ansehen können, ist dies bei Mobi-Dateien etwas komplizierter. Glücklicherweise gibt es das Python-Programm [KindleUnpack](#), mit dem Sie alle in einer Mobi-Datei enthaltenen Dateien extrahieren können. Damit Sie das Programm ausführen können, müssen Sie auf Ihrem Rechner zuerst die Programmiersprache Python installieren. Anschließend wechseln Sie in das Download- bzw. Installationsverzeichnis und führen `./Mobi_Unpack.pyw` aus. In einer simplen Benutzeroberfläche können Sie nun die zu bearbeitende Mobi-Datei sowie ein leeres Verzeichnis auswählen, in dem die Dateien ausgepackt werden.

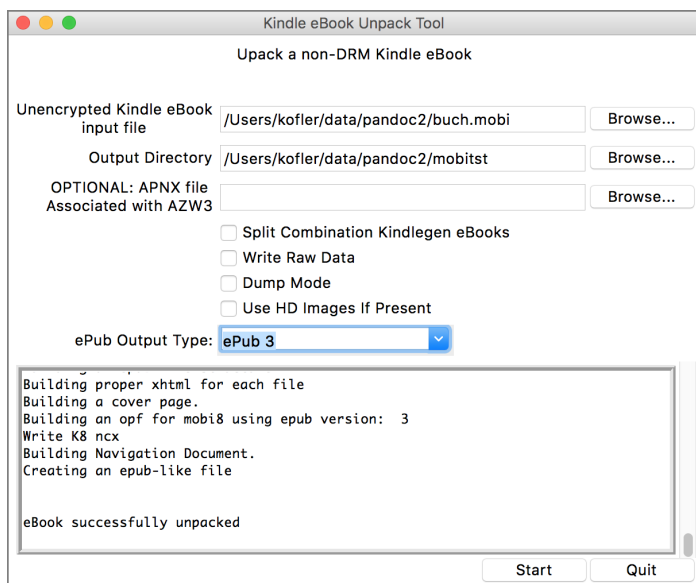


Abbildung 6.3: Das Mobi-Unpack-Tool

Ein Blick in das resultierende Verzeichnis verrät, dass `kindlegen` generell zwei Versionen des E-Books erzeugt, eine im `Mobi7`-Format für ältere Kindle-Geräte und eine zweite im `Mobi8`-Format für aktuelle Kindle-Geräte sowie für die diversen Kindle-Programme für Windows, macOS, iOS und Android. Die `Mobi7`-Version besteht primär aus einer `HTML`-Datei des E-Books und den dazugehörigen Bildern. Die `Mobi8`-Version ist hingegen kapitelweise aus `XHTML`-Dateien zusammengesetzt, samt `CSS`-Dateien zur Formatierung. Der gesamte Aufbau hat große Ähnlichkeiten mit einer `EPUB`-Datei. Falls Sie die

Mobi-Datei aus einer EPUB-Datei erzeugt haben, wird auch die zugrundeliegende EPUB-Datei in die Mobi-Datei eingebettet. Das ist auch der Grund, weswegen die Mobi-Datei circa doppelt so groß wie die EPUB-Datei ist.

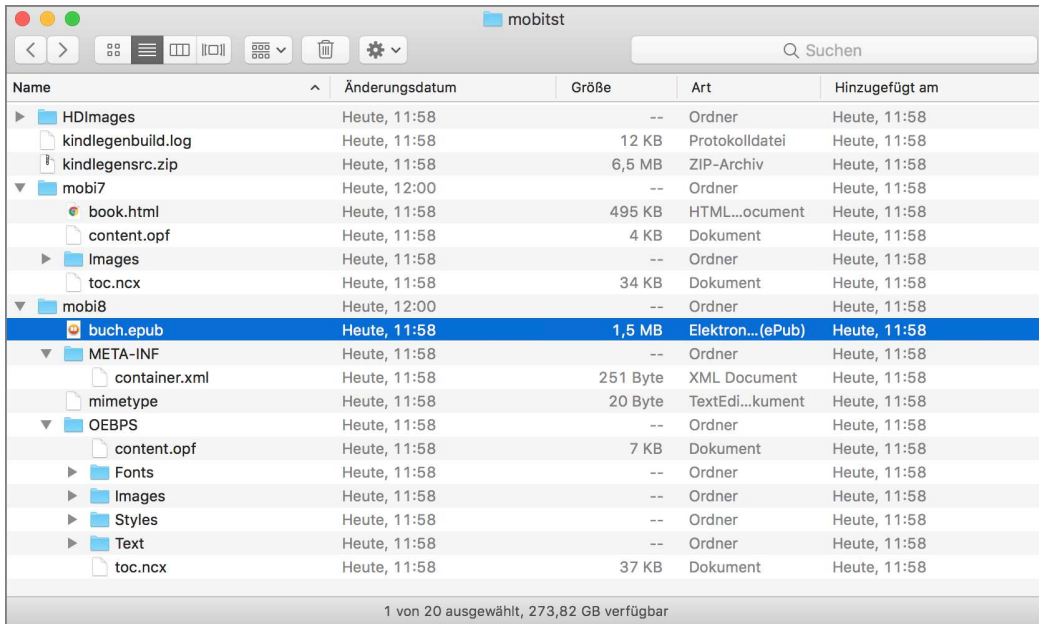


Abbildung 6.4: Der Inhalt eines Mobi-E-Books

7 LaTeX und PDF

In diesem Kapitel geht es darum, Markdown-Dokumente in LaTeX-Quelltext umzuwandeln. Anschließend können Sie daraus mit verschiedenen LaTeX-Kommandos PDF- oder PostScript-Dateien erzeugen. An sich funktioniert dies problemlos; allerdings ist es in der Praxis mit einiger Mühe verbunden, das resultierende Dokument den eigenen Vorstellungen entsprechend zu formatieren.

Einen ähnlich genialen Formatierungsmechanismus wie CSS-Dateien für HTML stellt LaTeX leider nicht zur Verfügung. Stattdessen erzielen Sie die gewünschte Formatierung durch spezielle Einstellungen und Zusatzpakete. Der Lohn dieser Arbeit sind ästhetisch gelungene E-Books bzw. Druckdateien, die geeignet sind, um daraus ein »echtes« Buch zu produzieren.

Ich setze in diesem Kapitel voraus, dass Sie über LaTeX-Grundkenntnisse verfügen und auf Ihrem Rechner eine vollständige LaTeX-Umgebung installiert haben:

- Unter Linux installieren Sie dazu die `texlive`-Pakete, die von nahezu allen Distributionen mitgeliefert werden. Wenn Sie mit Debian oder Ubuntu arbeiten, liefert das folgende Kommando einen guten Startpunkt.

```
sudo apt install texlive-latex-recommended \  
texlive-latex-extra texlive-fonts-recommended \  
texlive-font-utils texlive-lang-german lmodern
```

Wenn Sie exotische Fonts benötigen, können Sie außerdem das Paket `texlive-fonts-extra` installieren (Platzbedarf ca. 1 GByte!).

- Unter macOS installieren Sie die gut gewartete [MacTeX-Distribution](#). Dabei handelt es sich um eine Portierung von TeXLive, also derselben Programmsammlung, die auch unter Linux zum Einsatz kommt.
- Ein Installationsprogramm für TeXLive unter Windows finden Sie auf der [TUG-Website](#). Eine für Einsteiger einfachere Alternative ist [MiKTeX](#), eine speziell für Windows optimierte TeX-Distribution.

7.1 LaTeX- und PDF-Export

Das Kommando `pandoc in.md -t latex -o out.tex` liefert eine LaTeX-Datei, die anschließend mit [LaTeX](#), [pdfTeX](#), [LuaTeX](#) oder [XeTeX](#) weiterverarbeitet werden kann. Welche dieser LaTeX-Engines zum Einsatz kommt, bleibt Ihnen überlassen und hängt nur davon ab, ob Sie die resultierende `*.tex`-Datei mit den Kommandos `latex`, `pdflatex`, `luatex` oder `xelatex` weiterverarbeiten. Am populärsten ist momentan `pdflatex`, das aus dem von Pandoc erzeugten LaTeX-Quelltext eine PDF-Datei erzeugt. Auch `luatex` und `xelatex` produzieren PDF-Dateien, bieten aber einige Zusatzfunktionen und vereinfachen insbesondere die Verwendung anderer Schriften.

Damit Pandoc in die LaTeX-Datei alle erforderlichen Header-Anweisungen einbaut, so dass die Datei selbstständig verwendet werden kann, müssen Sie außerdem noch die Option `-s` angeben. Eine typische Kommandofolge, um aus einem Markdown-Text zuerst eine LaTeX-Datei und daraus ein PDF-Dokument zu erzeugen, sieht so aus. Die resultierende PDF-Datei hat dann den Namen `out.pdf`.

```
pandoc in.md -s -t latex -o out.tex
pdflatex out.tex
```

Um wirklich eine ästhetisch ansprechende PDF-Datei zu erhalten, müssen Sie entweder zusätzliche LaTeX-Kommandos in den Markdown-Text einbauen (etwa `\newpage`, um an einer Stelle einen Seitenumbruch zu erzwingen), oder die von Pandoc erzeugte LaTeX-Datei manuell verbessern. Der zweite Ansatz hat den offensichtlichen Nachteil, dass

alle Änderungen mit einem neuen Pandoc-Durchlauf verloren gehen; dafür bleibt der Markdown-Quelltext frei von unübersichtlichem LaTeX-Code.

Wenn Sie zwischen dem Aufruf von `pandoc` und `pdflatex` keine Änderungen an der `*.tex`-Datei durchführen, können Sie die Erzeugung der PDF-Datei auch in einem einzigen Schritt erledigen. Dazu geben Sie einfach mit `-o` an, dass Sie eine PDF-Datei erzeugen möchten. Pandoc erzeugt dann eine LaTeX-Datei (es gilt implizit die Option `-s` für Standalone-LaTeX-Dateien) und ruft anschließend selbst `pdflatex` auf. Beachten Sie, dass die Option `-t pdf` bzw. `--to pdf` nicht zulässig ist! (PDF-Dateien können auf unterschiedliche Arten erzeugt werden. Deswegen gilt PDF nicht als eigenständiges Ausgabeformat.)

```
pandoc in.md -o out.pdf
```

PDF-Engines

Wenn Sie möchten, dass Pandoc die PDF-Datei nicht mit `pdflatex`, sondern stattdessen mit `lualatex` oder `xellatex` erzeugt, geben Sie mit der Option `--latex-engine` die gewünschte LaTeX-Engine an. Diese Option hat keinen Einfluss auf den von Pandoc erzeugten LaTeX-Code; dieser bleibt immer gleich. Die LuaLaTeX- bzw. XeLaTeX-spezifischen Code-teile sind *immer* enthalten, werden aber nur ausgewertet, wenn zur Weiterverarbeitung die entsprechende Engine verwendet wird.

```
pandoc --latex-engine lualatex in.md -o out-lua.pdf  
pandoc --latex-engine xellatex in.md -o out-xe.pdf
```

Sie werden zwischen den drei PDF-Varianten keinen spürbaren optischen Unterschied feststellen. Sinnvoll ist die Differenzierung zwischen den Engines nur, wenn Sie mit weiteren Optionen (z. B. mit `--include-before-body`) zusätzlichen Formatierungscode einbetten möchten. Dieser Code kann dann auf die spezifischen Erweiterungen von LuaTeX oder XeTeX zurückgreifen.

Hinweis

Pandoc unterstützt mit der Option `-t context` bzw. `--latex-engine context` auch die LaTeX-Alternative [ConTeXt](#). ConTeXt basiert ebenso wie LaTeX auf TeX, verwendet aber andere vordefinierte Makros und ein anderes Erweiterungssystem. ConTeXt-Dateien sind somit nicht LaTeX-kompatibel und müssen mit dem Kommando `context` weiterverarbeitet werden. Ich gehe in diesem Kapitel auf ConTeXt nicht weiter ein.

Des Weiteren können Sie mit den Optionen `--pdf-engine wkhtmltopdf` bzw. `--pdf-engine prince` PDFs auch aus HTML-Dateien erzeugen. Damit das funktioniert, müssen Sie vorher [wkhtmltopdf](#) oder [Prince](#) (kostenpflichtig) installieren.

LaTeX-spezifische Pandoc-Optionen

Die folgende Tabelle fasst in Kürze die wichtigsten LaTeX-spezifischen Optionen des `pandoc`-Kommandos zusammen. Eine detaillierte Beschreibung gibt der Abschnitt [Pandoc-Optionen](#), Anwendungsbeispiele finden Sie in den folgenden Abschnitten.

Option	Bedeutung
<code>-A datei</code>	fügt die Datei nach <code>\end{document}</code> ein.
<code>-B datei</code>	fügt die Datei vor <code>\begin{document}</code> ein.
<code>--listings</code>	verwendet das <code>listings</code> -Paket anstelle von <code>fancyvrb</code> .
<code>--number-sections</code>	nummeriert Kapitel und Abschnitte.
<code>-s</code>	erzeugt ein eigenständiges LaTeX-Dokument.
<code>-S</code> oder <code>--smart</code>	verwendet typographisch korrekte Satzzeichen.
<code>--toc</code>	erzeugt ein Inhaltsverzeichnis.
<code>--toc-depth=n</code>	legt die Anzahl der Ebenen für das Inhaltsverzeichnis fest.
<code>-V lang=de</code>	optimiert die Ausgabe für den deutschen Sprachraum.

Tabelle 7.1: Die wichtigsten LaTeX-spezifischen Pandoc-Optionen

Defaultformatierung

Der schnellste Weg vom Markdown-Text zum PDF-Dokument ist also `pandoc in.md -o out.pdf`. Es ist aber fraglich, ob Sie mit dem Ergebnis glücklich werden (siehe die folgende Abbildung).

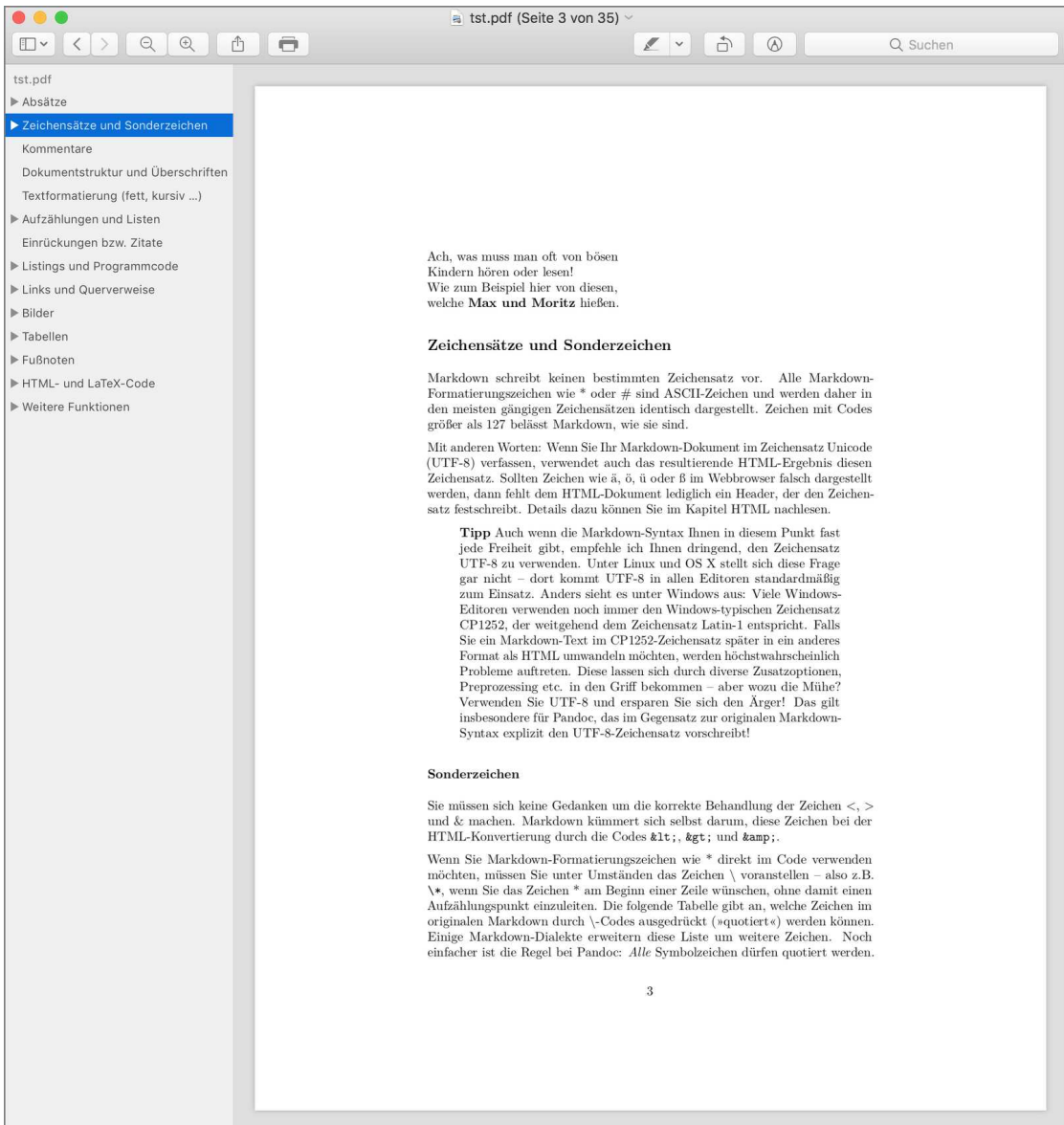


Abbildung 7.1: Ein PDF-Dokument in der Defaultformatierung

Standardmäßig gelten die folgenden Formatierungsregeln:

- Das Seitenformat ist US-Letter (216*279 mm).
- Die Seitenränder sind sehr großzügig; links, rechts, oben und unten gibt es mehrere Zentimeter freien Raum.
- Es gibt keine Kopfzeile, die Fußzeile zeigt mittig die Seitennummer.
- Absätze werden im Blocksatz dargestellt.
- Es kommen die LaTeX-Standardschriften zum Einsatz.
- Tabellen werden mittig angeordnet. Intern kommt das LaTeX-Erweiterungspaket `longtable` zum Einsatz; es erlaubt Tabellen, die über mehrere Seiten reichen.
- Innerhalb von Code-Listings erfolgt die Syntaxhervorhebung durch das `fancyvrb`-Paket.
- Breite Tabellen und Listings ragen über den Seitenspiegel hinaus.
- Die Beschriftung des Inhaltsverzeichnisses sowie von Tabellen und Bildern erfolgt in englischer Sprache (*Contents*, *Table n*, *Figure n*).

In der Defaultformatierung wollen Sie die PDF-Datei vermutlich weder ausdrucken noch weitergeben. Um Ihnen einen Ansatzpunkt zu bieten, wie Sie zu ästhetisch ansprechend gestalteten Dokumenten kommen, finden Sie auf den folgenden Seiten einige Tipps zur Layout-Optimierung. Erwarten Sie sich aber nicht eine Rezeptsammlung, die für jede Art von Dokument geeignet ist. Dazu müsste ich Ihnen die volle Bandbreite der LaTeX-Erweiterungen vorstellen, was an dieser Stelle nicht möglich ist.

LaTeX-Template und Template-Variablen

Mit `pandoc -D latex > default.latex` können Sie sich die Template-Datei für LaTeX-Dokumente ansehen. Die folgenden Listings zeigen einige wichtige Passagen dieser mehr als 400 Zeilen langen Datei. In den ersten Zeilen werden die Template-Variablen `fontsize`, `lang`, `papersize` und `documentclass` ausgewertet, um die Eckdaten des LaTeX-Dokuments einzustellen. Standardmäßig kommt die Schrift *Latin Modern* zum Einsatz.

Die Pakete `ifxetex` und `ifluatex` stellen die zusätzlichen Kommandos `\ifxetex` bzw. `\ifluatex` zur Verfügung. Die resultierende Bedingung ergibt nur dann `true`, wenn die Datei mit `xelatex` bzw. `lualatex` verarbeitet wird. Die beiden `\if`-Kommandos werden im weiteren Code dazu verwendet, um XeLaTeX- bzw. LuaLaTeX-spezifischen Code auszuführen. Wenn eine dieser beiden Engines aktiv ist, können mit den Template-Variablen `mainfont`, `sansfont`, `monofont` und `mathfont` die gewünschten Fonts eingestellt werden.

```
...
\documentclass[ $\if(fontsize)$$fontsize$, $endif$
                 $\if(lang)$$babel-lang$, $endif$
                 $\if(papersize)$$papersize$paper, $endif$...]{ $\$$ 
                documentclass $\$$ }

...
 $\if(fontfamily)$$
  \usepackage[ $\for(fontfamilyoptions)$$
     $\fontfamilyoptions$$sep$, $endfor$ $\$$ ]{ $\fontfamily$$ }
 $\else$ 
  \usepackage{lmodern}
 $\endif$ 

...
\usepackage{amssymb, amsmath}
\usepackage{ifxetex, ifluatex}
\usepackage{fixltx2e} % provides \textsubscript

...
 $\if(geometry)$$
  \usepackage[ $\for(geometry)$$geometry$$sep$, $endfor$ $\$$ ]{geometry}
 $\endif$$$$$$$$$ 
```

Als Nächstes werden einige optionale Pakete eingebunden, die zur Formatierung von Quellcode dienen (`listings`, `fancyvrb`), zur Verarbeitung von Literaturverweisen (`biblatex`), zur Einbettung von Bildern (`graphicx`), zum Umgang mit Links und Querverweisen (`hyperref`) etc.

Mehrere `\setlength`- bzw. `\setcounter`-Kommandos legen die Einrückung der Absätze, die Abstände zwischen zwei Absätzen und die Nummerierung von Überschriften fest.

```
...
$if(numbersections)$
\setcounter{secnumdepth}{$if(secnumdepth)$secnumdepth$else$5$endif$}
$else$
\setcounter{secnumdepth}{0}
$endif$
...
$if(indent)$
$else$
\IfFileExists{parskip.sty}{%
\usepackage{parskip}
}{% else
\setlength{\parindent}{0pt}
\setlength{\parskip}{6pt plus 2pt minus 1pt}
}
$endif$
```

Die restlichen Zeilen der Template-Datei sind dafür verantwortlich, die Include-Dateien zu integrieren (Pandoc-Optionen -H, -B und -A), den Dokumenttitel und das Inhaltsverzeichnis darzustellen und das eigentliche Dokument (Variable body) einzufügen.

```
% Header-Includes (Option -H)
$for(header-includes)$
$header-includes$
$endfor$

% Titel, Subtitel, Autoren, Datum etc.
$if(title)$
\title{$title$$if(thanks)$\thanks{$thanks}$endif$}
$endif$
...

% hier beginnt die eigentliche Ausgabe
\begin{document}
$if(title)$
$if(beamer)$
```



```
\frame{\titlepage}
$else$
  \maketitle
$endif$
...
$endif$

% Before-Includes (Option -B)
$for(include-before)$
  $include-before$
$include-before$
$endfor$

% Inhaltsverzeichnis (stark gekürzt)
$if(toc)$
  \tableofcontents
$endif$
...

% das eigentliche Dokument
$body$

... BibLaTeX-Referenzen

% abschließende Includes (Option -A)
$for(include-after)$
  $include-after$
$endfor$
\end{document}
```

Die folgende Tabelle fasst die wichtigsten LaTeX-spezifischen Template-Variablen zusammen, die innerhalb der Template-Datei verarbeitet werden. Eine Referenz weiterer Variablen, die für alle Dokumentformate gelten, finden Sie im Abschnitt [Template-Syntax](#).

Variable	Bedeutung
lang	Sprache (für Überschriften, Silbentrennung etc.)
fontsize	Schriftgrundgröße (10pt, 11pt oder 12pt)
papersize	Papiergröße, z. B. a4paper
geometry	Optionen für das geometry-Paket
documentclass	LaTeX-Dokumenttyp, z. B. book, report oder article

Tabelle 7.2: Die wichtigsten LaTeX-spezifischen Template-Variablen

7.2 Individuelle Formatierung

Um die Formatierung von LaTeX- bzw. daraus resultierenden PDF-Dateien zu beeinflussen, gibt es drei Wege:

- Sie beeinflussen das vorgegebene Layout durch die Einstellung von Template-Variablen sowie durch zusätzliche Include-Dateien.
- Sie verändern die Template-Datei. Da sich diese Datei in den vergangenen Jahren praktisch mit jeder Pandoc-Version verändert hat, rate ich davon aber eher ab.
- Sie erzeugen mit Pandoc nur den LaTeX-Code Ihrer Markdown-Dateien und betten das Ergebnis in Ihr eigenes LaTeX-Projekt ein (d. h., Sie verzichten auf die Option `-s` sowie auf das von Pandoc vorgegebene Template). Das setzt gute LaTeX-Kenntnisse voraus, gibt aber den größten Gestaltungsspielraum. (Dieses E-Book ist so entstanden.)

Dieser Abschnitt konzentriert sich auf die erste Variante und gibt Formatierungstipps für die wichtigsten Textelemente.

Seitenformat und Seitenlayout

Mit der Option `-V papersize=xxx` können Sie zwischen einigen vordefinierten Papiergrößen auswählen:

```
letterpaper | legalpaper | executivepaper | a4paper | a5paper | b5paper
```

Wesentlich mehr Papiergrößen kennt das LaTeX-`geometry`-Paket ([Link zur Paketbeschreibung](#)). Ein vordefiniertes Format stellen Sie mit `-V geometry=xxx` ein, z. B. `-V geometry=b6paper`. Mit der Template-Variable `geometry` können Sie auch eigene Papierformate definieren, indem Sie die Breite und Höhe des Papiers explizit angeben: `-V geometry=paperwidth=20cm,paperheight=20cm`

Mit einigen weiteren `geometry`-Einstellungen können Sie auch die Seitenränder verändern. Das folgende Pandoc-Kommando erzeugt ein PDF-Dokument im A4-Format, wobei die Seitenränder links 3 cm, rechts, oben und unten hingegen 2 cm betragen. Achten Sie darauf, dass Kopf- und Fußzeilen *innerhalb* der mit `geometry` definierten Seitenränder platziert werden.

```
pandoc in.md \  
-V geometry=a4paper,left=3cm,right=2cm,top=2cm,bottom=2cm \  
-o tst.pdf
```

Tipp

Einen ausgezeichneten Überblick darüber, wie Sie unter LaTeX Einfluss auf das Seitenlayout nehmen können, gibt die englischsprachige [Wikibooks-Pagelayout-Seite](#). Dort finden Sie auch Tipps für doppelseitige Layouts, bei denen für die linken (ungeraden) und die rechten (geraden) Seiten unterschiedliche Layouts gelten.

Kopf- und Fußzeilen

Zur individuellen Gestaltung der Kopf- und Fußzeilen kommt in LaTeX häufig das [fancyhdr-Paket](#) zum Einsatz. Im LaTeX-Template ist zur Verwendung dieses Pakets kein Code vorgesehen. Schreiben Sie den erforderlichen Code daher in eine kurze LaTeX-Datei und binden Sie diese mit der Option `-H` in den LaTeX-Header ein.

Die folgenden Zeilen geben ein Beispiel für eine derartige Datei mit dem Namen `myheader.tex`. Die Datei bewirkt, dass es auf der jeweils ersten Seite eines Kapitels gar keine Kopf- und Fußzeilen gibt. Auf allen anderen Seiten gibt es eine Kopfzeile, die links den Kapitelnamen und rechts die Seitennummer enthält. Die Kopfzeile ist unterstrichen. `\leftmark` ist dabei ein in LaTeX vordefiniertes Kommando, das die aktuelle Kapitelüberschrift zusammen mit einem weiteren Kommando zur Umwandlung des Texts in Großbuchstaben enthält. `\noupเปอร์case` hebt die Wirkung dieses Großbuchstabenkommandos auf.

```
% Datei myheader.tex
\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyhead{}      % vorhandene Kopf- und
\fancyfoot{}      % Fußzeilen löschen
                  % Kopfzeile links: Kapitelüberschrift
\lhead{\noupเปอร์case{\leftmark}}
\rhead{\thepage} % Kopfzeile rechts: Seitennummer
```

Der Pandoc-Aufruf sieht dann z. B. folgendermaßen aus:

```
pandoc in.md -H myheader.tex -o out.pdf
```

Überschriften

Normalerweise werden Überschriften nicht nummeriert. Die Pandoc-Option `--number-sections` bewirkt eine Nummerierung *aller* Überschriften. Wenn Sie ein Mittelding wünschen, z. B. nur eine Nummerierung der Überschriften der ersten und zweiten Ebene, fügen Sie die folgende Anweisung in eine eigene Header-Datei ein. Die Zählung der Ebenen beginnt bei 0.

```
% Ergänzung in myheader.tex
\setcounter{secnumdepth}{1}
```

Die Option `--top-level-division=chapter` bewirkt vordergründig, dass Überschriften erster Ebene als Kapitelüberschriften interpretiert werden. Hinter den Kulissen hat die Option aber zur Folge, dass Pandoc nun den LaTeX-Dokumenttyp `book` anstelle von `article` verwendet.

`book` ist ein Dokumenttyp mit zweiseitigem Layout: Linke und rechte Seiten werden nun mit unterschiedlichen Kopf- und Fußzeilen gestaltet, die linken und rechten Seitenränder werden bei jeder zweiten Seite vertauscht, um für die Buchbindung optimale Abstände zu erzielen etc. Wenn Sie das nicht wollen, stellen Sie die Template-Variable `papersize` auf `oneside` und verwenden Sie zur Seitengestaltung die `geometry`-Variable:

```
pandoc in.md --top-level-division=chapter \
-V geometry=a4paper \
-V papersize=oneside -o out.pdf
```

Wenn Sie auf die Gestaltung der Überschriften Einfluss nehmen möchten (Schriftart, Schriftgröße, Farben etc.), können Sie dazu das [titlesec-Paket](#) verwenden. Die erforderlichen Anweisungen fügen Sie in eine Header-Datei ein, auf die Sie beim Aufruf von Pandoc mit der Option `-H` verweisen.

```
% Ergänzung in myheader.tex
\usepackage{titlesec, color}
\definecolor{myred}{rgb}{0.8, 0.1, 0.1}
\titleformat{\chapter}[display]
  {\normalfont\sffamily\huge\bfseries\color{myred}}
  {\chaptertitlename\ thechapter}{20pt}{\Huge}
\titleformat{\section}
  {\normalfont\sffamily\Large\bfseries\color{myred}}
  {\thesection}{1em}{}
```

Kompatibilitätsprobleme

titelsec ist mit aktuellen Pandoc-Versionen inkompatibel, weil im LaTeX-Template `\paragraph` neu definiert wird. Das können Sie glücklicherweise verhindern, indem Sie beim Aufruf von Pandoc die Optionen `-V paragraph=1` und `-V subparagraph=1` verwenden.

Sprache

Damit Überschriften, Tabellen und Abbildungen in deutscher Sprache beschriftet werden (also z. B. *Kapitel 1* statt *Chapter 1*), verwenden Sie beim Pandoc-Aufruf die Option `-V lang=de`. Damit gelten auch die deutschen Regeln für die Silbentrennung.

Silbentrennung

Mitunter trennt LaTeX leider falsch bzw. nicht so, wie Sie es möchten. Sofern es Ihr Editor unterstützt, können Sie in Ihrem Markdown-Dokument in das Wort ein weiches Trennzeichen (*soft hyphen*, Unicode U+00AD) einfügen. (Unter Windows gelingt das zumeist mit der Tastenkombination *AltGr + -*.) Pandoc belässt das Zeichen, LaTeX wertet es korrekt aus.

Alternativ können Sie auch mit `\linebreak` einen Zeilenumbruch erzwingen oder mit `\mbox{Wort}` die Trennung des Worts unterbinden. (Die `mbox`-Variante ist allerdings inkompatibel zu anderen Ausgabeformaten. Wenn Sie also Ihr Markdown-Dokument in ein HTML-Dokument umwandeln, verschwindet das angegebene Wort.)

Wird ein oft benötigtes Wort ständig falsch getrennt, können Sie es in eine Ausnahmenliste eintragen, in der Sie die zulässigen Trennpositionen explizit angeben. Diese Ausnahmenliste kann sich in einer zusätzlichen LaTeX-Header-Datei befinden und wird mit der Pandoc-Option `-H` inkludiert.

```
% Ergänzung in myheader.tex
\hyphenation{MySQL mysql ge-startet Be-nutzer-ober-fläche
Be-nutzer-ober-flächen Client Clients Cross-Over Down-load
Down-loads Gnome Google Medi-buntu My-SQL Open-Office Open-VPN
open-SUSE SELinux Up-date Up-dates Web-site Web-sites Network-Manager
Kernel-optionen Kernel-option Kernel-parameter Boot-loader Boot-datei
Boot-dateien Virtual-Box}
```

Fonts und Schriftgröße

Ob und wie Sie Einfluss auf die von LaTeX verwendeten Schriften nehmen können, hängt stark von der LaTeX-Engine ab. Wenn Sie XeLaTeX oder LuaLaTeX verwenden, können Sie die Grundschriften recht unkompliziert mit vier Template-Variablen steuern. Dabei können Sie auf alle lokal verfügbaren TrueType- bzw. OpenType-Schriften zurückgreifen.

```
pandoc in.md --latex-engine xelatex \
-V mainfont=Arial -o out.pdf
```

Variable	Bedeutung
mainfont	Standardschrift
sansfont	Sans-Serif-Schrift (<code>\sf</code> , <code>\textsf</code>)
monofont	Schrift für Listings (<code>\tt</code> , <code>verbatim</code> -Umgebungen)
mathfont	Schrift für mathematische Formeln

Tabelle 7.3: Template-Variablen zur Font-Einstellung für XeLaTeX und LuaLaTeX

Wenn Sie mit der PdfLaTeX-Engine arbeiten, auf die Pandoc standardmäßig zurückgreift, können Sie versuchen, die Schriftart mit `-V fontfamily=electrum` einzustellen. Das setzt aber voraus, dass die betreffende Schriftart LaTeX-kompatibel installiert ist – eine durchaus nicht triviale Angelegenheit. Das TeXLive-Paket `texlive-fonts-extra` enthält eine Sammlung freier Schriften. (Versuchen Sie es z. B. mit `-V fontfamily=electrum`.)

Eine zweite Variante besteht darin, in die eigene Header-Datei LaTeX-Pakete einzubeziehen, die die LaTeX-eigenen Schriften durch Standard-PostScript-Fonts ersetzen. Dazu zählen z. B. die Pakete `helvet`, `mathpazo` oder `bookman`. Eine Referenz dieser Pakete gibt die [PSNFSS-Dokumentation](#). Falls Sie außerdem eine serifenlose Schrift vorziehen, führen Sie eine Neudefinition des LaTeX-Kommandos `\familydefault` durch. Bei langen Zeilen können Sie die Lesbarkeit erhöhen, indem Sie den Zeilenabstand mit `\linespread` vergrößern.

```
% Ergänzung in myheader.tex
% die PostScript-Schriften Helvetica verwenden
\usepackage{helvet}
% Fließtext mit einer Sans-Serif-Schrift darstellen
\renewcommand{\familydefault}{\sfdefault}
% Zeilenabstand um 10% vergrößern
\linespread{1.1}
```

Wenig Einflussnahme gibt LaTeX auf die Schriftgröße: Mit `-v fontsize=x` können Sie die Größe für normalen Fließtext einstellen, wobei aber nur die Einstellungen 10pt (gilt standardmäßig), 11pt und 12pt zulässig sind. Die Schriftgrößen in Überschriften, Fußnoten etc. werden daraus automatisch abgeleitet.

Tipp

Wenn es Ihnen bei der PDF-Erzeugung nur um die Darstellung am Bildschirm geht, nicht um den Ausdruck, dann können Sie die visuell wahrgenommene Schriftgröße auch durch die Papiergröße verändern: Je kleiner Sie das Papierformat wählen, desto größer erscheint der Text.

Fließtext

Absätze werden standardmäßig im Blocksatz dargestellt. Wenn Sie das nicht möchten, aktivieren Sie in Ihrer eigenen Header-Datei das Paket `ragged2e`. Den Abstand zwischen zwei Absätzen können Sie durch eine Neueinstellung von `\parskip` ändern. Im folgenden Listing wird das Sollmaß auf 2 mm festgelegt. Bei Bedarf darf LaTeX den Abstand um einen halben Millimeter vergrößern bzw. um einen viertel Millimeter verkleinern, um ein optimales Seitenlayout zu erzielen.

```
% Ergänzung in myheader.tex
% kein Blocksatz
\usepackage[document]{ragged2e}

% Abstand zwischen Absätzen
\setlength{\parskip}{2mm plus 0.5mm minus 0.25mm}
```


Listen und Aufzählungen

Um das Erscheinungsbild von Aufzählungen zu verändern, fügen Sie in Ihre Header-Datei am besten das `enumitem`-Paket ein ([Link](#)). Die folgenden Zeilen zeigen die Anwendung:

```
% Ergänzung in myheader.tex
\usepackage{enumitem}
\usepackage[geometry]{ifsym}

% gewöhnliche Listen
\setlist[itemize]{labelwidth=1.2em,labelsep=0cm,leftmargin=1.2em,
  rightmargin=0cm,listparindent=0cm,itemindent=0cm,topsep=-0.8mm,
  itemsep=0mm,align=left,label=\scriptsize\FilledSmallSquare}
\setlist[itemize,2]{label=--} % 2. Ebene
\setlist[itemize,3]{label=\tiny\Circle} % 3. Ebene

% nummerierte Listen
\setlist[enumerate]{labelwidth=1.2em,labelsep=0cm,leftmargin=1.2em,
  rightmargin=0cm,listparindent=0cm,itemindent=0cm,topsep=-0.8mm,
  itemsep=0mm,align=left}
% label=\bfseries{\arabic*} wird ignoriert, weil
% Pandoc in den LaTeX-Code \def\labelenumi{\arabic{enumi}.}
% einbaut

% Definitionslisten
\setlist[description]{leftmargin=1.2em,rightmargin=0cm,
  font={\bfseries},topsep=-0.8mm,itemsep=2mm,labelsep=1000mm}
%labelsep erzwingt neue Zeile ohne Abstand
```

Pandoc unterscheidet zwischen kompakten und normalen Listen. Kompakte Listen kommen dann zur Anwendung, wenn Sie im Markdown-Text keine Leerzeilen zwischen den Aufzählungspunkten bzw. bei Definitionslisten keine Leerzeile zwischen dem Schlüsselbegriff und der nachfolgenden Erklärung lassen. Pandoc baut in solchen Fällen das Kommando `\tightlist` in den LaTeX-Code ein.

`\tightlist` ist im Default-Template definiert. Wenn Sie die von Pandoc erzeugte Datei selbst weiterverarbeiten, müssen Sie auch `\tightlist` in Ihrer eigenen Header-Datei definieren, z. B. so:

```
\newcommand{\tightlist}{\itemsep1pt\parskip0pt\parsep0pt}
```

Tabellen

Standardmäßig zentriert Pandoc alle Tabellen, die schmaler als die Seitenbreite sind. Abhilfe schafft die folgende Einstellung in einer eigenen Header-Datei:

```
% Ergänzung in myheader.tex
% Tabellen linksbündig
\usepackage[margins=raggedright]{floatrow}
```

Wahrscheinlich möchten Sie nun auch linksbündige Tabellenunterschriften. Die erreichen Sie am schnellsten mit dem `caption`-Paket. Die entsprechenden Anweisungen fügen Sie ebenfalls in die Header-Datei ein.

```
% Ergänzung in myheader.tex
% Gestaltung von Unterschriften von Tabellen und Abbildungen
\usepackage{caption}
\captionsetup{labelsep=colon,justification=justified,
             singlelinecheck=off}
\renewcommand{\captionfont}{\small\it}

% kleinerer Abstand zwischen Tabelle und Unterschrift
\setlength{\abovecaptionskip}{7pt plus 1pt minus 1pt}
```

Die Standardformatierung von LaTeX-Tabellen durch Pandoc ist minimalistisch. Mit dem folgenden Header-Code erreichen Sie farbige Begrenzungslinien sowie alternierende Hintergrundfarben für jede Zeile.

```

% Ergänzung in myheader.tex
% Farbe und Breite der Tabellenlinien
% alternierende Hintergrundfarben
\usepackage[table]{xcolor}
\definecolor{lightgray1}{gray}{0.88}
\definecolor{lightgray2}{gray}{0.94}
\let\oldlongtable\longtable
\let\endoldlongtable\endlongtable
\renewenvironment{longtable}%
  {\rowcolors{2}{lightgray2}{lightgray1}\oldlongtable}%
  {\endoldlongtable}

% Farbe und Breite der Tabellenlinien
\usepackage{colortbl}
\arrayrulecolor{black}%
\setlength\arrayrulewidth{1pt}% wirkt für hline, nicht für toprule etc.

% Tabellenzeilen etwas großzügiger dimensionieren
\renewcommand*\arraystretch{1.25}

```

Allerdings reichen nun die grauen Hintergrundbalken anfänglich beidseitig über die Tabelle hinaus. Weiters stören weiße Zwischenräume ober- und unterhalb der Trennlinien. Schließlich hat der Hintergrund der Beschriftungszeile dieselbe Farbe wie jene der ersten Zeile.

Um all diese optischen Mängel zu beheben, muss das Erzeugen der PDF-Datei in drei Schritte zerlegt werden: Zuerst erzeugt Pandoc eine LaTeX-Datei. Dann verändert das `sed`-Kommando (verfügbar nur unter macOS oder Linux) den LaTeX-Code. Und schließlich macht ein zweiter Pandoc-Aufruf aus der veränderten LaTeX-Datei die PDF-Datei.

Das `sed`-Kommando eliminiert `\@{}` in der Spaltendefinition der Tabelle, ersetzt die LaTeX-Kommandos `toprule`, `bottomrule` und `midrule` durch `hline` und löscht alle Zeilen zwischen den LaTeX-Kommandos `endfirsthead` und `endhead`.

```
pandoc in.md <viele Optionen ...> \
  --to latex -o tmp.tex

sed -e 's,\@{, ,g' \
  -e 's,\\toprule,\\hline,' \
  -e 's,\\bottomrule,\\hline,' \
  -e 's,\\midrule,\\hline,' \
  -e '/\endfirsthead/,/\endhead/d' \
  < tmp.tex > tst.tex

pdflatex tst.tex -o tst.pdf
```

Vorsicht

Die obigen Veränderungen an dem von Pandoc erzeugten LaTeX-Code sind durchaus problematisch. Sollte Pandoc in einer zukünftigen Version einen anderen LaTeX-Code produzieren, werden die hier vorgestellten Anpassungen vermutlich nicht mehr funktionieren. Dieses Problem hatte ich in der Vergangenheit schon mehrfach.

Damit sind die meisten Ärgernisse im Zusammenhang mit Tabellen behoben, eines verbleibt aber noch: Der Pandoc-Entwickler ist der Meinung, dass die Tabellenunterschrift eine -überschrift ist und daher oberhalb der Tabelle anzuordnen ist. Das mag im englischen Sprachraum üblich sein, bei uns aber nicht.

In früheren Pandoc-Versionen ließ sich dieses Problem unkompliziert beheben, z. B. durch die Einstellung

```
\captionsetup[table]{position=bottom}
```

in der LaTeX-Header-Datei. In aktuellen Pandoc-Versionen werden Tabellen aber als longtable dargestellt, wobei der prinzipielle Aufbau so aussieht:

```
\begin{longtable} [ ]{@{} l l l l @{}}
\caption{Sonderzeichen}\tabularnewline
... Code zur Darstellung der Tabelle
\end{longtable}
```

Da sich das `caption`-Kommando vor dem Code für die eigentliche Tabelle befindet, wird die Überschrift auch oberhalb angezeigt. Die einzige mögliche Abhilfe besteht darin, den Code wie folgt zu ändern:

```
\begin{longtable}[]{@{}l1111@{}}  
... Code zur Darstellung der Tabelle  
\caption{Sonderzeichen}\tabularnewline  
\end{longtable}
```

Genau diese Aufgabe übernimmt das folgende Python-Script:

```
#!/usr/bin/env python3  
import sys  
  
modus = 'copy'  
# zeilenweise aus Standardeingabe lesen  
for line in sys.stdin:  
    if modus == 'copy':  
        if line.startswith(r'\begin{longtable}'): # eine neue Tabelle startet, Text zwischenspeichern  
            modus = 'table'  
            table = line  
            caption = ''  
        else:  
            # im Copy-Modus Zeile einfach wieder ausgeben  
            sys.stdout.write(line)  
  
    elif modus == 'table':  
        if line.startswith(r'\caption{'):  
            # Beginn der überschrift gefunden, zwischenspeichern  
            caption = line  
            if not caption.strip().endswith(r'}\tabularnewline'):  
                # überschrift geht noch weiter  
                modus = 'caption'  
        elif line.startswith(r'\end{longtable}'): # Ende der Tabelle gefunden  
            sys.stdout.write(table)
```

```
# rowcolor ist nur notwendig, wenn die Tabelle
# alternierende Hintergrundfarben verwendet
sys.stdout.write(r'\rowcolor{white}')
```

```
sys.stdout.write(caption)
sys.stdout.write(line)
modus = 'copy'
else:
    # Tabelle geht weiter
    table += line

elif modus == 'caption':
    caption += line
    if line.strip().endswith(r'}\tabularnewline'):
        # überschrift ist fertig
        modus = 'table'
```

Der Prozess zur Erzeugung einer PDF-Datei ist jetzt schon relativ aufwendig (und Sie bekommen allmählich eine Idee, wie die PDF-Ausgabe dieses E-Books entstanden ist):

```
pandoc in.md <viele Optionen ...> \
  --to latex -o tmp1.tex

sed -e 's,\,\,g' \
  -e ... \
  < tmp1.tex > tmp2.tex

./caption-below.py < tmp2.tex > out.tex

# erzeugt out.pdf
pdflatex out.tex
```

Abbildungen

Pandoc zentriert Abbildungen. Dazu baut Pandoc in den LaTeX-Code die Anweisung `\centering` ein. Damit Abbildungen wie Tabellen rechtsbündig dargestellt werden, müssen Sie die `\centering`-Anweisungen mittels `sed` aus dem LaTeX-Quelltext entfernen. Gleichzeitig können Sie auch gleich die `figure`-Umgebung mit `[h]` ergänzen – dann bemüht sich LaTeX, die Bilder dort zu platzieren (*here*), wo sie im Text vorkommen. (Ohne diese Option wandern die Bilder sehr oft an den Beginn oder das Ende der Seite, auch wenn das gar nicht notwendig wäre. Dies passiert zwar auch mit dieser Option, aber etwas seltener.)

```
sed ... \
    -e 's,^\{\centering,\raggedright,' \
    -e 's,\{\begin{figure}\}[h],\{\begin{figure}\}[h],' \
< tmp1.tex > tmp2.tex
```

Falls Sie das im vorigen Abschnitt erwähnte `floatrow`-Paket einsetzen, müssen Sie außerdem die folgende Zeile in Ihre Header-Datei einbauen:

```
% in myheader.tex
\usepackage{floatrow}
\floatsetup[figure]{objectset=raggedright}
```

Zur Gestaltung der Bildunterschrift gelten dieselben Regeln wie für Tabellen. Glücklicherweise kommt Pandoc dabei nicht auf die Idee, aus der Unterschrift eine Überschrift zu machen.

```
% in myheader.tex
% Tabellen- und Bildbeschriftung linksbündig unten
\usepackage{caption}
\captionsetup{labelsep=colon,justification=justified,
              singlelinecheck=off}
\renewcommand{\captionfont}{\small\it}
```

Einrückungen bzw. Zitate

Pandoc verpackt mit `>` eingerückte Zitate in das LaTeX-`quote`-Environment, wobei mehrere Absätze nicht in *einer* derartigen Umgebung landen, sondern in *mehreren* hintereinander platzierten Umgebungen. Im resultierenden PDF-Dokument sind die Zitate tatsächlich eingerückt, allerdings mit unnötig großen Abständen ober- und unterhalb.

Wenn Sie eine andere Formatierung wünschen, bietet sich der Einsatz des `quoting`-Pakets an ([Link](#)):

```
% in myheader.tex
\usepackage{quoting}
\quotingsetup{leftmargin=1.2em,vskip=2.5mm,font={small,itshape}}
```

Außerdem müssen Sie in dem von LaTeX erzeugten Text `quote` durch `quoting` ersetzen. Mit `sed` gelingt das wie folgt:

```
sed -e 's,\\begin{quoting},\\begin{quoting},' \
    -e 's,\\end{quoting},\\end{quoting},' \
    < tmp1.tex > tmp2.tex
```

Um bei Zitaten, die über mehrere Absätze reichen, zu große Abstände zu entfernen, müssen Sie im LaTeX-Quelltext das Ende und den nachfolgenden Start von `quoting` eliminieren.

```
\begin{quoting}
text
\end{quoting}      % <- diese Zeile löschen

\begin{quoting}    % <- diese auch
text
\end{quoting}
```

Diese Arbeit kann ein kleines Python-Script übernehmen:


```
#!/usr/bin/env python3
# Verwendung: ./join-quoting.py < in > out

# alle Zeilen lesen
import sys
fin = sys.stdin
fout = sys.stdout
lines = fin.readlines()
i=0
max=len(lines)

# Schleife über alle Zeilen, jeweils 3 Zeilen auf einmal vergleichen
while i<max-3:
    if lines[i].strip() == r'\end{quoting}' and \
        lines[i+1].strip() == '' and \
        lines[i+2].strip() == r'\begin{quoting}':
        del lines[i+2]
        del lines[i]
        max-=2
    i+=1

# Ausgabe
for line in lines:
    print(line, end='', file=fout)
```

Der Aufruf von Pandoc, sed, des Python-Scripts und latex sieht dann so aus:

```
pandoc in.md <viele Optionen ...> \
--to latex -o tmp1.tex

sed -e 's,\\begin{quoting},\\begin{quoting},' \
-e 's,\\end{quoting},\\end{quoting},' \
< tmp1.tex > tmp2.tex

./join-quoting.py < tmp2.tex > out.tex

# erzeugt out.pdf
pdflatex out.tex
```

Listings mit dem fancyvrb-Paket

Standardmäßig verwendet Pandoc das fancyvrb-Paket, um Quellcode darzustellen. Im Header-Bereich des LaTeX-Quellcodes sind die Umgebungen Shaded und Highlighting sowie Kommandos zur Hervorhebung bestimmter Codeteile definiert. Die Funktionsweise geht aus dem folgenden Beispiel hervor. Zuerst der Markdown-Code:

```

---java
// Hello World in Java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
---
```

Und hier der daraus erzeugte LaTeX-Code, wobei ich sehr lange Zeilen zur besseren Lesbarkeit geteilt habe und einige zusätzliche Zeilenumbrüche eingefügt habe:

```

% Header
\usepackage{fancyvrb}
\DefineVerbatimEnvironment{Highlighting}{Verbatim}{commandchars=\\\{\}}
\newenvironment{Shaded}{}{}
...
\newcommand{\KeywordTok}[1]{\textcolor[rgb]{0.00,0.44,0.13}{\textbf{
    #1}}}
\newcommand{\NormalTok}[1]{#1}
\newcommand{\StringTok}[1]{\textcolor[rgb]{0.25,0.44,0.63}{#1}}
...

% das eigentliche Listing
\begin{Shaded}
\begin{Highlighting}[]
\CommentTok{// Hello World in Java}

\KeywordTok{public} \KeywordTok{class}\NormalTok{ HelloWorld \{}
```

```

\KeywordTok{public} \DataTypeTok{static} \DataTypeTok{void}
  \FunctionTok{main}\NormalTok{{}\BuiltInTok{String}
  \NormalTok{[] args) \{}}

  \BuiltInTok{System}\NormalTok{.}\FunctionTok{out}\NormalTok{.}
  \FunctionTok{println}\NormalTok{{}\StringTok{"Hello World!"}\
    NormalTok{);}

\NormalTok{ \}}

\NormalTok{\}}
\end{Highlighting}
\end{Shaded}

```

Sie sehen schon, der Java-Code ist im LaTeX-Quelltext praktisch nicht mehr zu lesen. Gewöhnliche Listings ohne Syntax-Highlighting werden in einer normalen verbatim-Umgebung dargestellt.

Wenn Sie möchten, dass Listings – egal ob mit oder ohne Hervorhebungen – in einer etwas kleineren Schrift dargestellt werden, fügen Sie in Ihrer eigenen LaTeX-Header-Datei die folgenden Zeilen ein:

```

% Ergänzung in myheader.tex
\DefineVerbatimEnvironment{Highlighting}{Verbatim}%
  {commandchars=\\\{\}, fontsize=\small}
\DefineVerbatimEnvironment{verbatim}{Verbatim}%
  {fontsize=\small}

```

Um die Listings außerdem zu umranden, gehen Sie auf diese Weise vor:

```
% Ergänzung in myheader.tex
\usepackage{xcolor}
\DefineVerbatimEnvironment{Highlighting}{Verbatim}%
  {commandchars=\\\{\},fontsize=\small,%
  frame=single,rulecolor=\color{gray}}
\DefineVerbatimEnvironment{verbatim}{Verbatim}%
  {fontsize=\small,%
  frame=single,rulecolor=\color{gray}}
```

Leider sieht das `fancyvrb`-Paket weder die Einstellung einer Hintergrundfarbe noch eine Veränderung des Liniensstils für die Umrandung vor. Für Quelltext mit Syntaxhervorhebung können Sie die Umgebung `Shaded` neu definieren und auf diese Weise eine Hintergrundfarbe einstellen; bei meinen Tests hat das aber nicht zufriedenstellend funktioniert. Außerdem hat die Vorgehensweise den Nachteil, dass sie für Listings ohne Syntaxhervorhebung wirkungslos bleibt.

Listings mit dem `listings`-Paket

Sie können Pandoc auch mit der Option `--listings` aufrufen. Der produzierte LaTeX-Code verwendet nun zur Darstellung von Quelltext das `listings`-Paket. Das ist mit Vor- und Nachteilen verbunden:

- Das `listings`-Paket bietet mehr Möglichkeiten zur optischen Gestaltung der Listings.
- Die Quelltexthervorhebung erfolgt nicht durch von Pandoc erzeugte Kommandos, sondern direkt durch `listings`-eigenen Code. Die Lesbarkeit des LaTeX-Quellcodes ist damit viel besser. Allerdings funktioniert das Syntaxhighlighting für weniger Programmiersprachen.
- Unbegreiflicherweise hat das `listings`-Paket Probleme mit den deutschen Buchstaben `äöüß` sowie mit anderen Nicht-ASCII-Zeichen. Diese Probleme lassen sich aber durch einige Anweisungen in einer eigenen Header-Datei beheben.
- Auch sonst werden Sie mit dem `listings`-Paket nur zufriedenstellende Ergebnisse erzielen, wenn Sie ein wenig Zeit investieren und das Paket per `\lstset`-Kommando in Ihrer eigenen Header-Datei konfigurieren.

Der von Pandoc erzeugte LaTeX-Quelltext für Listings sieht nun so aus:

```
% allgemeines Listing
\begin{lstlisting}
abc
\end{lstlisting}

% Java-Listing
\begin{lstlisting}[language=Java]
// Hello World in Java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
\end{lstlisting}
```

Die folgenden Zeilen geben einen Vorschlag zur Konfiguration des `listings`-Pakets. Sie erreichen damit, dass diverse Nicht-ASCII-Zeichen korrekt verarbeitet werden und dass alle Listings mit hellgrauem Hintergrund und dunkelgrauer Umrandung dargestellt werden.

```
% Ergänzung in myheader.tex
% Farben für den Hintergrund und bestimmte Code-Elemente
\usepackage{xcolor}
\definecolor{lstback}{rgb}{0.94 0.94 0.94}
\definecolor{cmtcol}{rgb}{0.0 0.5 0.0}
\definecolor{keycol}{rgb}{0.5 0.0 0.0}

% listings-Konfiguration
\lstset{%
    frame=single,
    aboveskip=12pt,
    backgroundcolor=\color{lstback},
    rulecolor=\color{lightgray},
    basicstyle=\ttfamily\small,
    keywordstyle=\color{keycol},
```

```
emphstyle=\bfseries,
commentstyle=\color{cmtcol},
showstringspaces=false, % Leerzeichen normal anzeigen
breaklines=true,        % zu lange Zeilen umbrechen
extendedchars=true,     % Nicht-ASCII-Zeichen korrekt behandeln
literate={ä}{\a}1      {ö}{\o}1 {ü}{\u}1
                   {Ä}{\A}1      {Ö}{\O}1 {Ü}{\U}1
                   {ß}{\ss}1,
```

Wenn Sie ein einzelnes Listing abweichend von den Defaulteinstellungen behandeln möchten, können Sie `\lstset` auch im Markdown- oder LaTeX-Quellcode einsetzen, um einzelne Einstellungen vorübergehend zu ändern. Im folgenden Beispiel soll ein einzelnes Listing in einer noch kleineren Schrift dargestellt werden.

```
\lstset{basicstyle=\ttfamily\footnotesize}

...
Dieses Listing mit sehr langen Zeilen soll in einer besonders kleinen Schrift
dargestellt werden.
...

\lstset{basicstyle=\ttfamily\small}
```

Kurze Codepassagen innerhalb des Texts (also z. B. ``abc``) werden von Pandoc nun zum LaTeX-Code `\lstinline!abc!` verarbeitet. Wenn das erste Zeichen einer derartigen Codepassage ein doppeltes Anführungszeichen oder ein deutsches Sonderzeichen (also äöüß) ist, tritt ein Fehler auf. Dieser Fehler hat nichts mit Pandoc zu tun, vielmehr handelt es sich offensichtlich um einen Fehler im `listings`-Paket. Abhilfe schafft es, in solchen Fällen per `sed` dem ersten Zeichen einen Backslash voranzustellen.

```
pandoc in.md [viele Optionen ...] -o tmp.tex
sed -e 's,\\lstinline!\",\\lstinline!\\\",g' \
    -e 's,\\lstinline!\ä,\\lstinline!\\ä,g' \
    -e 's,\\lstinline!\ö,\\lstinline!\\ö,g' \
    ... \
    < tmp.tex > out.tex
pdflatex out.tex
```

Beispieldateien

Auf der Begleitseite zu diesem E-Book stelle ich eine Sammlung von Beispieldateien zum Download zur Verfügung. Dort finden Sie unter anderem eine umfangreiche LaTeX-Header-Datei mit allen Beispielen aus diesem Kapitel sowie ein Shell-Script zum Aufruf von pandoc, sed, latex etc.

https://kofler.info/ebooks/markdown_pandoc

8 Folien

Nicht immer muss es gleich ein ganzes Buch bzw. E-Book sein! Zum Alltag vieler Autoren zählt die Vorbereitung von Vorträgen. Pandoc hilft Ihnen auch dabei und bietet einen besonders effektiven Weg, um Vortragsfolien zu erstellen. Dieses Kapitel zeigt, welche Optionen Ihnen dabei zur Auswahl stehen und welche Gestaltungsmöglichkeiten es gibt. Eines gleich vorweg: Wenn Sie bei Vortragsfolien an lustige Animationen, Sound-Effekte etc. denken, bleiben Sie besser bei Powerpoint oder LibreOffice! Mit Pandoc erstellte Folien sehen professionell aus, verzichten aber auf unnötige Spielereien.

8.1 Grundlagen

PDF oder HTML?

Wie die folgende Tabelle zeigt, unterstützt Pandoc gleich sieben Formate zur Gestaltung von Vortragsfolien; letztlich ergeben sich daraus aber nur drei grundlegende Varianten: PDF, HTML oder PowerPoint:

- Obwohl es beeindruckend ist, dass Pandoc sogar das PowerPoint-Format unterstützt, ist die praktische Bedeutung dieser Variante eher gering. Wenn Sie wirklich PowerPoint-Folien brauchen, ist es zumeist sinnvoller, gleich mit Microsoft Office zu arbeiten.
- Für die PDF-Variante auf Basis des LaTeX-Pakets beamer spricht der Umstand, dass die Folien unkompliziert als PDF-Datei weitergegeben werden können. Naturwissenschaftliche Autoren werden außerdem die perfekte Unterstützung mathematischer Formeln zu schätzen wissen.

- Der Hauptvorteil des HTML-Formats besteht darin, dass sich die Vorträge gut in Webseiten einbetten lassen. Außerdem ersparen Sie sich die Auseinandersetzung mit LaTeX-Eigenheiten. Für welche der vier HTML-Varianten Sie sich entscheiden, ist primär eine Designfrage. Probieren Sie einfach alle vier Varianten aus und verwenden Sie jene, die Ihnen optisch am meisten zusagt.

Format	Formatname
PowerPoint	pptx
PDF (LaTeX Beamer)	beamer
HTML (DZSlides)	dzslides
HTML (reveal.js)	revealjs
HTML (Slidy)	slidy
HTML (Slideous)	slideous
HTML (S5 Slides)	s5

Tabelle 8.1: Von Pandoc unterstützte Folien-Formate

Strukturierung von Dokumenten

Sie verfassen Ihren gesamten Vortrag in einer Markdown-Datei. Die naheliegende Frage ist nun: Wie erkennt Pandoc, an welcher Stelle eine neue Folie beginnt?

- Automatische Trennung aufgrund von Überschriften:** Pandoc versucht anhand der Strukturierung Ihres Textes zu erkennen, welche Überschriftenebene Sie für Folienüberschriften verwenden. Diese Überschriften werden *slide level headers* genannt. Als *slide level* gelten die Überschriften der niedrigsten Ebene, denen unmittelbar Text folgt. Am einfachsten ist das anhand des folgenden Beispiels zu verstehen:

```
# Vortragsüberschrift

## Teil 1: Einführung
```

```
### Folie 1

bla bla

### Folie 2

noch mehr bla bla

## Teil 2: Grundlagen
```

Bei diesem Dokument gelten also Überschriften der Ebene 3 (eingeleitet durch `###`) als *slide level headers*. Bei jeder derartigen Überschrift beginnt automatisch eine neue Folie. Für Überschriften der Ebenen 1 und 2 erzeugt Pandoc Folien, die nur die Überschrift enthalten. Insgesamt bestünde der obige Minivortrag aus sechs Seiten: der Startseite, einer Seite mit der Vortragsüberschrift, einer weiteren mit der Überschrift *Teil 1 ...* sowie zwei Seiten für *Folie 1* und *Folie 2*.

- **Manuelle Trennung aufgrund von Überschriften:** In vielen Fällen funktioniert die automatische Erkennung des *slide levels* gut. Problematisch wird es aber, wenn Sie auch noch Text auf einer Überschriftenfolie unterbringen möchten:

```
# Vortragsüberschrift

## Teil 1: Einführung

* eins
* zwei
* drei

### Folie 1

bla bla

... weiter wie oben

## Teil 2: Grundlagen
```

Pandoc glaubt nun, der *slide level* sei 2. Das Ergebnis besteht aus nur vier Seiten: der Startseite, der Vortragsüberschrift, einer Seite für *Teil 1* ... und noch einer Seite für *Teil 2* ...

Abhilfe schafft in solchen Fällen die Pandoc-Option `--slide-level n`. Für das obige Beispiel müssten Sie `n` durch 3 ersetzen.

- **Manuelle Trennung durch -----:** Losgelöst von den Überschriften können Sie eine neue Folie jederzeit durch eine horizontale Linie starten, die aus zumindest drei Minuszeichen bestehen muss.

Titelseite

Pandoc erzeugt automatisch eine Titelseite mit drei Angaben: dem Vortragstitel, dem bzw. den Autorennamen und dem Datum. Dazu müssen Sie diese drei Informationen am Beginn Ihrer Markdown-Datei in folgender Form angeben (siehe auch den Abschnitt [Metadaten](#)):

```
% Titel  
% Autor  
% Datum
```

8.2 HTML-Folien

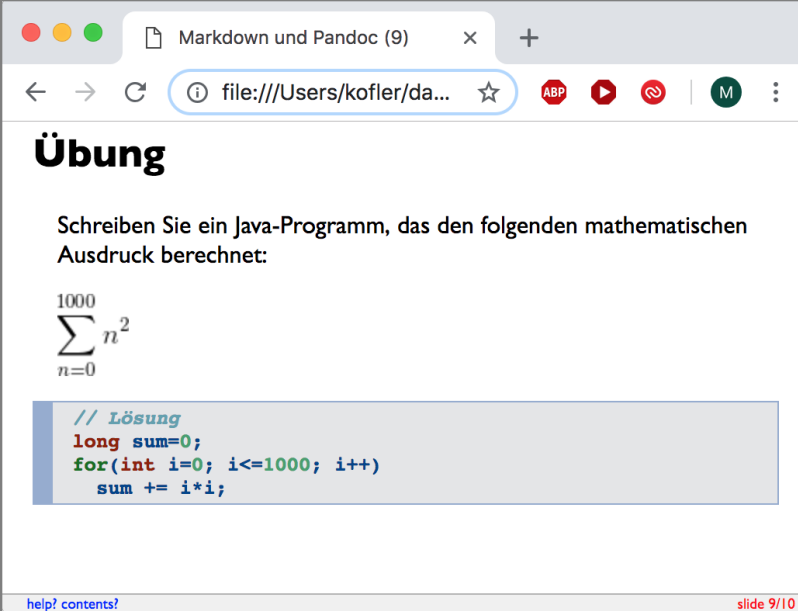
Mit den folgenden Kommandos erzeugen Sie HTML-basierte Präsentationen in den Formaten [dzslides](#), [reveal.js](#), [S5](#), [Slidy](#) bzw. [Slideous](#).

```
pandoc in.md -s -t revealjs -o out.html  
pandoc in.md -s -t dzslides -o out.html  
pandoc in.md -s -t s5 -o out.html  
pandoc in.md -s -t slidy -o out.html  
pandoc in.md -s -t slideous -o out.html
```

Die fünf HTML-Folienformate haben einen gemeinsamen Nenner: Die resultierende HTML-Seite setzt zur Präsentation JavaScript-Code voraus, der in einem lokalen Verzeichnis zur Verfügung stehen muss oder von der jeweiligen Website heruntergeladen wird.

Ich empfehle Ihnen, die fünf Formate einfach auszuprobieren. Bei den Beispieldateien zu diesem E-Book finden Sie im Verzeichnis `slides-html` ein Script, das Sie unter Linux oder macOS ausführen können. Es erzeugt aus der Datei `vortrag.md` HTML-Seiten für alle Formate, die Sie dann in einem Webbrowser miteinander vergleichen können. Beachten Sie, dass die meisten Folienformate unterschiedliche CSS-Dateien zur Formatierung anbieten. Das Erscheinungsbild kann sich damit stark ändern.

Auf Anhieb am ehesten überzeugt hat mich Slidy: Die Präsentation macht nicht den Eindruck einer verunglückten Webseite, die Navigation gelingt wahlweise per Mausklick oder mit Cursor-Tasten vollkommen unkompliziert, es gibt ein einblendbares Inhaltsverzeichnis etc.



The screenshot shows a web browser window with the title "Markdown und Pandoc (9)". The address bar shows a file path: `file:///Users/kofler/da...`. The slide content is as follows:

Übung

Schreiben Sie ein Java-Programm, das den folgenden mathematischen Ausdruck berechnet:

$$\sum_{n=0}^{1000} n^2$$

```
// Lösung
long sum=0;
for(int i=0; i<=1000; i++)
    sum += i*i;
```

At the bottom left, there is a link [help? contents?](#) and at the bottom right, it says `slide 9/10`.

Abbildung 8.1: Slidy-Vortrag in der Standardformatierung

Ein generelles Problem der meisten HTML-Formate besteht darin, dass es relativ schwierig ist, vernünftige Handouts zu erzeugen. Bei einigen Formaten können Sie den Vortrag wie eine lange Webseite ausdrucken, der Foliencharakter geht dabei aber verloren.

Standalone-Folien

Pandoc erzeugt standardmäßig sehr kompakte Präsentationen, die allerdings auf JavaScript- und CSS-Dateien aus dem Internet angewiesen sind. Eine Offline-Präsentation ist damit nicht möglich. Abhilfe schafft die Option `--self-contained`. Pandoc erzeugt dann eine HTML-Datei, in die alle referenzierten Dateien eingebettet werden – also sowohl eigene Bilder oder CSS-Dateien als auch JavaScript-Code und andere Dateien des jeweiligen Präsentationsformats. (Bei meinen Tests funktionierte `--self-contained` mit allen HTML-Formaten, nur nicht mit `reveal.js`.)

`--self-contained` ist allerdings inkompatibel mit den meisten JavaScript-Lösungen zur Darstellung mathematischer Formeln. Bewährt hat sich in meinen Tests die Kombination der Optionen `--self-contained` und `--webtex`:

```
pandoc in.md -s -t s5 --self-contained -o out.html
```

8.3 PDF-Folien (Beamer)

Mit der Option `-t beamer` greift Pandoc zur Erzeugung des PDF-Dokuments mit den Folien auf LaTeX und dessen Erweiterungspaket `beamer` zurück. Dieses Paket ist ideal, wenn Sie Ihre Folien als PDF-Datei weitergeben möchten, oder wenn Sie in den Folien mathematische Formeln verwenden möchten. Es muss Ihnen nur klar sein, dass hinter den Kulissen LaTeX zum Einsatz kommt – mit all seinen Möglichkeiten, aber auch mit all seinen Tücken und oft unbegreiflichen Fehlermeldungen.

Tipp

Wenn Sie das `beamer`-Paket nicht kennen, finden Sie hier eine gute Kurzeinführung, die natürlich selbst mit diesem Paket erstellt wurde (wenn auch vermutlich ohne Pandoc).

https://slhck.info/documents/pa.wgi_tutorial2.pdf

Normalerweise erzeugen Sie die PDF-Datei mit den Folien mit folgendem Kommando:

```
pandoc in.md -t beamer -o out.pdf
```

Wenn Sie möchten, können Sie mit `-o` auch eine `*.tex`-Zieldatei angeben. Pandoc schreibt den LaTeX-Code dorthin; anschließend erzeugen Sie daraus mit `pdflatex`, `xelatex` oder `lua1atex` selbst die PDF-Datei.

```
pandoc in.md -s -t beamer -o out.tex
pdflatex out.tex
```

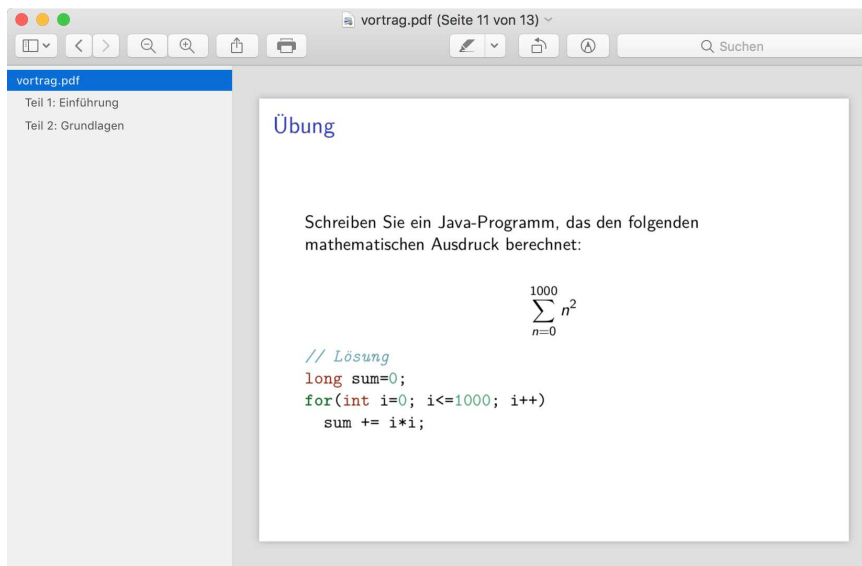


Abbildung 8.2: Eine Folie im Beamer-Defaultlayout

Die meisten im Kapitel [LaTeX und PDF](#) vorgestellten LaTeX-spezifischen Pandoc-Optionen und Template-Variablen gelten auch für Beamer-Folien. Beispielsweise erreichen Sie mit `--listings`, dass Quellcode mit dem `listings`-Paket und nicht mit dem `fancyvrb` dargestellt wird. Mit `-v lang=de` stellen Sie Deutsch als Sprache ein. Mit `-H` können Sie eine eigene LaTeX-Header-Datei angeben. Wenn Sie XeLaTeX oder LuaLaTeX verwenden möchten, geben Sie die gewünschte Engine mit `--latex-engine` an.

Die Template-Variablen `theme` und `colortheme` bestimmen, welches Beamer-Layout und -Farbschema zum Einsatz kommt. Zur Einstellung verwenden Sie `-V theme=xxx` und `-V colortheme=yyy`. Einen guten Überblick über die in Frage kommenden Layouts und Farben gibt die Website [Beamer Theme Matrix](#). Im folgenden Beispiel wird das Thema *Warsaw* mit dem Farbschema *beaver* kombiniert.

```
pandoc in.md -V theme=Warsaw -V colortheme=beaver \  
-t beamer -o out.pdf
```

Tipp

Wenn Sie beim Aufruf des Pandoc-Kommandos unverständliche Fehlermeldungen erhalten (z. B. `undefined control sequence`), dann ist womöglich eine temporäre Datei des vorigen Pandoc-Aufrufs schuld. Löschen Sie alle `.aux-`, `*.toc-` und `*.nav-`Dateien!*

Strukturierung

Pandoc wandelt Folieninhalte unterhalb des *slide levels* in block-Umgebungen um. Werfen Sie einen Blick auf den folgenden Markdown-Code für eine Folie mit `--slide-level 3`:

```
### Folienüberschrift  
  
bla bla  
  
#### Unterüberschrift  
  
noch mehr bla bla  
  
##### Unter-Unterüberschrift  
  
unwichtige Details
```

Pandoc macht daraus diesen LaTeX-Code, den ich zur besseren Lesbarkeit eingerückt habe:

```
\begin{frame}\frametitle{Folienüberschrift}
  bla bla
  \begin{block}{Unterüberschrift}
    noch mehr bla bla
    \begin{block}{Unter-Unterüberschrift}
      unwichtige Details
    \end{block}
  \end{block}
\end{frame}
```

Wie die verschachtelten Blöcke dargestellt werden, hängt stark vom Folienthema ab. Die folgende Abbildung zeigt das Thema *Warsaw*, das Blöcke in Form von runden Boxen darstellt, aber Probleme mit verschachtelten Blöcken hat.

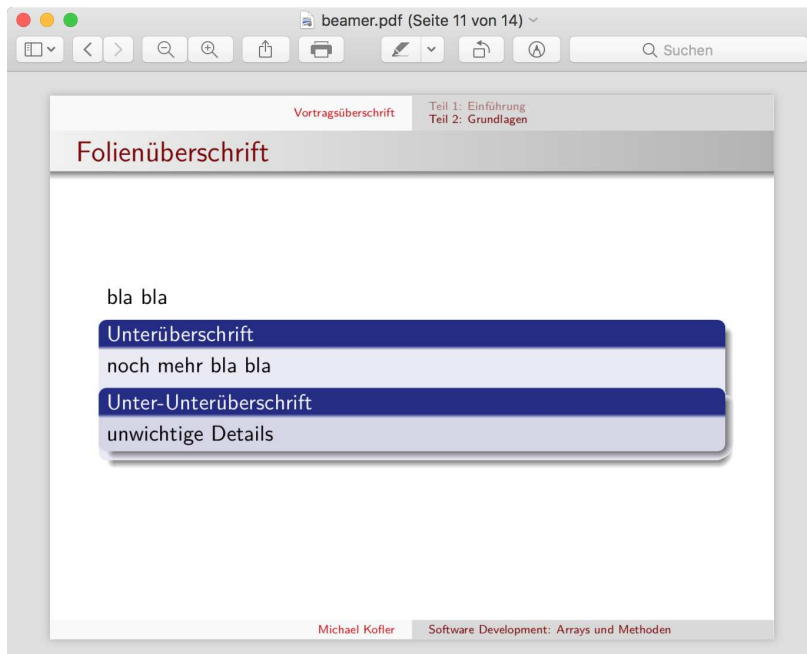


Abbildung 8.3: Darstellung von Blöcken mit dem Warsaw-Folienthema

LaTeX-Code

In die Markdown-Datei können Sie LaTeX-Code einbetten (siehe auch den Abschnitt [Mathematische Formeln und LaTeX-Code](#)). Die LaTeX-Anweisungen werden unverändert an die LaTeX-Engine weitergegeben. Somit ist es problemlos möglich, mathematische Formeln in Folien darzustellen.

Zweispaltige Folien

Das `beamer`-Paket bietet die Möglichkeit, zwei- oder dreispaltige Folien zu gestalten. In der Markdown-Syntax fehlen aber entsprechende Auszeichnungen. Um dennoch mehrspaltige Folien zu gestalten, definieren Sie in einer LaTeX-Header-Datei die folgenden neuen LaTeX-Kommandos:

```
% Datei myheader.tex
\newcommand{\colA}[1]{\begin{columns}[t]\begin{column}#{1}}
\newcommand{\colB}[1]{\end{column}\begin{column}#{1}}
\newcommand{\colEnd}{\end{column}\end{columns}}
```

In Ihrer Markdown-Datei können Sie diese Kommandos nun folgendermaßen verwenden:

```
\colA{4cm}

* eins
* zwei
* drei

\colB{7cm}

Lorem ipsum dolor sit amet ...

\colEnd
```

Beim Aufruf von Pandoc müssen Sie die Header-Datei mit der Option `-H` angeben:

```
pandoc -t beamer -H myheader.tex in.text -o out.pdf
```

Tipp

Weitere Beispiele, wie Sie das Layout von Beamer-Folien mit wenigen Zeilen LaTeX-Code optimieren können, finden Sie bei den Beispieldateien zu diesem E-Book im Verzeichnis *beamer2*.

https://kofler.info/ebooks/markdown_pandoc

Abbildungen

Abbildungen können Sie in der üblichen Markdown-Syntax einfügen. Wenn Sie keine Bildunterschrift wünschen, lassen Sie das erste eckige Klammernpaar einfach leer. Damit Pandoc die Bilder mit *Abbildung* und nicht mit *Figure* beschriftet, geben Sie die zusätzliche Pandoc-Option `-V lang=de` an.

```
![Bildunterschrift](bild.png){width=60%}
```

Handouts mit pgfpages

Die von Pandoc produzierte PDF-Datei enthält pro Seite eine Folie. Für Handouts ist es aber oft wünschenswert, mehrere Folien pro Seite auszudrucken. Am einfachsten erreichen Sie dieses Ziel, indem Sie die folgenden zwei Zeilen in eine eigene LaTeX-Header-Datei einfügen:

```
% Datei myheader.tex
\usepackage{pgfpages}
\pgfpagesuselayout{4 on 1}[a4paper,border shrink=5mm,landscape]
% optional, um jede Folie zu umranden
\pgfpageslogicalpageoptions{1}{border code=\pgfusepath{stroke}}
\pgfpageslogicalpageoptions{2}{border code=\pgfusepath{stroke}}
\pgfpageslogicalpageoptions{3}{border code=\pgfusepath{stroke}}
\pgfpageslogicalpageoptions{4}{border code=\pgfusepath{stroke}}
```

Pandoc liefert nun eine PDF-Datei, die pro Seite vier Folien enthält. Das Paket `pgfpages` ist bei der Folienanzahl allerdings recht limitiert. Die Optionen sind 2, 4, 8 oder 16 Folien pro Seite.

Tipp

Der Blog-Artikel *Handouts with notes* beschreibt eine Erweiterung des `pgfpages`-Paket, die neben jede Folie einige linierte Zeilen für eigene Anmerkungen platziert.

Handouts mit pdfpages

Wesentlich mehr Gestaltungsmöglichkeiten bietet das LaTeX-Paket `pdfpages`. Die Nutzung des Pakets ist allerdings etwas umständlicher als bei `pgfpages`. Zuerst müssen Sie mit Pandoc eine gewöhnliche PDF-Datei Ihres Vortrags erstellen, wobei Sie aber die Option `-V handout` angeben:

```
pandoc -V handout -t beamer in.text -o out.pdf
```

Im nächsten Schritt erstellen Sie die winzige Datei `handout.tex`, die wie das folgende Muster aussieht (hier für ein Layout mit 2*3 Folien):

```
\documentclass[a4paper]{article}
\usepackage{pdfpages}
\begin{document}
\includepdf[pages=1-last, nup=2x3, landscape=false,
            frame=true, noautoscale=true, scale=0.7,
            delta=0mm 5mm, offset=5mm 0]{beamer.pdf}
\end{document}
```

Dabei müssen Sie `beamer.pdf` durch den Namen der ursprünglichen PDF-Datei ersetzen. Ein Aufruf von `pdflatex` erstellt nun die Handout-Datei `handout.pdf`:

```
pdflatex handout.tex
```

Eine ausführliche Beschreibung aller Optionen gibt die [Dokumentation](#) des Pakets.

9 Fortgeschrittene Arbeitstechniken

Im Jahr 2012 habe ich mein erstes E-Book als Markdown-Text geschrieben und anschließend als EPUB- und PDF-Datei im Eigenverlag sowie als Mobi-Datei auf Amazon veröffentlicht (»Ubuntu 12.04«, ebooks.kofler).

Der nächste Meilenstein war 2014 das Buch »Raspberry Pi«, das ich zusammen mit zwei Co-Autoren für den Rheinwerk Verlag geschrieben habe. Dieses Buch war mein erster Markdown-Text, bei der vom Markdown-Text bis zur fertigen Druckdatei keinerlei manuellen Eingriffe mehr notwendig waren. Alle erforderlichen LaTeX-spezifischen Satzanweisungen, z. B. für Seitenumbrüche, habe ich direkt in die Markdown-Dateien eingebaut. (Halb-manuell sind in diesem Fall Inhaltsverzeichnis und Stichwortverzeichnis entstanden; derartige Nacharbeiten sind aber auch bei anderen Satzsystemen unvermeidlich.)

Seither habe ich alle neuen Bücher und E-Books im Markdown-Format verfasst. Pandoc zählt für mich zu den wichtigsten Werkzeugen meiner persönlichen Software-Toolbox.

Dieses Kapitel beschreibt einige fortgeschrittene Arbeitstechniken. Das Kapitel soll Ihnen Anregung geben, wie Sie Pandoc bis an seine Grenzen ausreizen können. Ob das immer sinnvoll ist, sei dahingestellt; für mich hat sich die Vorgehensweise auf jeden Fall bewährt, und vielleicht finden Sie in dem Kapitel die eine oder andere Anregung, die sich auch für Ihren Einsatz von Pandoc eignet.

Das Kapitel endet mit einem Abschnitt zum gemeinsamen Verfassen von Markdown-Texten unter Zuhilfenahme von GitLab. Diesen Abschnitt haben Axel Dürkop und Tina Ladwig beigesteuert – vielen Dank dafür!

9.1 Atom und Pandoc kombinieren

Meine Markdown-Begeisterung hat dazu geführt, dass ich eine ganze Reihe von Co-Autoren dazu genötigt habe, ihre Texte ebenfalls in der Markdown-Syntax zu schreiben. Dabei wurde ich immer wieder mit der Frage konfrontiert: »Mit welchem Editor soll ich den Text denn nun schreiben?«

Ich habe es schon erwähnt, dass ich dieses E-Book wie die meisten meiner Bücher mit dem Editor *Emacs* verfasst habe. Aber einem Markdown-Einsteiger auch noch Emacs mit seiner sehr gewöhnungsbedürftigen Benutzeroberfläche zuzumuten – das hieße den Bogen zu überspannen :-)

Stattdessen empfehle ich das Programm *Atom* (<https://atom.io>). Dieser Editor wird ebenfalls höchsten Ansprüchen gerecht, lässt sich quasi unbegrenzt erweitern und konfigurieren – und ist dennoch vergleichsweise einsteigertauglich.

Alternative VSCode

Im Prinzip gilt die vorangegangene Lobeshymne uneingeschränkt auch für VSCode (<https://code.visualstudio.com>). Mit diesem von Microsoft entwickelten Editor machen Sie ebenfalls nichts verkehrt. Es ist primär eine Geschmacksfrage, welcher Editor mehr zusagt. Aber diese Anleitung gilt nun einmal für Atom.

Im Folgenden gehe ich davon aus, dass Sie Atom installiert und sich mit den Grundfunktionen vertraut gemacht haben. Eine gute Einführung in den Umgang mit Atom finden Sie im sogenannten [Flight Manual](#).

Absatz neu umbrechen

Mit der Tastenkombination *Strg+Alt+Q* bzw. *cmd+alt+Q* (macOS) wird der markierte Text bzw. der aktuelle Absatz neu umgebrochen. Das ist praktisch, wenn Sie Änderungen am Text durchgeführt haben und nun manche Zeilen zu kurz, andere zu lang sind. Die Tastenkombination stellt sicher, dass keine Zeile mehr als 80 Zeichen lang ist.

Atom-Paketverwaltung

Atom ist ein modular aufgebauter Editor. Alle Funktionen mit Ausnahme elementarer Grundfunktionen sind in Form von Paketen realisiert. Standardmäßig sind bereits 80 Core-Pakete installiert. Welche das sind, verrät *File/Settings* bzw. *Atom/Preferences* (macOS) im Dialogblatt *Packages*.

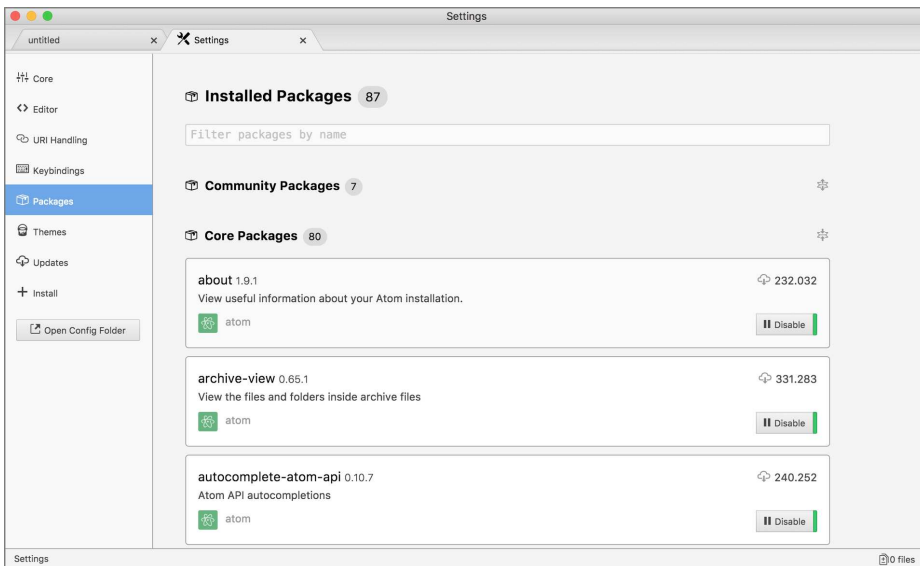


Abbildung 9.1: Liste der installierten Packages in Atom unter macOS

Neue Pakete und Themes (die das Aussehen von Atom beeinflussen) können Sie im Dialogblatt *Install* der Einstellungen suchen und installieren. Dabei haben Sie aktuell die Wahl aus fast **8000 Paketen**.

Pakete können nach der Installation mühelos wieder entfernt oder auch nur deaktiviert werden. Außerdem sehen viele Pakete einen Settings-Button vor, um Einstellungen des Pakets zu verändern. Zur Bedienung der Pakete finden Sie im Menü *Packages* entsprechende Kommandos. In diesem Menü sind die Pakete alphabetisch geordnet. Neu installierte Pakete werden anfänglich am Ende des Menüs platziert, ab dem nächsten Neustart von Atom aber wieder in alphabetischer Reihenfolge angeordnet.

Markdown-Unterstützung ohne Konfiguration

Grundsätzlich kommt Atom mit Markdown-Dateien standardmäßig zurecht. Bei Dateien mit der Endung *.md führt Atom automatisch ein ganz brauchbares Syntax-Highlighting durch. Die im Standardpaket *markdown-preview* definierte Tastenkombination *Shift+Ctrl+M* öffnet neben dem Text eine Preview-Ansicht, die ebenfalls für viele Zwecke ganz passabel ist. Damit übertrifft Atom quasi aus dem Stand und ohne jede Konfigurationsarbeit den Funktionsumfang vieler Markdown-Editoren. Pandoc-spezifische Erweiterungen wie Tabellen überfordern die Standard-Preview-Funktion allerdings.

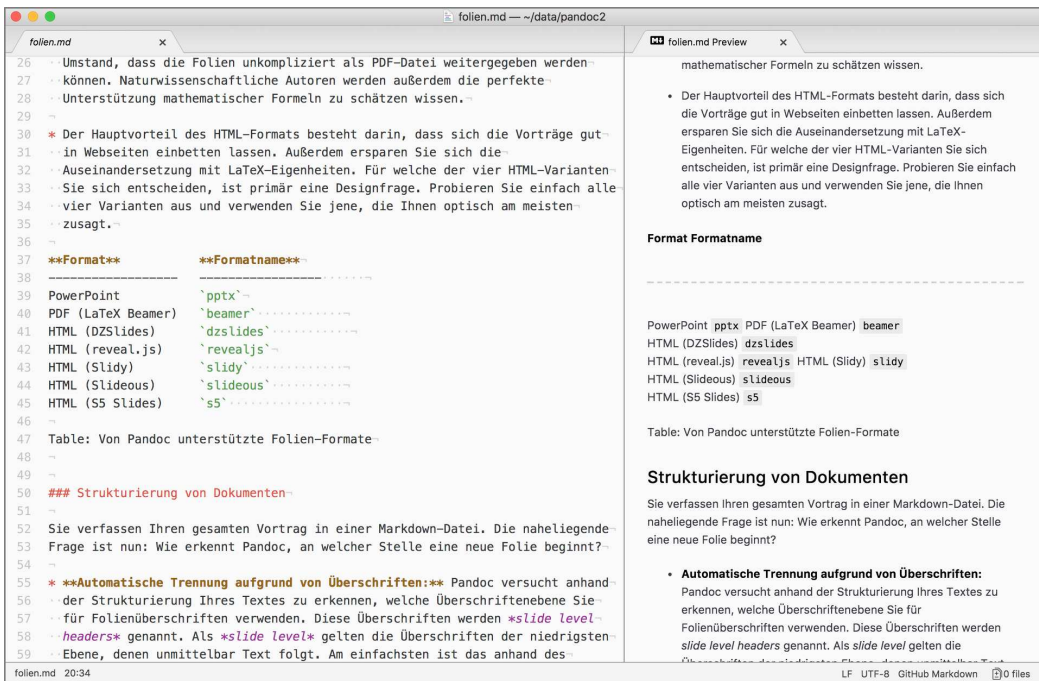


Abbildung 9.2: Die Markdown-Vorschau kommt nicht mit Pandoc-Erweiterungen zurecht

Markdown-Preview-Enhanced und Pandoc

Das automatisch installierte Paket *markdown-preview* verwendet für die Preview-Funktion die GFM-Syntax (GitHub Flavored Markdown). Alle Pandoc-spezifischen Erweiterungen werden daher nicht oder nur fehlerhaft dargestellt. Eine wesentlich bessere Vorschau gelingt mit dem Paket *markdown-preview-enhanced*. Nachdem Sie dieses Paket installiert haben, müssen Sie das standardmäßig aktive Package *markdown-preview* deaktivieren (Button *Disable* in der Paketverwaltung).

Der entscheidende Vorteil von *markdown-preview-enhanced* besteht darin, dass es anstatt der simplen GitHub-Markdown-Engine auch Pandoc aufrufen kann. Dazu müssen Sie jedoch einige Optionen des Pakets verändern.

- Die Option *Break in Single New Line* deaktivieren Sie.
- Dafür aktivieren Sie die Option *Use Pandoc Parser*.
- Unter macOS und Windows müssen Sie im Feld *Pandoc Options: Path* den vollständigen Pfad des pandoc-Kommandos angeben. Unter Windows findet das Paket pandoc in der Regel selbst. (Bei aktuellen Installationen ist `C:\Users\<username>\AppData\Local\Pandoc\pandoc.exe` der übliche Pfad.)
- Im Feld *Pandoc Options: Markdown Flavor* geben Sie an, welche Markdown-Variante pandoc verwenden soll. Standardmäßig gilt `markdown-raw_tex+tex_math_single_backslash`. Oft funktioniert die simple Einstellung `markdown` noch besser. Damit kommt der Standard-Markdown-Dialekt von Pandoc zum Einsatz.

Tipps, wie Sie Einfluss auf den von der Vorschau verwendeten CSS-Code nehmen können, finden Sie in der [GitHub-Dokumentation](#) des Pakets.

9.2 Der Satz dieses E-Books

Die EPUB- und Mobi-Ausgaben dieses E-Books habe ich so erzeugt, wie ich dies in den Kapiteln [HTML-Dokumente](#) und [E-Books im EPUB- und Mobi-Format](#) beschrieben habe. Der einzige Unterschied besteht darin, dass ich mit einem sed-Script vor dem Aufruf von Pandoc einige LaTeX-Satzanweisungen aus dem Markdown-Quelltext entferne.

Beispieldateien

Auf meiner [Website](#) finden Sie Beispieldateien zu diesem E-Book. Das Verzeichnis *latex2* enthält dabei die in diesem Abschnitt beschriebenen Scripts sowie alle weiteren Dateien, die zum Satz des ersten Kapitels dieses E-Books erforderlich sind. Aus Platzgründen habe ich darauf verzichtet, den gesamten Code hier wiederzugeben.

Die PDF-Version des Buchs verwendet *FF-DIN-Schriften*. Da es sich dabei um kommerzielle Schriften handelt, fehlen die entsprechenden Dateien in den Beispieldateien.

PDF-Version

Im Gegensatz zu Amazon und Apple bin ich der Meinung, dass die Leser eines E-Books zumindest die Option auf eine ordentliche PDF-Ausgabe haben sollten. Das gilt insbesondere für (IT-)Fachbücher mit Listings, Tabellen usw.

Im Kapitel [LaTeX und PDF](#) habe ich aber schon darauf hingewiesen, dass es nicht ganz einfach ist, aus Markdown-Dateien ästhetisch ansprechende PDFs zu machen. Im Folgenden beschreibe ich Ihnen, wie ich vorgegangen bin, um die PDF-Version dieses E-Books zu erzeugen.

Pandoc- und LaTeX-Aufruf trennen

Wenn Sie Pandoc mit der Option `-o out.pdf` aufrufen, wandelt Pandoc Ihren Markdown-Text in ein LaTeX-Dokument um und erzeugt daraus mit `pdflatex` direkt eine PDF-Datei. Ich gehe ein wenig anders vor:

- Bevor ich Pandoc aufrufe, führe ich mit einem sed-Kommando einige automatisierte Änderungen an den Markdown-Dateien durch.

- `pandoc ... -o alltext.tex` erzeugt nun eine LaTeX-Datei für alle Kapitel des Buchs. Ich rufe Pandoc dabei *ohne* die Option `-s` aus, damit ich die LaTeX-Datei mit eigenen Header-Dateien ausstatten kann.
- Mit einem weiteren `sed`-Kommando sowie einigen Python-Scripts optimiere ich den LaTeX-Code nun für meine Zwecke und kopiere die resultierende Datei in das Unterverzeichnis `latex`.
- Dort befindet sich die Datei `buch.tex`. Sie enthält Import-Anweisungen für diverse Header-Dateien sowie für `alltext.tex`. Mit `latex buch` erzeuge ich daraus eine DVI-Datei, mit `dvips` eine PostScript-Datei und mit `ps2pdf` schließlich eine PDF-Datei.

Das Script, das im Verzeichnis mit meinen Markdown-Dateien Pandoc aufruft, sieht wie folgt aus. Der seltsame Code ab `2>&1` bewirkt, dass Pandoc-Warnungen durch `grep` geleitet werden. Alle Warnungen, dass eine mit `\input` zu inkludierende Datei fehlt, werden dabei herausgefiltert. (Die betreffenden `*.tex`-Dateien befinden sich im Unterverzeichnis `latex`. Dass Pandoc die Dateien liest, war nie beabsichtigt. Die Warnung ist neu seit Pandoc 2.3.)

```
#!/bin/bash
# Script pr (pandoc run)
chaps="title vorwort intro tools syntax pandoc \
      html ebooks latex folien techniken"
pchaps=""
# Preprocessing, Variable lchaps mit den
# Dateinamen aller Pandoc-Dateien zusammensetzen
lchaps=""
for fn in $chaps; do
    ./sed-before-pandoc-for-latex $fn.md tmp/$fn.pandoc
    lchaps="$lchaps tmp/$fn.pandoc"
done

# Pandoc aufrufen, dabei include-Warnungen unterdrücken
pandoc -t latex --listings --top-level-division=chapter \
    $lchaps -o latex/tmp.tmp.tex 2>&1 >/dev/null | grep -v 'include'

# Postprocessing, Ergebnis -> latex/alltext.tex
./sed-after-pandoc-for-latex latex/tmp.tmp.tex latex/alltext.tex
```

Ein zweites Script befindet sich im Unterverzeichnis `latex`. Es erzeugt aus `alltext.tex` die PDF-Datei:

```
#!/bin/bash
# Script lr (LaTeX run)
latex buch.tex && dvips buch -o buch.ps && ps2pdf buch.ps
```

LaTeX versus PDFLaTeX

Der Hauptgrund, weswegen ich `latex` dem üblicheren Kommando `pdflatex` vorziehe, liegt darin, dass ich in meinen Header-Dateien auf das LaTeX-Paket `ps tricks` zurückgreife. Damit kann ich unkompliziert Linien und Boxen zeichnen, was ich unter anderem zur Gestaltung der Kopf- und Fußzeilen verwendet habe. `ps tricks` ist aber leider zu `pdflatex` inkompatibel. Bisher hat mir aber die Zeit gefehlt, mich in ein anderes, `pdflatex`-kompatibles Paket für derartige Zwecke einzuarbeiten.

Preprocessing

Das Script `sed-before-pandoc-for-latex` eliminiert im Markdown-Text alle Passagen, die mit `\dropinlatex` beginnen mit `\dropinlatexend` enden.

```
#!/bin/bash
# Script sed-before-pandoc-for-latex
sed -e '/\\dropinlatex/,/\\dropinlatexend/d' < $1 > $2
```

Der Sinn dieses Scripts besteht darin, dass ich so im Markdown-Code Text formulieren kann, die nur für die HTML-/EPUB-/Mobi-Version gilt. Den entsprechenden LaTeX-Text schreibe ich in eine eigene `*.tex`-Datei, die ich mit `\input{name.tex}` importiere.

Zwei unterschiedliche Textzweige zu warten, ist natürlich mühsam. In seltenen Fällen, z. B. wenn sich eine Tabelle in der Markdown-Syntax nicht perfekt gestalten lässt, kann ich so für die HTML- und für die LaTeX-Variante jeweils optimalen Code nutzen.

Postprocessing

Wesentlich mehr gibt es in `sed-after-pandoc-for-html` zu tun:

- Es ersetzt alle Dateiangaben der Form `../bilder/name.eps` durch `.././bilder/name.eps`. Die Veränderung des Pfads ist notwendig, damit LaTeX die Bilder findet, obwohl das Kommando relativ zum Verzeichnis mit dem Markdown-Dateien im Unterverzeichnis `latex` aufgerufen wird.

LaTeX kommt im Gegensatz zu PDFLaTeX nicht mit `*.png`-Bildern zurecht, sondern erwartet `*.eps`-Dateien. Um aus `*.png`-Dateien geeignete `*.eps`-Dateien zu machen, verwende ich natürlich ebenfalls ein Script (siehe Verzeichnis `sonstiges` in den Beispieldateien).

- Es ersetzt `\textasciitilde` durch den LaTeX-Code `\~` für eine weiche Trennung. Damit kann ich in Wörtern, die LaTeX nicht so trennt wie ich das möchte, eine manuelle Trennung in der Form `Trenn~zeichen` einbauen. (Bei aktuellen Pandoc- und LaTeX-Versionen funktioniert auch das Unicode-Zeichen `U+00AD`, bei älteren Versionen ist das aber nicht der Fall.)
- Diverse Suchen- und Ersetzen-Anweisungen optimieren die Abstände in Abkürzungen und bauen z. B. zwischen `»z.«` und `»B.«` mit dem LaTeX-Code `\,` einen fixen, kleinen Abstand ein.
- Damit in Markdown mit `>` eingerückter Text so angezeigt wird, wie ich mir das wünsche, ersetze zwei weitere `sed`-Anweisungen die Kommandos zum Start und Ende einer `quote`-Umgebung durch entsprechende `quoting`-Kommandos.
- Die `sed`-Kommandos zur Verbesserung der Darstellung von Tabellen und Bildern habe ich im Kapitel [LaTeX und PDF](#) schon erläutert. Das gilt auch für die `sed`-Kommandos, die einen Bug des `listings`-Pakets umgehen.
- Das Python-Script `caption-below.py` baut den von Pandoc erzeugten LaTeX-Code von Tabellen so um, dass Tabellenüberschriften zu Tabellenunterschriften werden.
- Das Script `join-quoting.py` verbindet mehrere `quoting`-Umgebungen zu einer einzigen derartigen Umgebung.

```
#!/bin/bash
# Script sed-after-pandoc-for-latex

sed -E -e 's,bilder/(.*)\.\.{3},../bilder/\1.eps,' \
-e 's,\\textasciitilde\\{-,\\-,g' `# Trennoption` \
-e 's/z\.B\. /z.\\,B./' `# Abstände in` \
-e 's/z\\.\\,B\\. /z.\\,B.\\ /' `# Abkürzungen` \
-e 's,zB\\. ,z.B.,g' \
-e 's/d\\.h\\. /d.\\,h./' \
-e 's/d\\.\\,h\\. /d.\\,h.\\ /' \
-e 's/u\\.a\\. /u.\\,a./' \
-e 's,\\mylabel\\{(.*\\)\\},\\}\\mylabel\\{\\1\\},' \
`# Einrückungen` \
-e 's,\\begin\\{quote\\},\\begin\\{quoting\\},' \
-e 's,\\end\\{quote\\},\\end\\{quoting\\},' \
`# Tabellen` \
-e 's,\\@\\{\\},,g' \
-e 's,\\toprule,\\hline,' \
-e 's,\\bottomrule,\\hline,' \
-e 's,\\midrule,\\hline,' \
-e '/\\endfirsthead/,/\\endhead/d'
`# Bilder` \
-e 's,\\begin\\{figure\\},\\begin\\{figure\\}[h],' \
-e 's,^\\centering,\\raggedright,' \
`# Bug im listings-Paket` \
-e 's,\\lstinline!\\",\\lstinline!\\\\" ,g' \
-e 's,\\lstinline!ä,\\lstinline!\\\"ä,g' \
etc. für öüÄÖüß \
< $1 > tmp/tmp2.tex

# Bildüberschrift -> Bildunterschrift
./caption-below.py < tmp/tmp2.tex > tmp/tmp3.tex

# mehrere quoting-Umgebungen zu einer verbinden
./join-quoting.py < tmp/tmp3.tex > $2
```

LaTeX-Dateien

Im Verzeichnis latex befindet sich die Datei buch.tex, die die vom obigen Script erzeugte Datei alltext.txt mit anderen LaTeX-Dateien kombiniert:

```
% Datei buch.tex
\input{header} % Header-Datei lesen
\begin{document} % Start des Dokuments

% Inhaltsverzeichnis erzeugen
\markboth{Inhaltsverzeichnis}{Inhaltsverzeichnis}
\tableofcontents

% alltext.tex lesen
\input{alltext}
\end{document}
```

header.tex enthält den Code, um diverse LaTeX-Erweiterungspakete zu laden und zu konfigurieren. Der Code in header.tex bestimmt beispielsweise die Seitengröße und die Kopfzeilen, die verwendeten Schriftarten usw.

LaTeX-Code in den Markdown-Dateien

In der Schlussphase eines E-Books oder Buchs versuche ich, das Layout der einzelnen Seiten zu optimieren. Dazu baue ich in die Markdown-Dateien an geeigneten Stellen `\linebreak`- oder `\newpage`-Anweisungen ein, um den Umbruch bestmöglich zu gestalten. Für die EPUB-/Mobi-Variante des Buchs eliminiert Pandoc diese LaTeX-spezifischen Anweisungen automatisch, so dass sie dort nicht stören.

Automatischer Aufruf der Scripts

Die beiden Scripts `pr` und `latex/1r` werden automatisch gestartet, sobald eine Markdown-Datei gespeichert wird. Dazu laufen in zwei Terminal-Fenstern im Hintergrund die Scripts `prauto` bzw. `1rauto`, wobei ich hier nur auf `prauto` näher eingehen werde. `1rauto` funktioniert analog.

`prauto` wartet darauf, dass `ls -l *.md` vom bisherigen Aufruf abweichende Informationen liefert und startet dann `pr`. Das Kommando `sleep` verhindert, dass das Script unnötig die CPU aufheizt.

```
#!/bin/bash
# wartet, bis sich eine *.md-Datei ändert,
# und ruft dann automatisch ./pr auf
tmp2=$(ls -l --time-style=full-iso *.md)
while true
do
  if [ "$tmp1" != "$tmp2" ]; then
    tmp2=$(ls -l --time-style=full-iso *.md)
    ./pr
  else
    sleep 1
  fi
  tmp1=$(ls -l --time-style=full-iso *.md)
done
```

Der Satz von »echten« Büchern

Noch komplizierter ist der Satz »echter« Bücher, weil nun auch die Layoutrichtlinien eines Verlags zu beachten sind. Dazu können z. B. die folgenden Punkte zählen:

- eigenes Seitenlayout, aufwendigere Gestaltung von Kopf- und Fußzeilen, des Inhaltsverzeichnisses, der Tabellen etc.
- Verwendung anderer Fonts
- Gestaltung von Querverweise in der Form »siehe Abschnitt 17.2«

- eigene Formatierung für Menükommandos (Kapitälischen)
- eigene Schriftart für Tastenkürzel

All das erfordert noch mehr Code in der LaTeX-Header-Datei sowie in den Scripts zur Nachbearbeitung des von Pandoc erzeugten LaTeX-Codes.

9.3 Docker-Container für Pandoc

Pandoc wird seit Jahren stetig weiterentwickelt. Das ist an sich erfreulich, allerdings führen die Updates immer wieder zu Inkompatibilitäten, insbesondere dann, wenn Sie komplizierte Setups verwenden und den von Pandoc erzeugten HTML- oder LaTeX-Code mit eigenen Scripts nachträglich manipulieren.

Für mich ist das ein großes Problem: Ich muss während der Lebenszeit eines Buchs, also zumeist über mehrere Jahre, in der Lage sein, für einen Nachdruck neue, geringfügig korrigierte Druckdateien zu erzeugen. Während dieses Zeitraums ist ein Pandoc-Versionswechsel undenkbar. Gleichzeitig würde ich bei neuen Projekten aber gerne die gerade aktuellste Pandoc-Version verwenden.

Ein Ausweg aus diesem Dilemma wäre die Verwendung virtueller Linux-Maschinen, in denen jeweils Pandoc, LaTeX und alle anderen für mich erforderlichen Werkzeuge installiert sind. Mittlerweile gibt es aber einen noch eleganteren Weg: Ich bin dazu übergegangen, die gesamte aus Pandoc und LaTeX bestehende Werkzeugsammlung in Form von Docker-Containern einzurichten. Das hat gleich noch einen zweiten Vorteil: Die Pandoc-Arbeitsumgebung lässt sich mit minimalem Aufwand auf unterschiedlichen Rechnern und unter diversen Betriebssystemen einrichten.

Der offensichtliche Nachteil besteht darin, dass die hier beschriebene Vorgehensweise sowohl die Installation von Docker als auch ein grundlegendes Verständnis dafür voraussetzt, wie Docker funktioniert. Dieses Wissen kann ich hier nicht vermitteln. Gegebenenfalls müssen Sie sich also auf der [Docker-Website](#) einlesen oder sich ein Buch zu Docker besorgen.

Dockerfile

Für mein Pandoc-Image verwende ich das folgende Dockerfile:

```
# Datei Dockerfile
FROM haskell

# Pakete installieren
RUN apt-get update -y && \
    apt-get install -y -o Acquire::Retries=10 \
        --no-install-recommends \
        texlive-latex-recommended \
        texlive-latex-extra \
        texlive-fonts-recommended \
        texlive-lang-german \
        texlive-pstricks \
        texlive-font-utils \
        lmodern \
        imagemagick \
        unzip \
        python3 \
        ghostscript \
        subversion \
        joe \
        vim \
        less && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Pandoc installieren
RUN cabal update && \
    cabal install pandoc-2.3.1 && \
    ln -s /root/.cabal/bin/pandoc /usr/bin/pandoc

# Mitteleuropäische Zeitzone
# (siehe https://serverfault.com/questions/683605)
RUN cp /usr/share/zoneinfo/Europe/Berlin /etc/localtime
```

```
# Fonts für LaTeX installieren
ADD myfonts.tgz /usr/local/share/texmf
RUN texhash

# Volume /data, bei docker run mit dem Arbeitsverzeichnis
# verbinden, also: docker run -v $(pwd):/data
WORKDIR /data
VOLUME ["/data"]

ENTRYPOINT ["/bin/bash"]
```

Die Liste der zu installierenden Pakete werden Sie natürlich Ihren eigenen Bedürfnissen entsprechend anpassen. Wenn Sie bei der Zeile `cabal install pandoc-n.n.n` die Versionsnummer weglassen (also `cabal install pandoc`), dann kommt automatisch die gerade aktuellste Pandoc-Version zum Einsatz.

Die beiden Zeilen `ADD myfonts ...` und `RUN texhash` sind nur dann relevant, wenn Sie eigene, LaTeX-taugliche Fonts besitzen. Die Datei `myfonts.tgz` muss sich im gleichen Verzeichnis wie das `Dockerfile` befinden.

Image erzeugen

Ausgehend von diesem `Dockerfile` müssen Sie nun einmalig das daraus resultierende Image erzeugen. Dazu führen Sie dieses Kommando aus:

```
docker build . -t pandoc_23
```

Dabei bezeichnet der nach `-t` angegebene Name das Image. Das Erzeugen des Images beansprucht je nach Hardware ca. 15 bis 25 Minuten. (Besonders aufwendig ist `cabal install pandoc`, weil im Zuge dieses Kommandos diverse Pakete heruntergeladen und kompiliert werden.)

Der Build-Prozess erfordert viel RAM

Damit der Build-Prozess fehlerfrei durchläuft, muss Docker über genug RAM verfügen. Unter Windows und macOS ist Docker zumeist so konfiguriert, dass es maximal 2 GByte RAM nutzen darf. Das ist zu wenig und führt zur folgenden Meldung: »ExitFailure (-9). This may be due to an out-of-memory condition.«

Abhilfe: Erhöhen Sie das RAM-Limit für Docker auf 4 GByte. (Für die spätere Ausführung der Container reichen die üblichen 2 GByte vollkommen. Sie können das Limit also später wieder zurücksetzen, wenn Sie möchten.)

Container erzeugen und starten

Ist das Image einmal fertig, können daraus beliebig viele Container erzeugt werden. Das geht zum Glück blitzschnell. Sie wechseln dazu in Ihr Markdown-Projektverzeichnis und führen `docker run` aus:

```
cd meinprojektverzeichnis
docker run -it -v $(pwd):/data \
  --name meinprojekt -h meinprojekt pandoc_23
```

Kurz eine Erklärung des Kommandos:

- `run -it` erzeugt einen Container für die interaktive Benutzung und startet ihn sofort.
- `-v $(pwd):/data` verbindet das gerade aktuelle Verzeichnis mit dem Volume-Verzeichnis `/data` des Containers.
- `--name meinprojekt` gibt dem Container einen Namen.
- `-h meinprojekt` weist dem Container einen Hostnamen zu. Wenn Sie auf die Option verzichten, erzeugt Docker einen zufälligen Hostnamen. Da der Hostname im Kommandoprompt angezeigt wird, erleichtert der Hostname die Unterscheidung zwischen mehreren laufenden Containern. Insofern ist es eine gute Idee, denselben Namen wie bei `--name` zu verwenden.
- Der letzte Parameter gibt den Image-Namen an. Er muss mit dem Namen übereinstimmen, den Sie mit `-t` bei `docker build` angegeben haben.

Der neue Container läuft nun in einem Terminal- oder cmd.exe-Fenster. ls sollte dieselben Dateien wie in Ihrem Projektverzeichnis zeigen. pandoc -v liefert die Pandoc-Versionsnummer.

```
root@pa23:/data# ls *.md
folien.md  intro.md  pandoc.md ...

root@pa23:/data# pandoc -v
pandoc 2.3 ...
```

Mit *Strg+D* beenden Sie die Ausführung des Containers. Um den vorhandenen Container später neu zu starten, verwenden Sie das im Folgenden angegebene Kommando. projektname ist dabei der Name, den Sie dem Container zuvor mit `docker run ... --name` gegeben haben.

```
docker start -ai projektname
```

Sie können mehrere Instanzen des Containers gleichzeitig in mehreren Terminal-Fenstern nutzen. Sofern bereits eine Instanz läuft, starten Sie weitere Instanzen wie folgt:

```
docker exec -it projektname bash
```

9.4 Kollaborativ schreiben mit GitLab

Quellenangabe

Wie bereits in der Einleitung des Kapitels erwähnt, wurde dieser Abschnitt von Axel Dürkop und Tina Ladwig (beide tätig an der TU Hamburg) verfasst. Der Abschnitt ist eine wunderbare Ergänzung zu diesem Kapitel und zeigt, wie Pandoc im wissenschaftlichen Umfeld eingesetzt werden kann.

Docker spielt seine Stärke nicht nur im lokalen Einsatz auf Ihrem Rechner aus, wie Sie es im vorigen Abschnitt kennengelernt haben. Das gleiche Docker-Image mit Pandoc können Sie auch verwenden, um Texte gemeinsam mit Unterstützung von Git zu schreiben. So können z. B. Lehrbücher entstehen, zu denen verschiedene Autorinnen und Autoren beitragen.

Wenn diese Werke dann wiederum unter [Creative-Commons-Lizenzen](#) gestellt werden, handelt es sich um [Open Educational Resources \(OER\)](#). Der Clou bei der Arbeit mit Markdown und Pandoc ist, dass auch der »Quellcode« von Lehrmaterialien zur Verfügung gestellt werden kann. So haben andere die Möglichkeit zu deren Änderung und Erweiterung. An der TU Hamburg haben wir in diesem Sinne und mit dem hier beschriebenen Vorgehen z. B. das Buch *Ethics and Technology: Some Issues* verfasst (siehe <https://doi.org/10.15480/882.1570>).

Ausgehend von diesem Gedanken werden Sie in den folgenden Abschnitten einen Workflow kennenlernen, bei dem Markdown-Dateien als »Quellcode« verstanden werden, aus dem die Zielformate mit Pandoc entstehen. Um kollaborativ am Quellcode arbeiten zu können, erweitern wir das Trio Markdown, Pandoc und Docker noch um den Mitspieler *GitLab*. GitLab ist wie GitHub eine serverbasierte Software, die zum gemeinsamen Arbeiten und Teilen von Quellcode gedacht ist. Ein entscheidender Unterschied zu GitHub besteht darin, dass es sich bei GitLab um freie Software handelt, die auch auf einem eigenen Server installiert werden kann.

Das große Ganze

Bevor wir die einzelnen Schritte durchgehen, werfen wir einen Blick auf den Gesamtzusammenhang des Workflows: Mehrere Autorinnen und Autoren arbeiten gemeinsam an Texten in GitLab. Wann immer sie in GitLab an den Markdown-Dateien eine Änderung oder Ergänzung vornehmen, wird das Zielprodukt (z. B. ein PDF-Dokument), neu generiert.

Um diesen Ablauf zu koordinieren, assistiert eine Software mit dem Namen *GitLab Runner*. Dieser GitLab Runner verwendet ein Docker-Image, in dem Pandoc installiert ist und das auf der Plattform *Docker Hub* zur Verfügung gestellt wird. Nachdem er den Quellcode geklont hat, startet der GitLab Runner einen Pandoc-Container, konvertiert mit dessen Hilfe den Quellcode zum Zieldokument und lädt dieses wieder in GitLab hoch. Der beschriebene Vorgang dauert je nach Konfiguration des Systems in der Regel nur Sekunden. Die folgende Abbildung zeigt die einzelnen Schritte in einem Sequenzdiagramm.

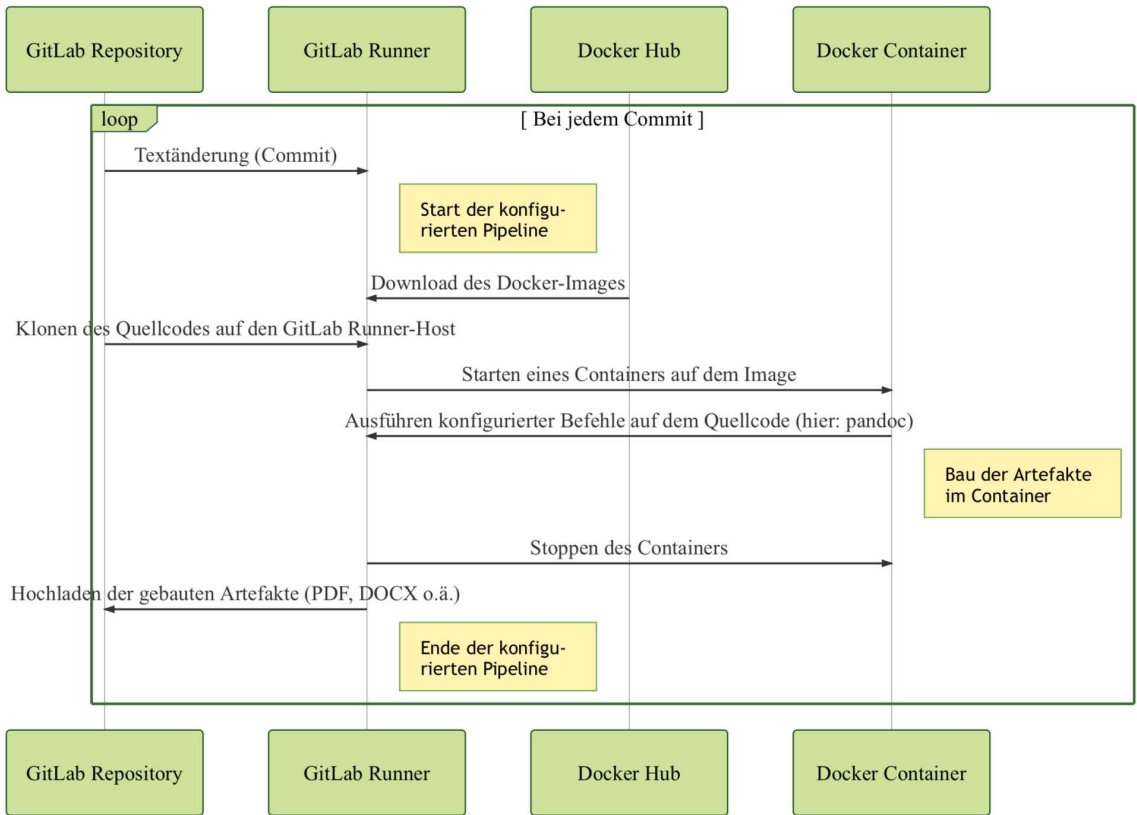


Abbildung 9.3: Ablauf des Konvertierungsvorgangs in GitLab

Bereitstellen des Docker-Images auf Docker Hub

Damit der beschriebene Workflow funktioniert, muss unser Docker-Image online verfügbar sein. Dies kann über die Plattform Docker Hub geschehen, wo es unter der folgenden Adresse zu finden ist:

https://hub.docker.com/r/xldrpk/pandoc_23_auto

Bei gitlab.com anmelden

GitLab ist freie Software und kann in der Community Edition kostenlos auf eigenen Servern installiert werden. Wir empfehlen Ihnen jedoch, zunächst einen Account auf gitlab.com anzulegen und die folgenden Schritte dort nachzuvollziehen.

Erstellen Sie nach der Registrierung ein neues Projekt mit dem Namen kollaborativeschreiben. Initialisieren Sie es auch mit einer README-Datei, in der Sie später Ihr Projekt genauer beschreiben können.

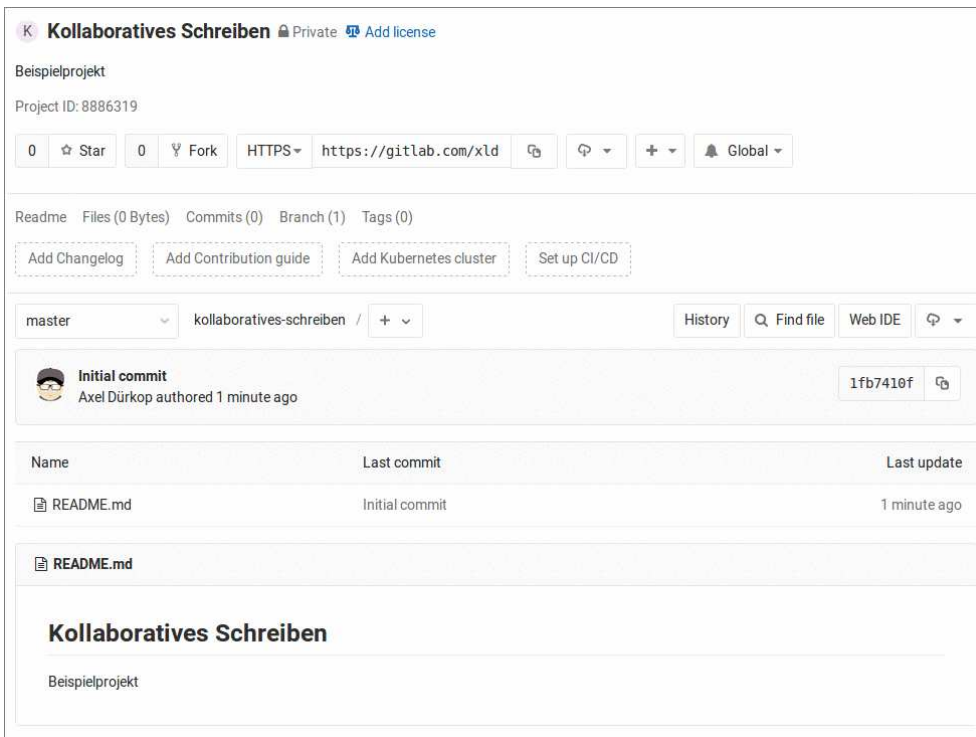


Abbildung 9.4: GitLab-Projekt nach dem Einrichten

Mit dem GitLab-Projekt und dem Docker-Image in Stellung können Sie nun beginnen, die Dokumentstruktur einzurichten. Dies kann einfach über die Browseroberfläche erfolgen, tiefere Kenntnisse von Git sind zunächst nicht erforderlich. Um die Kollaboration komplett auszureizen, sollten Sie sich später auch mit den Konzepten von Branches, Merge Requests und dem Pushen und Pullen vom eigenen Rechner beschäftigen.

GitLab-Repository einrichten

Stellen Sie sich für die kommenden Schritte vor, dass GitLab ein Content-Management-System (CMS) ist, vergleichbar mit Wordpress, TYPO3 oder Joomla. Über verschiedene Button und Formularfelder verwalten Sie Ihre Inhalte direkt im Browser.

In diesem Sinne legen Sie in GitLab die Markdown-Dateien an, aus denen Sie ein PDF generieren wollen. Um eine mögliche Arbeitsteilung anzudeuten, erstellen Sie mindestens die drei Dateien 01_einfuehrung.md, 02_hauptteil.md und 03_schluss.md. Notieren Sie in jeder eine Überschrift zweiter Ordnung und ein wenig [Blindtext](#).





Name	Last commit	Last update
 01_einfuehrung.md	Update 01_einfuehrung.md	1 minute ago
 02_hauptteil.md	Add new file	1 minute ago
 03_schluss.md	Add new file	37 seconds ago
 README.md	Initial commit	13 minutes ago

Abbildung 9.5: Ansicht der angelegten Dateien in GitLab

Konfiguration des GitLab-Runners

Damit aus diesen Markdown-Dateien nun ein PDF generiert wird, muss noch eine entsprechende Konfiguration im Repository erstellt werden. Dies erfolgt in der Datei `.gitlab-ci.yml`, die Sie ebenfalls einmalig anlegen müssen.

```
image: xldrkp/pandoc_23_auto

build:
  script:
    - pandoc *.md -o bericht.pdf
    - pandoc *.md -o bericht.odt
  artifacts:
    paths:
      - "*.pdf"
      - "*.odt"
```


Diese Konfiguration sagt dem GitLab-Runner, wie er den Buildvorgang koordinieren und ausführen soll. In der ersten Zeile steht die Anweisung, das Docker-Image vom Docker Hub zu verwenden. Unter `script` stehen die Befehle, die im Docker-Container ausgeführt werden sollen, wenn dieser bei jedem Durchlauf gestartet wird. In diesem Beispiel soll sowohl ein PDF als auch ein LibreOffice-Dokument erstellt werden. Sie können in diesen Zeilen sämtliche Parameter angeben, die zuvor im Buch zur Sprache gekommen sind.

Unter `artifacts` geben Sie an, welche Endprodukte des Buildprozesses Sie aufbewahren möchten. In der Datei `.gitlab-ci.yml` ließen sich noch zahlreiche weitere Angaben machen, die Sie in der offiziellen [Dokumentation von GitLab](#) nachlesen können. Diese minimale Konfiguration reicht jedoch vollkommen aus, um den beispielhaften Workflow zu implementieren.

Eigene GitLab Runner verwenden

Auf [gitlab.com](#) stehen mehrere so genannte Shared Runner zur Verfügung, wie Sie unter [Settings | CI/CD | Runners](#) sehen können. Deren Vorteil ist, dass Sie nichts weiter tun müssen, um gleich loslegen zu können. Nachteilig ist, dass die Shared Runner manchmal sehr lange brauchen, um zu starten und auch das Image immer neu herunterladen. Diese Zeiten lassen sich verkürzen, wenn Sie einen eigenen Runner einrichten. Näheres dazu verrät die [GitLab-Dokumentation](#):

https://docs.gitlab.com/ee/ci/quick_start/README.html#configuring-a-runner

Jobs und Pipelines

Mit `build` in der `.gitlab-ci.yml` haben Sie einen Job definiert, den ein GitLab Runner bei jeder Dateiänderung ausführen soll. Die Bearbeitung von Jobs erfolgt in sogenannten Pipelines:

<https://docs.gitlab.com/ee/ci/pipelines.html>

Unter dem Menüpunkt *CI/CD*, wobei die Abkürzung für *Continuous Integration/Continuous Deployment* steht, finden Sie diese beiden Begriffe wieder und können unter *CI/CD | Jobs* genauer nachvollziehen, was stattfindet, wenn Sie eine Textänderung bzw. einen Commit machen.

Status	Job	Pipeline	Stage	Name	Coverage
passed	#108520828 master a948dde9	#33097984 by	test	build	01:42 23 minutes ago

Abbildung 9.6: Erfolgreich erledigter Job

Das farbige Label am Anfang der Zeile zeigt den Erfolg oder Misserfolg des GitLab Runners an, der den Job bearbeitet hat. Ein Klick auf dieses Label gibt genaueren Einblick in den Bearbeitungsvorgang (siehe die folgende Abbildung).

```

passed Job #108520828 triggered 27 minutes ago by Axel Dürkop

Running with gitlab-runner 11.4.0-rc1 (1ff344e1)
  on docker-auto-scale fa6cab46
Using Docker executor with image xldrkp/pandoc_23_auto ...
Pulling docker image xldrkp/pandoc_23_auto ...
Using docker image
sha256:75db50d83730034609c8340d8b291f8d335c133e3d838d6a079732993a8904f6 for
xldrkp/pandoc_23_auto ...
Running on runner-fa6cab46-project-8886319-concurrent-0 via runner-fa6cab46-
srm-1539679929-91159a51...
Cloning repository...
Cloning into '/builds/xldrkp/kollaboratives-schreiben'...
Checking out a948dde9 as master...
Skipping Git submodules setup
$ pandoc *.md -o bericht.pdf
$ pandoc *.md -o bericht.odt
Uploading artifacts...
*.pdf: found 1 matching files
*.odt: found 1 matching files
Uploading artifacts to coordinator... ok      id=108520828
responseStatus=201 Created token=xqiuVXc2
Job succeeded
    
```

Abbildung 9.7: Analyse des Jobs

Zunächst holt sich der GitLab Runner das Image vom Docker Hub (1) und kloniert anschließend das Repository mit den Markdown-Dateien (2). In einem Container, den der Runner auf Basis des Image erstellt und startet, führt er die Befehlszeilen aus der `.gitlab-ci.yml` aus (3). Die entstandenen Artefakte lädt er abschließend wieder in das Repository hoch (4).

Artefakte finden und herunterladen

Die erzeugten Artefakte – eine PDF- sowie eine ODT-Datei – können nun direkt über den Downloadbutton auf der Job-Seite heruntergeladen werden. Aber auch auf der Homepage des Projekts finden Sie einen direkten Zugang zu den letzten erzeugten Artefakten.

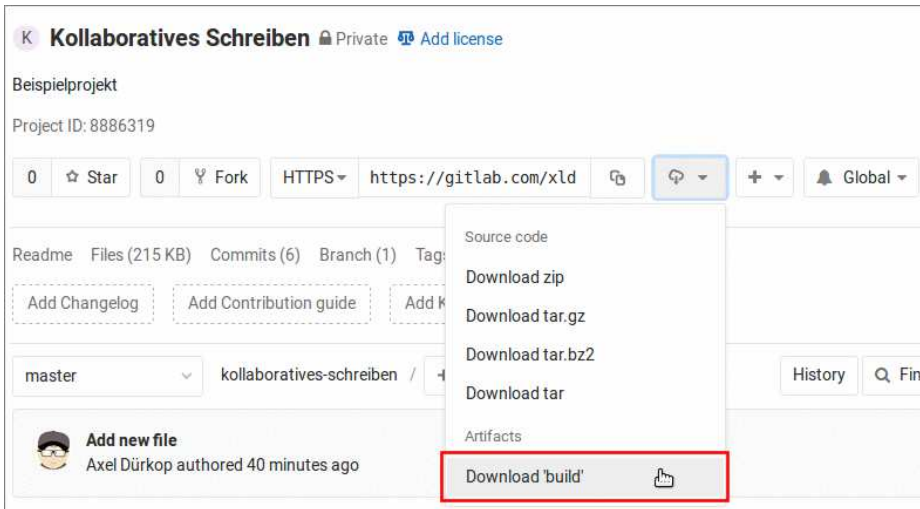


Abbildung 9.8: Herunterladen der Artefakte auf der Homepage des Projekts

README.md von der Konvertierung ausnehmen

Sie werden feststellen, dass auch die README.md von der allgemeinen Angabe des Dateityps erfasst wird. Wenn Sie die Datei davon ausnehmen wollen, benennen Sie sie um in README.markdown.

Kollaborativ schreiben

Die Einrichtung des Workflows zum kollaborativen Schreiben ist damit abgeschlossen. Wann immer Sie jetzt eine Datei hinzufügen oder ändern, arbeitet der Runner in einer Pipeline den konfigurierten Job ab und stellt die Artefakte in GitLab bereit. Wenn Sie mit mehreren Personen an Ihren Dokumenten zusammenarbeiten wollen, fügen Sie andere GitLab-Nutzerinnen und -Nutzer über *Settings / Members* Ihrem Projekt hinzu.

Im Moment arbeiten alle Beteiligten im master-Branch, dem ersten und zunächst einzigen Entwicklungszweig des Projekts. Für kollaborative Szenarien macht es jedoch Sinn, in verschiedenen Branches zu arbeiten. Wenn Bob bspw. für die Einführung zuständig ist, könnte er über *Repository / Branches* einen neuen Branch einfuehrung auf Grundlage des masters erstellen. Der Branch einfuehrung ist in dem Moment eine exakte Kopie des master-Zweig. Darin kann Bob nun schreiben, ohne dass seine Arbeit den master-Zweig beeinträchtigt. Und auch für seinen neuen Branch erstellt GitLab eigene Artefakte, die unter *CI/CD / Pipeline* heruntergeladen werden können!

Wenn Bob seine Arbeit für soweit gediehen hält, dass sie diskutiert und in das finale Dokument integriert werden können, stellt er einen *Merge Request* von seinem Branch einfuehrung auf den master. Alice, die für das Projekt verantwortlich ist, sowie alle anderen Autorinnen und Autoren können in der Detailansicht eines *Merge Requests* nun über den Beitrag von Bob diskutieren.

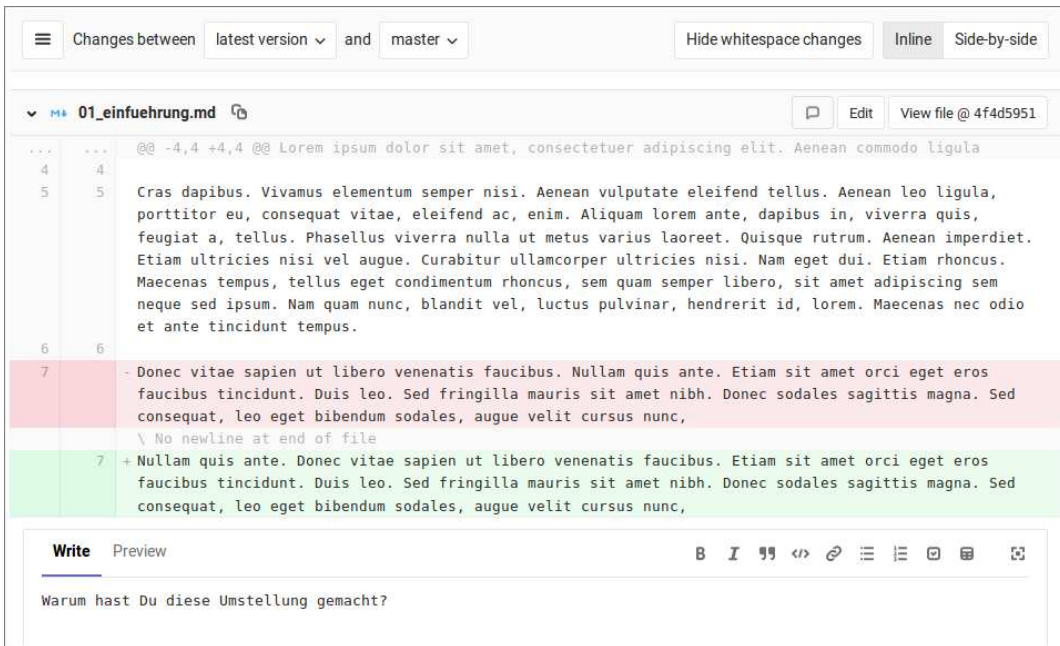


Abbildung 9.9: Ansicht der Änderungen in einem Merge Request. Jede Zeile in einem Merge Request kann kommentiert und diskutiert werden.

Entspricht der Beitrag von Bob den Qualitätsrichtlinien, auf die sich das Team verabredet hat, kann sein *Merge Request* angenommen werden. Damit werden Bobs Änderungen und Ergänzungen mit den Dateien des *masters* vereint.

Abschließende Betrachtung

Die skizzierte Art der Kollaboration wird heute in den meisten Software-Entwicklungsprojekten praktiziert, wobei dort nicht Texte gemeinsam bearbeitet werden, sondern Quellcode von Skript- und Programmiersprachen. Wir haben in verschiedenen Projekten und in der Arbeit mit Studierenden die Erfahrung gemacht, dass der Einstieg über das kollaborative Schreiben hilft, diese Lernkurve zu senken. Denn nicht jeder kann programmieren. Auch für die gemeinsame Entwicklung von Webseiten eignet sich der Workflow ganz ausgezeichnet. So können mit Pandoc generierte HTML-Dokumente bspw. mit `scp` in ein Verzeichnis eines Webservers hochgeladen werden. Auch dies können Sie in der `.gitlab-ci.yml` konfigurieren.

Durch das kollaborative Schreiben mit Markdown, Pandoc, Docker und GitLab können Sie sich mit zeitgemäßen Praktiken der Entwicklung von Software und digitalen Medien vertraut machen.