Markdown und 'ne Prise *Pandoc* *(Vorab-Version)*

Kurt Pfeifle <kurt.pfeifle@mykolab.com>

FrOSCon 2015,
St. Augustin,
22. August 2015

Inhaltsverzeichnis

Dieses Dokument gibt eine Einführung in das Textformat *Markdown*. Außerdem deutet es an, wie man mittels des Dokumenten-Konverters `pandoc` dieses einfache Textformat nach ausdrucksstärkeren Formaten wie HTML, LaTeX, PDF, ODT, EPUB und andere wandelt.

Der Beitrag stellt hierzu diverse praktische Beispiele vor.

Selbstverständlich ist sein Quelltext in Markdown geschrieben! Und ebenso selbstverständlich hat `pandoc` das dem Leser jetzt vorliegende Dokumenten-Format aus genau diesem Markdown erzeugt – was immer dieses Format auch sei: PDF, HTML, EPUB, DOCX oder ODT.

Die Fortsetzung dieses Dokuments, *"Pandoc Power-Features"* (siehe Link unter "Files" auf der FrOSCon15-Webseite zum Workshop)[1], behandelt einige fortgeschrittenere Themen, die für die Erstellung längerer wissenschaftlicher Texte wichtig sind: z.B. Tabellen, Fußnoten, Mathe-Formeln und Bibliographien.

---

[1] http://programm.froscon.de/2015/events/1620.html

Kapitel 1

Das Kleine Einmaleins von Markdown (mit etwas *pandoc*)

Markdown und `pandoc` sind zwei verschiedene Sachen:

- Markdown ist ein *Format* zum Schreiben von Texten.
- `pandoc` ist ein Kommandozeilen-Tool zum Konvertieren von Markdown-Texten in andere 'reiche' Formate.

Markdown beruht auf reinem (ASCII- oder UTF-8-)Text. Man kann dem Text damit ziemlich leicht bestimmte Auszeichnungen einzelner Elemente mitgeben. Beim Konvertieren des Markdown-Textes in andere Formate erscheinen die markierten Stellen dann als die beabsichtigen Formatierungen. Aufgrund der Einfachheit der Auszeichnungs-Elemente und ihrer fast schon intuitiv erscheinenden Arbeitsweise bleibt das Ursprungs-Dokument auch in der Quelltext-Form weiterhin gut lesbar.

Man stelle sich vor: man soll eine E-Mail schreiben, ohne HTML zu verwenden zu dürfen. *Wie würde man den Email-Text gestalten, um damit gewisse Formatierungs-Absichten auszudrücken? Um Betonungen, Hervorhebungen, Überschriften, Listen usw. darzustellen?*

Eben.

Man hat das doch schon öfters gesehen... oder nicht?!?

Github[1] übrigens unterstützt Markdown ebenfalls. Viele der dort hinterlegten Software-Dokumentationen sind als `README.md` vorhanden. Auf **StackOverflow**[2] basiert die Eingabe von Frage- oder Antwort-Beiträgen gleichermaßen auf Markdown.

Wer einmal etwas vertraut mit dem Schreiben von Markdown und dem Umgang mit `pandoc` geworden ist, kommt schnell in die Versuchung, seine gesamte Produktion von größeren Dokumenten auf diese beiden Komponenten zu stützen. Denn damit kann man in Windes-Eile sehr viele verschiedene Ausgabe-Formate erzeugen, alle aus ein und demselben Quelltext.

Markdown-Quelltext und `pandoc` versetzen selbst Anwender, die keinerlei Ahnung von LaTeX haben, in die Lage, mit dem Erstellen dieses anspruchsvollen Dokumenten-Formats erfolgreich zu beginnen. Selbst fortgeschrittene LaTeX-Gurus können enorm profitieren: denn Markdown-Schreiben geht um einiges schneller von der Hand als direkt LaTeX zu coden – und die aller-anspruchsvollsten Feinheiten, die `pandoc` und Markdown (noch) nicht abdecken, kann ein Profi im Anschluss an die Markdown => LaTeX-Konvertierung immer noch händisch dem `pandoc`-generierten LaTeX-Code beifügen.

[1] http://github.org/
[2] http://stackoverflow.com/

Kapitel 2

Etwas Markdown-Geschichte

Angefangen hat es 2004 mit einer Initiative von John Gruber[1], der dabei die Unterstützung von Aaron Swartz[2] hatte. Diese wollten eine möglichst einfache Methode schaffen, um aus Text mit simplen Hervorhebungen vollwertiges HTML zu erzeugen. Die beiden legten eine Reihe von Regeln fest, wie diese Hervorhebungen im Ursprungs-Text beschaffen sein sollten. Zugleich erschuf John ein (ebenfalls `markdown` genanntes) Tool, um solchen Text nach HTML zu konvertieren.

Andere Initiative und Projekte erweiterten John's und Aaron's anfängliche Regeln mit der Zeit, um noch mehr Formatierungs-Optionen zu erhalten. Auf diese Weise entstanden bis heute mehrere verschiedene Markdown-Dialekte (die leider z.T. nicht immer miteinander harmonieren). Ebenso kamen neue Konvertierungs-Pfade hinzu: nach HTML, nach LaTeX oder nach ODT.

Dabei entstanden im Lauf der Jahre mehrere neue Markdown-Konverter. Wer einen Vergleich unterschiedlicher Markdown-nach-HTML-Konverter sucht, ist hier an der richtigen Stelle:

- **johnmacfarlane.net/babelmark2/**

Der mächtigste all dieser Konverter ist heutzutage zweifellos **`pandoc`**[3], geschrieben von John MacFarlane[4], Professor für Philosophie und Logik an der Berkeley University of California.

[1] http://daringfireball.net/projects/markdown/
[2] http://www.aaronsw.com/weblog/001189
[3] http://johnmacfarlane.net/pandoc/demos.html
[4] http://johnmacfarlane.net/

Bereits bei seiner ersten Release, 2006, konnte Pandoc die folgenden Format-Transformationen durchführen:

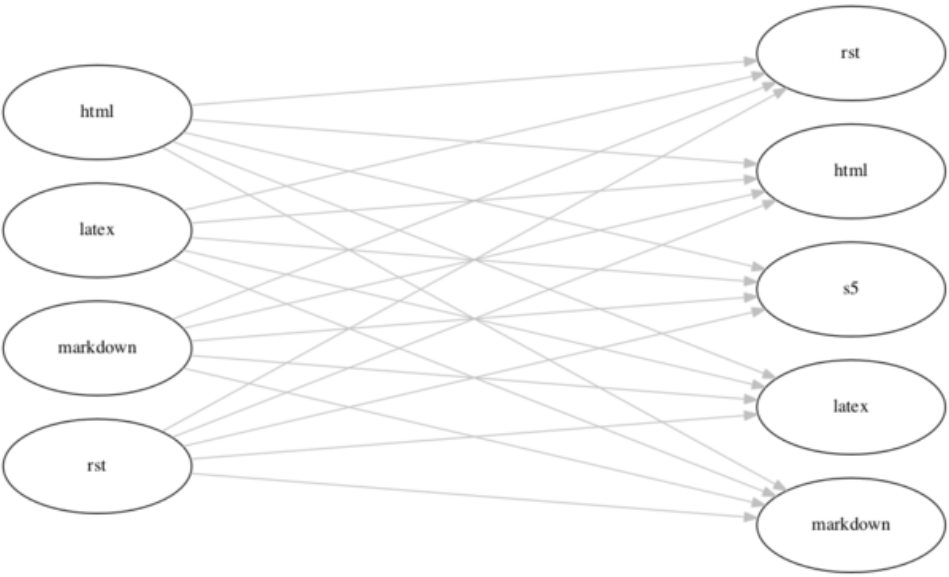

Abbildung 2.1: Pandoc 2006 – 4 Eingabe- und 5 Ausgabe-Formate

Kapitel 3

Pandoc

`pandoc` kann inzwischen viel mehr als nach der ersten Veröffentlichung vor neun Jahren.

3.1 Fortschritte

Es nicht nur in der Lage, das ursprüngliche, einfache Markdown verarbeiten:

1. Erstens kommt `pandoc` mit den meisten modernen Markdown-Dialekten ebenfalls sehr gut zurecht.
2. Als ***Eingabe*** können ebenso HTML, LaTeX, DocBook, RST (reStructuredText) sowie praktisch alle aktuellen Markdown-Varianten dienen.
3. Auch bei den ***Ausgabe***-Formaten erweist es sich als Tausendsassa: nicht nur HTML, DocBook, LaTeX, RST und alle möglichen Markdown-Dialekte sowie PDF sind möglich, sondern auch ConTeXt, MediaWiki, ODT (Open/LibreOffice Text), EPUB, EPUB3, JSON, XML, ManPage, DOCX (MS Word), FictionBook2, AsciiDoc und Rich Text Format.
4. Schließlich kann man beim HTML-Output auf einfache Weise auch so anspruchsvolle Dinge wie *RevealJS*[1]- und *ImpressJS*[2]-Präsentationen erzeugen, oder für's PDF-Zielformat *Beamer*[3]-PDF-Folien.

Die Entwicklung von `pandoc` schreitet nach wie vor zügig voran. Während der letzten beiden Jahre kamen praktisch monatlich neue Features hinzu. Seit der

[1] http://lab.hakim.se/reveal-js/
[2] http://bartaz.github.io/impress.js/
[3] http://de.wikipedia.org/wiki/Beamer_(LaTeX)

Version 1.0 im September 2008 gab es ca. 60 verschiedene Releases, also ca. alle 7 Wochen eine.

Der neueste Stand ist Version 1.15.0.6 (21. August 2015).

Und die aktuellen Konvertierungs-Pfade sehen wie folgt aus:

Viele dieser Konvertierungen lassen sich auch Online im Browser testen:

- **pandoc.org/try/**

3.2 Standardisierungs-Bemühungen: CommonMark

Um den Wildwuchs der diversen Markdown-Dialekte zu zähmen, hat seit ca. 1 Jahr eine Initiative namens *CommonMark* die Arbeit auf sich genommen, eine genaue und wasserdichte Spezifikation für einen gemeinsamen Standard zu definieren.

Die aktuelle Spezifikation für CommonMark ist hier zu finden:

- **spec.commonmark.org/current/**

Zugleich implementiert die Initiative Referenz-Konverter für Markdown-nach-HTML in mehreren Programmiersprachen (darunter JavaScript und C).

Ein "Dingus" genanntes Online-Tool ermöglicht, mit der JavaScript-Implementierung interaktiv im Browser-Fenster mit CommonMark zu spielen:

- **spec.commonmark.org/dingus**

3.3 Warnung: Keine Illusionen!

Wer jetzt glaubt, alle diese Formate würden verlustfrei und vom Aussehen her 1:1 ineinander konvertiert, ist allerdings auf dem Holzweg.

Pandoc übernimmt beim Konvertieren keinerlei *Aussehen* des Ursprungs-Dokuments. Pandoc überträgt vielmehr die Dokumenten-*Struktur* (sowie natürlich die -Inhalte) auf das Ziel-Format. Ähnliche oder gar identische Wiedergabe des Aussehens ist kein Ziel von Pandoc-Dokumentenkonvertierungen!

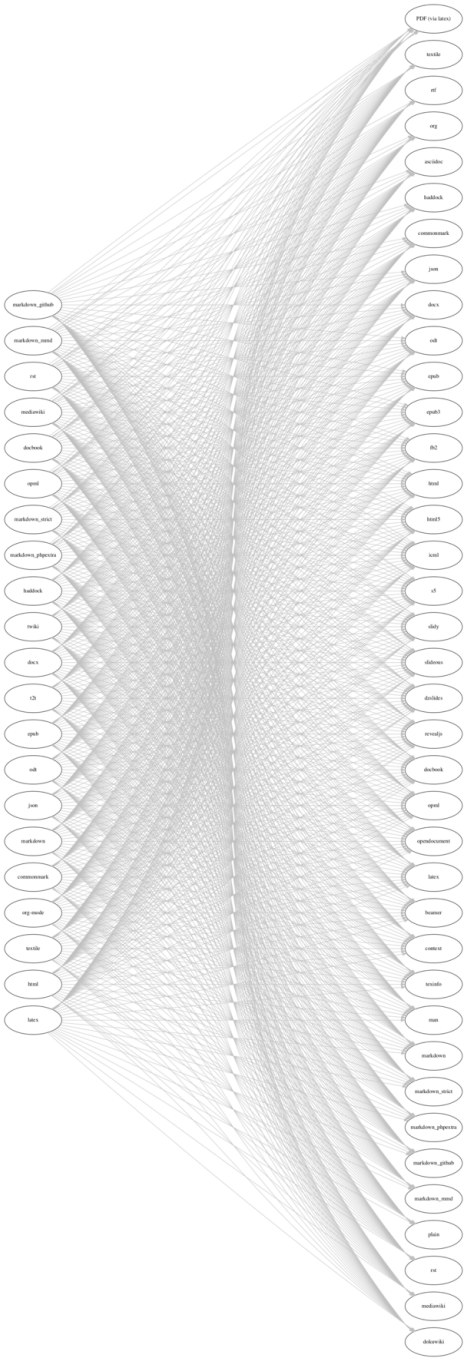

Abbildung 3.1: Pandoc 2015 – 21 Eingabe- und 38 Ausgabe-Formate

3.4 Interne Pandoc-Strukturen

Pandoc's interne Programm-Struktur ist sehr modular konzipiert.

- Für jedes Eingabe-Format existiert ein spezialisiertes *Reader*-Modul.
- Für jedes Ausgabe-Format existiert ein entsprechendes *Writer*-Modul.

Bei jeder Konvertierung stellt der zuständige Pandoc-Reader erstmal ein Pandoc-internes Dokumenten-Format her, genannt *"native"*. Dieses Native-Format übernimmt dann der zuständige Writer, um daraus das Ziel-Dokument zu erstellen. Wer erforschen möchte, wie das Native-Format aussieht, kann dies mit beliebigem Input tun, denn es existiert selbstverständlich auch ein Native-Writer:

```
pandoc input --to=native
```

Die eigentliche Stärke von Pandoc liegt für die meisten überzeugten Anwender vermutlich in einem Workflow, der es erlaubt, praktisch alle Inhalte sehr effizient als Markdown-Text zu schreiben, und anschliessend per Befehl `make mydocument.alldocs` alle möglichen (oder auch nur ein einziges) Zielformat "in einem Rutsch" zu erzeugen.

3.5 Beispiel LaTeX-Workflow

Wer kein LaTeX kann, kann trotzdem sehr gut aussehende Dokumente produzieren – weil Pandoc den LaTeX-Code schreibt. Wer LaTeX-Anfänger ist, kann sehr viel über LaTeX lernen, wenn er sich mit dem interaktiven Modus anfreundet. Wer LaTeX-Profi ist, kann seinen Fließ-Text und seine ersten paar Entwürfe viel effizienter in Markdown schreiben, und Pandoc eine erste LaTeX-Fassung erstellen lassen.

Hier ein Schnell-Vergleich:

- Das vorliegende Dokument hat in seiner Markdown-Fassung (Quelltext) eine Größe von ca. 40.000 Zeichen.
- Nach Konvertierung in LaTeX besteht das Dokument aus ca. 53.000 Buchstaben.

Das wären ca. 32% mehr an reiner Tipp-Arbeit, falls man das direkt als LaTeX eintippen wollte – von den gelegentlichen oder auch häufigeren Problemen ganz zu

schweigen, etwa die Syntax eingebauter LaTeX-Tabellen korrekt einzugeben. (Und man wäre auch der Optionen beraubt, dasselbe Dokument ohne viel Extra-Aufwand zusätzlich als HTML, EPUB, DOCX oder ODT zu veröffentlichen.) Wo der Profi mit manchen Resultaten der Pandoc-Ausgabe nicht zufrieden ist, kann er für sein eigenes Fein-Tuning dann an der LaTeX-Fassung selbst wieder Hand anlegen.

Auch beim Korrekturlesen des Inhalts müssen Auge und Hirn sich mit 32% mehr überflüssigen Buchstaben herumschlagen – zumal solchen, die den Lesefluss eher behindern als befördern...

Das Text-Quellformat (Markdown) ist ideal zum Speichern und zur Versions-Kontrolle in einem Versions-Kontrollsystem, wie z.B. Git. Als kostenlose "Beigabe" liefert Git für Text-basiertes Schreiben das Feature mit, ein "kollaboratives Arbeiten" an Dokumenten zu ermöglichen.

Pandoc macht sich insbesondere für solche Schreiber und Autoren sehr schnell unverzichtbar, die z.B. als Freiberufler technische Dokumentation für ihre Projekte schreiben müssen, wobei von Kunde zu Kunde andere Anforderungen hinsichtlich des Zielformats vorgegeben und einzuhalten sind.

Auf diese Weise kann man sogar Word-Dokumente erstellen, ohne Word installiert zu haben – sogar auf Macs, auf Linux, auf *BSDs oder auf Raspberry Pis.

Kapitel 4

Markdown: die Grundlagen

Also, nochmal zur Frage: *Wie würde man Text gestalten, um folgende Formatierungs-Elemente darzustellen?*

- Listen wie diese hier
- *kursive* Worte
- **fette** Worte
- ***fett-kursive*** Worte
- Satz-Teile, welche `code`-Elemente enthalten
- Absätze (werden durch Leerzeilen voneinander getrennt)

Die Antwort ist ganz einfach. Nämlich so:

```
* Listen wie diese hier
* *kursive* Worte
* **fette** Worte
* ***fett-kursive*** Worte
* Satz-Teile, welche `code`-Elemente enthalten
* Absätze werden durch Leerzeilen voneinander getrennt
```

Es gibt für manche der angestrebten Formatierungen mehrere Möglichkeiten, das entsprechende Markdown zu schreiben. Man kann z.B. als Listen-Markierer statt dem Sternchen (‘`*`’) auch Plus- oder Minus-Zeichen (‘`+`’ oder ‘`-`’) verwenden. Zudem lassen sich einige Syntax-*Erweiterungen* nutzen, die `pandoc` unterstützt, in der Ur-Form von Markdown jedoch nicht vorhanden sind. Beispielsweise könnte man direkten HTML- oder LaTeX-Code in den Quelltext einfügen, sofern man mit diesen Sprachen vertraut ist und die `pandoc`-Erweiterungen `+raw_html` und `+raw_tex` nutzt..

Wofür ist Markdown also gut? Na, für mehreres:

- ***Erstens*** hat man ein Dokument in Text-Form, welches ziemlich übersichtlich aussieht und sich sehr gut lesen lässt.
- ***Zweitens*** lässt sich dieses Dokument (da in Text-Form) ausgezeichnet in Versions-Kontroll-Systemen verwenden.
- ***Drittens*** kann man aus Markdown-Dokumenten mittels verschiedener Tools auf sehr einfache Weise andere Formate erzeugen:

  i. HTML
  ii. PDF
  iii. RTF
  iv. LaTeX
  v. LaTeX Beamer Präsentationen
  vi. ODT (OpenOffice/LibreOffice)
  vii. DOCX (MS Word)
  viii. Text
  ix. AsciiDoc
  x. reStructuredText
  xi. DocBook XML
  xii. groff man pages
  xiii. GNU texinfo
  xiv. Text
  xv. MediaWiki markup
  xvi. ConTeXt
  xvii. EPUB, nämlich:
    1. EPUB v2
    2. EPUB v3

Es fehlt noch die Auflistung von Elementen, die man benötigt, um die Dokumentenstruktur ein bißchen zu gliedern. Damit wären die Grundlagen dann bereits fertig:

- Überschriften 1. Ordnung
- Überschriften 2. Ordnung
- [. . . .]
- Überschriften 6. Ordnung

Zu Beginn dieses Dokuments befindet sich eine *Überschrift 1. Ordnung.* Folgende Markdown-Textzeile hat zu dieser Überschrift geführt:

```
# Das Kleine Einmaleins von Markdown (mit etwas `pandoc`)
```

4.1 Überschrift 2. Ordnung

Die vorhergehende *Überschrift 2. Ordnung* kam so zustande:

```
## Überschrift 2. Ordnung
```

4.1.1 Überschrift 3. Ordnung

Die obige *Überschrift 3. Ordnung* hatte diesen Quell-Text:

```
### Überschrift 3. Ordnung
```

Wie Überschriften 4., 5. und 6. Ordnung gehen, sollte jetzt leicht zu erraten sein.

> **Das war jetzt eigentlich schon das Wichtigste.** ***Mehr braucht man nicht zu wissen, um mit Markdown-Schreiben anzufangen!*** Wie man das mittels `pandoc` konvertiert, braucht man an dieser Stelle noch gar nicht zu wissen. Das kann man sich aneignen, sobald man seinen ersten Markdown-Text geschrieben hat.

Kapitel 5

Vertiefende Details

Die Liste im vorhergehenden Abschnitt war übrigens verschachtelt, und sie benutzte in der 2. und 3 Ebene jeweils ein anderes Aufzählungszeichen. Das ging so:

```
* ***Erstens*** hat man ein Dokument in Text-Form, welches ...
* ***Zweitens*** lässt dieses Dokument (da in Text-Form) au...
* ***Drittens*** kann man aus Markdown-Dokumenten mittels v...
    i. HTML
    i. PDF
    i. RTF
    i. LaTeX
    i. LaTeX Beamer Präsentationen
    i. ODT (OpenOffice/LibreOffice)
    i. DOCX (MS Word)
    i. Text
    i. AsciiDoc
    i. reStructuredText
    i. DocBook XML
    i. groff man pages
    i. GNU texinfo
    i. Text
    i. MediaWiki markup
    i. ConTeXt
    i. EPUB, nämlich:
        1. EPUB v2
        1. EPUB v3
```

Beim Verschachteln von Listen sollte man einfach die nächste Ebene um 4 zusätzliche Leerzeichen einrücken. Die Auswahl der Aufzählungs-Zeichen bietet selbstverständlich einige Optionen mehr als die oben gezeigten: `a)`, `A.`, `(i)`, um nur drei zu nennen.

Wie man unschwer erkennen kann, braucht man nummerierte Listen nicht händisch zu pflegen, um die richtige Reihenfolge der Nummerierungs-Zeichen zu erhalten – das macht `pandoc` bereits ganz automatisch.

5.1 Weitere Elemente

Bisher fehlen noch ein paar weitere Elemente, um einen Text aufzupeppen:

1. Hyperlinks[1]
2. Bilder und Grafiken
3. Einrückungen/Zitate
4. Definitions-Listen
5. Nummerierte Listen (wie diese hier)
6. Code-Blöcke
7. Tabellen
8. Horizontale Linien

5.1.1 Hyperlinks

Die schnellste Möglichkeit, einen Hyperlink einzufügen, besteht darin, folgendes Markdown zu schreiben: `[Hyperlinks](http://en.wikipedia.org/wiki/Hyperlink)`. So habe ich das in der vorhergehenden Liste auch gemacht. Das darf sogar mitten im Satz vorkommen:

- In eckigen Klammern, `[....]`, steht der im Zieldokument anzuzeigende Link-Text.
- Direkt dahinter, ohne Leerraum, folgt in runden Klammern, `(....)`, das Sprungziel des Hyperlinks.

5.1.2 Horizontale Linien

Horizontale Linien wie diejenige vor der obigen Überschrift entstehen, indem man mindestens 4 “Dash”-Zeichen hintereinander eingibt, auf einer eigenen Linie:

[1] http://en.wikipedia.org/wiki/Hyperlink

```
---
```

5.1.3 Einrückungen/Zitate

Ein Zitat bzw. eine Einrückung ist dadurch markiert, dass sich vor dem Absatz ein `>` (plus 1 Leerzeichen) befindet:

> **Dies ist ein Zitat. Es besteht aus 2 Absätzen.** Lorem ipsum dolor sit amet, consetetur sadipscing elitr.
>
> Stet clita kasd *gubergren*, no sea takimata sanctus est Lore ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy.

Es ist aus folgendem Markdown entstanden:

```
> **Dies ist ein Zitat. Es besteht aus 2 Absätzen.** Lorem ipsum dolor
> sit amet, consetetur sadipscing elitr.
>
> Stet clita kasd *gubergren*, no sea takimata sanctus est Lore ipsum
> dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing
> elitr, sed diam nonumy.
```

5.1.4 Bilder und Grafiken

Die schnellste Möglichkeit, ein Bild einzufügen, besteht darin, folgendes Markdown zu schreiben: `![Bild-Unterschrift: Ghostscript-Logo](../resources/logo.png)`. Das darf allerdings ***NICHT*** mitten im Satz vorkommen, sondern es muss auf einer separaten Zeile stehen, mit jeweils einer Leerzeile davor und dahinter! Statt PNG kann man natürlich auch JPEG verwenden.

Eine dergestalt eingebundene Abbildung rendert `pandoc` als eigenen Absatz und verleiht ihr eine automatisch nummerierte Bild-Unterzeile[2].

WICHTIG: Der Markdown-Code für ein Bild fängt immer mit einem Ausrufezeichen an. Also so:

```
![Bild-Unterschrift: Ghostscript-Logo](../resources/logo.png)
```

Das erzeugt dann das (eventuell erst auf der folgenden Seite) dargestellte Bild.

[2] Leider funktioniert dies bei den Ausgabeformaten ODT, RTF und OpenDocument (noch) nicht – hier wird der Bild-Untertitel ausgelassen.

Abbildung 5.1: Bild-Unterschrift: Ghostscript-Logo

Dabei erscheint das Bild unskaliert in seinen Original-Abmessungen bei einer Auflösung von 72 PPI, horizontal mittig auf der Seite. Sollte das Bild zu groß sein, wird es entsprechend verkleinert dargestellt, damit es auf die Seite passt. Bei diesem Runterskalieren werden die Bild-Daten nicht verändert – entsprechend steigt die PPI-Zahl im Bildbereich der Seite an.

Leider ist es derzeit (noch) nicht möglich, den Bildbereich der Seiten vom Markdown-Code aus zu skalieren, oder Bilder rechts oder links am Seitenrand zu platzieren, um sie von Text umfließen zu lassen.

5.1.5 Definitions-Listen

Definitionslisten kann man mittels folgender Markdown-Syntax erzeugen. Dabei sind innerhalb der Listen-Elemente weitere Markdown-Auszeichnungen zulässig:

```
Terminus 1.

:   Hier kommt die *Definition 1*, welche den *Terminus 1* erklärt.
    Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam.

Shellfunktion mit `Ghostscript`.

:   *Definition 2.* Lorem ipsum dolor sit amet, consetetur sadipscing
    elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore.

    Nach dem zweiten Absatz fügen wir sogar noch einen Code-Block mit
    Syntax-Highlighting ein.

    ~~~ {.bash}
     # Etwas eingefügter Code, Teil der Definition 2. Es handelt sich
     # um ein Beispiel, welches eine Bash-Funktion darstellt, die es
     # ermöglicht, Ghostscript als Taschenrechner für die Kommando-
     # Zeile zu benutzen:
     function gscalc()
     {
         IFS=" ";
         gs -dNODISPLAY -q -c "$* == quit" | head -n 1
     }
     # Anwendungsbeispiel:
     #    gscalc 13 2 div 5 5 mul add
     # berechnet: (13/2)+(5*5)
    ~~~

    Vierter Absatz der Definition 2. Lorem ipsum dolor sit amet,
    consetetur.

Terminus 3
```

```
:   *Definition 3*. Innerhalb von Definitionslisten kann man
    ***natürlich*** auch `weitere` Auszeichnungen **einbauen**.
```

Da kommt dann folgendes raus:

Terminus 1. Hier kommt die *Definition 1*, welche den *Terminus 1* erklärt. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam.

Shellfunktion mit `Ghostscript`. *Definition 2.* Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore.

Nach dem zweiten Absatz fügen wir sogar noch einen Code-Block mit Syntax-Highlighting ein.

```
# Etwas eingefügter Code, Teil der Definition 2. Es handelt sich
# um ein Beispiel, welches eine Bash-Funktion darstellt, die es
# ermöglicht, Ghostscript als Taschenrechner für die Kommando-
# Zeile zu benutzen:
function gscalc()
{
    IFS=" ";
    gs -dNODISPLAY -q -c "$* == quit" | head -n 1
}
# Anwendungsbeispiel:
#    gscalc 13 2 div 5 5 mul add
# berechnet: (13/2)+(5*5)
```

Vierter Absatz der Definition 2. Lorem ipsum dolor sit amet, consetetur.

Terminus 3 *Definition 3.* Innerhalb von Definitionslisten kann man ***natürlich*** auch `weitere` Auszeichnungen **einbauen**.

5.1.6 Code-Blöcke

Falls man einen Code-Block benötigt, der so aussehen soll (ohne Syntax-Highlighting):

```
%!PS
%
% PostScript-Code

/Helvetica findfont 48 scalefont setfont
   72 600 moveto
```

```
   1 0 0 setrgbcolor
       (Hello, World!) show
showpage
```

rückt man den entsprechenden Abschnitt im Markdown-Quelltext einfach um jeweils (mindestens) 4 Leerzeichen ein:

```
    %!PS
    %
    % PostScript-Code

    /Helvetica findfont 48 scalefont setfont
       72 600 moveto
       1 0 0 setrgbcolor
           (Hello, World!) show
    showpage
```

Alternativ zur Einrückung um 4 Leerzeichen kann man auch sogenannte `fenced code blocks` (*"eingezäunte Code-Blocks"*) verwenden. Diese erfordern keine Einrückung, sondern das "Einzäunen" des Code-Bereichs: darunter versteht man das Umschließen des Code-Blocks durch zwei Zeilen mit je (mindestens) 3 `tilde`-Zeichen oder `backticks`. Vorteil von *fenced code blocks*: hier kann man ein Syntax-Highlighting dazuschalten. Dieses Dazuschalten bewirkt man, indem man am Ende der ersten 'Zaun'-Zeile in geschweiften Klammern eine der unterstützten Sprachen, eingeleitet durch einen Punkt, hinzufügt. Hier das Beispiel für das Anfordern von Syntax-Highlighting bei PostScript-Code:

```
~~~ {.postscript}
 %!PS
 %
 % PostScript-Code

 /Helvetica findfont 48 scalefont setfont
    72 600 moveto
    1 0 0 setrgbcolor
   (Hello, World!) show
 showpage
~~~
```

Das Ergebnis sieht dann z.B. so aus (Details wie Farben, Schrift-Größe und -Fonts sind selbstverständlich abhängig vom jeweiligen CSS-File (bei HTML-Ausgabe),

Referenz-Dokument (bei ODT-Ausgabe) oder LaTeX-Vorlage (bei LaTeX- oder PDF-Ausgabe), welches man verwendet):

```
%!PS
%
% PostScript-Code

/Helvetica findfont 48 scalefont setfont
   72 600 moveto
   1 0 0 setrgbcolor
      (Hello, World!) show
showpage
```

An den meisten Stellen dieses Dokuments kommen solche ‘fenced code blocks’ zum Einsatz, bereits ab dem 1. Kapitel. Dabei war die Syntax-Hervorhebung meist ausgeschaltet, indem der erste ‘Zaun’ wie folgt definiert ist:

```
~~~ {.noweb}
```

Wenn man wissen möchte, welche Sprachen `pandoc` per Highlighting unterstützt, kann man dies mit folgendem Kommando abfragen:

```
$>  pandoc --version
```

Es gibt z.Zt. allerdings keinen Code, der z.B. `{.no-syntax-highlight}` oder ähnlich intuitiv lautet. Die Methode mit `{.noweb}` funktioniert als Übergangs-Lösung jedoch ebenso gut. . . .

Um für die Code-Blöcke trotzdem den farbigen (=dunklen) Hintergrund erhalten, kam beim Konvertieren der Kommandozeilen-Parameter `--highlight-style=espresso` zum Einsatz.

Kapitel 6

Tabellen

Das Erzeugen von Tabellen geht ganz einfach.

6.1 Einfachste Möglichkeit: *Simple Tables*

Aus folgendem Markdown...

```
  Right     Left     Center      Default
-------     ------ ----------    -------
   1245     1245      1245       1245
 012345     012345   012345      012345
    123     123        123         123
      1     1           1             1

Table: Beispiel für eine einfache Tabellen-Syntax.
```

...entsteht die Darstellung dieser Tabelle:

Tabelle 6.1: Beispiel für eine einfache Tabellen-Syntax.

Right	Left	Center	Default
1245	1245	1245	1245
012345	012345	012345	012345
123	123	123	123
1	1	1	1

Anmerkungen:

- Die Tabellen-Unterschrift kommt nur dann zustande, wenn im Markdown das (englische!) Schlüsselwort *"Table:"* (mit Doppelpunkt) vorhanden ist und mit einer Leerzeile Abstand vor oder hinter dem Tabellen-Code am Zeilenbeginn steht. (Abgekürzt kann man diesen Effekt ebenfalls erhalten, indem man nur einen einzelnen Doppelpunkt schreibt. . .)
- Die Spalten-Grenzen gibt's nur dort, wo die Strich-Linie an den entsprechenden Stellen unterbrochen ist.
- Eine zweite gestrichelte Linie ist hier überflüssig.
- Im obigen Fall ist die linke Spalte rechtsbündig (Ausrichtung an Dezimal-Kommas geht momentan noch nicht). Die zweite Spalte ist linksbündig, die nächste rechts davon ist zentriert, und die letzte ist wieder rechtsbündig (weil dies die Vor-Einstellung ist).
- Man darf zum Ausrichten der Spalten keine Tabulatoren benutzen!
- Die Kopf- und die Tabellenzeilen müssen im Quelltext ohne Umbruch in einer einzigen Zeile stehen.
- In der Ausgabe als LaTeX oder PDF erscheint die Tabelle in ihrer Gesamtheit immer zentriert.
- Die jeweiligen Spaltenbreiten lassen sich nicht beeinflussen – sie werden automatisch auf die erforderlichen Maße eingestellt.
- Die Ausrichtung der Spalten steuert die jeweilige Position des Kopf-Textes zur darunter liegenden Strich-Linie. Dabei gelten folgende Regeln:

    1. Falls die Strich-Linie auf der rechten Seite bündig mit dem Kopf-Text ist, jedoch auf der linken Seite übersteht, dann ist die Spalte rechtsbündig.
    2. Falls die Strich-Linie auf der linken Seite bündig mit dem Kopf-Text ist, jedoch auf der rechten Seite übersteht, dann ist die Spalte linksbündig.
    3. Falls die Strich-Linie relativ zur Länge des Kopf-Textes auf beiden Seiten übersteht, dann ist die Spalte zentriert.
    4. Falls die Strich-Linie auf beiden Seiten bündig zum Kopf-Text ist, kommt die standard-mäßig voreingestellte Spalten-Ausrichtung zu Tragen (meistens wird dies Linksbündigkeit sein).

Wie leicht zu sehen ist, brauchen die Texte der Tabellen-Zellen nicht selbst exakt ausgerichtet zu sein. Selbstverständlich liest sich der Quelltext etwas schöner, wenn er ebenso formatiert ist, wie man für das End-Ergebnis beabsichtigt.

Alle Formen der folgenden *Simple Table*-Quelltexte werden zur selben Tabellen-Darstellung wie oben in *Tabelle 1* führen.

```
      Right        Left              Center                Default
------------       -------------- ----------------------   -------
       1245         1245             1245                  1245
      012345       012345           012345                 012345
      1              1                1                     1
```

```
   Right        Left          Center            Default
--------       --------    ----------          -------
   1245         1245        1245                1245
  012345       012345       012345              012345
  1              1            1                   1
```

```
   Right Left       Center   Default
-------- -------- ---------- -------
1245         1245 1245         1245
012345     012345 012345     012345
1                1 1                1
```

```
   Right     Left       Center      Default
--------     ------   ----------    -------
    1245     1245        1245       1245
  012345     012345     012345      012345
       1     1            1         1
```

Right	Left	Center	Default
1245	1245	1245	1245
012345	012345	012345	012345
1	1	1	1

Alle o.a. Markdown-Beispiele führen zum selben Ausgabe-Ergebnis (nur 1x gezeigt, oben). Welche dieser Formen einem innerhalb seines Quelltextes besser gefallen, bleibt jedem selbst überlassen.

`pandoc` unterstützt mehrere Erweiterungen, um Tabellen per Markdown zu definieren: `grid_tables`, `pipe_tables` und `multiline_tables`. Diese werden in dem anderen Dokument, *"`pandoc` Power-Features"* ausführlicher besprochen.

Kapitel 7

Mathe-Formeln à la LaTeX

Man kann auch LaTeX-Formeln mit Markdown einsetzen. *(Aber die folgenden Beispiele habe ich zugegebenermaßen er-google-t – ich könnte sie nicht selbst schreiben, wie es wohl richtige LaTeX-Gurus könnten; LaTeX ist momentan noch ein Buch mit 7 Siegeln für mich…)*

Das funktioniert allerdings nur mit der neuesten `pandoc`-Version. Man schreibt einfach den originalen LaTeX-Code für eine Formel und schließt diese innerhalb von 2 `$`-Zeichen ein:

```
$\forall x \in X, \quad \exists y \leq \epsilon$

$\cos (2\theta) = \cos^2 \theta - \sin^2 \theta$

$\lim_{x \to \infty} \exp(-x) = 0$

$\frac{n!}{k!(n-k)!} = \binom{n}{k}$

$\frac{\frac{1}{x}+\frac{1}{y}}{y-z}$

$\int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}x$
```

Wenn es funktioniert, sieht das Ergebnis für die Formeln so aus:

$\forall x \in X, \quad \exists y \leq \epsilon$

$\cos(2\theta) = \cos^2 \theta - \sin^2 \theta$

$\lim_{x \to \infty} \exp(-x) = 0$

$\frac{n!}{k!(n-k)!} = \binom{n}{k}$

$$\frac{\frac{1}{x}+\frac{1}{y}}{y-z}$$

$$\int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}x$$

Im HTML-, ODT- und DOCX-Output kommen diese Formeln auf Anhieb allerdings nur teilweise korrekt rüber – im LaTeX-Output erscheinen sie tadellos (und ebenfalls im PDF, falls man es via LaTeX erzeugt). Damit *alle* Formeln korrekt funktionieren, muß man an mehreren Stellen zusätzlich Hand anlegen – ein Thema, welches über den Rahmen dieses Beitrags hinausgeht.

Kapitel 8

Diverses

8.1 Superscript, Subscript

Nun zu einer kleine Spezialität, die oft ganz nützlich ist: Superscript und Subscript. Hier zunächst der verwendete Markdown-Code:

```
* H~2~O ist (meist) eine klare Flüssigkeit.  2^10^ ergibt 1024.
```

Mal sehen, ob das beim Konvertieren einwandfrei klappt:

- H_2O ist (meist) eine klare Flüssigkeit. 2^{10} ergibt 1024.

8.2 Durchstreichen von Textstellen

Man kann bestimmte Worte auch ~~dUrchstREIchen~~ durchstreichen. Das geht im Markdown so...

```
auch ~~dUrchstREIchen~~ durchstreichen.
```

...also durch Einfassung der durchzustreichenden Stelle in jeweils doppelte "Tilden". – Alles klar?

8.3 Fußnoten

Als nächstes: Fußnoten. Und zwar zuerst eine Form, die im Markdown-Quelltext als *inline*[1]-Fußnote angelegt ist).

Diese Fußnote kommt vom folgenden Markdown:

```
Als nächstes: Fußnoten. Und zwar zuerst eine Form, die im
Markdown-Quelltext als *inline*^[Fußnoten sind sehr leicht zu schreiben.
Man legt diese an beliebigen Stellen des Textes einfach an, indem man
das Zeichen `^` tippt, und direkt anschließend daran in eckigen Klammern
(`[...]`) den Text, den die Fußnote enthalten soll.]-Fußnote angelegt
ist).
```

[1] Fußnoten sind sehr leicht zu schreiben. Man legt diese an beliebigen Stellen des Textes einfach an, indem man das Zeichen `^` tippt, und direkt anschließend daran in eckigen Klammern (`[...]`) den Text, den die Fußnote enthalten soll.

Kapitel 9

Konvertierung dieses Dokuments

Wie bereits gesagt: zum Konvertieren von Markdown-Text in andere Formate verwendet man am besten `pandoc`. (Pandoc kann nicht nur Markdown einlesen, sondern auch AsciiDoc, Textile, reStructuredText, HTML, Textile, DocBook, MediaWiki markup und LaTeX!).

Daraus erzeugt `pandoc` dann die Formate, die bereits weiter oben aufgelistet sind.

Für manche Ausgabe-Formate benötigt `pandoc` evtl. noch ein paar anderweitig vorbereitete Templates, damit das Ergebnis dann auch so aussieht, wie gewünscht. So lässt sich z.B. das ODT/OpenDocumentText-Format ganz unterschiedlich stylen, je nach dem, was man in dem entsprechenden Template hinterlegt hat.

9.1 Konvertieren nach HTML

HTML-Erzeugung aus Markdown ist sehr einfach:

```
pandoc  --to=html  --from=markdown
        -o output.html froscon2015-markdown-und-eine-prise-pandoc.mmd
```

Allerdings erhält man jetzt ein HTML, welches nach wie vor externe Referenzen beinhaltet (z.B. Bilder, Javascripts, CSS-Stylesheets). Falls man ein *transportables* HTML-File erzeugen möchte, welches alle Referenzen eingebettet in sich trägt, muss man die zusätzlich die Parameter `--standalone --self-contained` verwenden.

Des weiteren bietet sich noch an, per `--css=my-stylesheet.css` ein eigenes CSS einzubinden, welches das Erscheinungsbild des HTML-Dokuments nach eigenem Gusto prägt:

```
pandoc                     \
  --to=html                \
  --standalone             \
  --self-contained         \
  --smart                  \
  --highlight-style=espresso \
  --normalize              \
  --filter=pandoc-citeproc   \
  --from=markdown          \
  --toc                    \
  --mathjax                \
  --css=../resources/kp.css  \
  --bibliography=../resources/my.bib                                  \
  --include-after-body=./resources/footer.html                        \
  --output=./build-full/froscon2015-markdown-und-eine-prise-pandoc.html \
    froscon2015-markdown-und-eine-prise-pandoc.mmd
```

Das erzeugte HTML enthält jetzt alle erforderlichen Komponenten und kann (auch bei fehlender Netzverbindung) auf jedem Computer und jedem Betriebssystem betrachtet werden, sofern dort ein Browser vorhanden ist, der modern genug ist.

9.2 Konvertieren nach ODT (Libre/OpenOffice Text-Dokument)

Folgendes Kommando kann zum Erzeugen eines ODTs aus dem Markdown-Quellcode dieses Dokuments dienen:

```
pandoc                                          \
  --to=odt                                      \
  --standalone                                  \
  --smart                                       \
  --reference-odt=../resources/reference---kp.odt \
  --output=../odt/markdown-minidocu.odt         \
  --from=markdown                               \
    clt14-markdown-und-ne-prise-pandoc.mmd
```

Die lange Zeile mit den vielen +-Zeichen (*. . . +mmd_title_block+definition_lists+. . .*) teilt `pandoc` mit, dass das vorliegende Markdown an manchen Stellen entsprechende Syntax-Erweiterungen verwendet. Nur dann erfolgt die Konvertierung in der beabsichtigten Weise.

Der Parameter `--reference-odt=reference---kp.odt` legt fest, dass das resultierende OpenDocument dieselben Schriftarten, Absatz- und Überschriften-Stile (und auch dieselben Seiten-Formate, Kopf- und Fußzeilen) verwenden soll wie das genannte Referenz-Dokument.

9.3 Konvertieren nach PDF

Zur Konversion nach PDF gibt es natürlich auch die Möglichkeit, `pandoc` zuerst ein ODT erzeugen zu lassen (s.o.), dieses in LibreOffice/OpenOffice zu öffnen und dann nach PDF zu exportieren.

Für eine direkte *"Markdown-nach-PDF"*-Konversion benötigt `pandoc` eine lokal installierte LaTeX-Engine: *pdflatex*, *lualatex* oder *xelatex*. Sobald man im Markdown-Quelltext nicht-ASCII-Zeichen verwendet (z.B. `ä`, `ö`, `ü` oder `s`), kommt *pdflatex* allerdings in Schwierigkeiten. Dann müssen *lualatex* oder *xelatex* in die Bresche springen.

Das aller-aller-einfachste Kommando zum Erzeugen von PDF aus dem vorliegenden Markdown (die Quelldatei trägt den Namen *clt14-markdown-und-ne-prise-pandoc.mmd*) sieht so aus:

```
pandoc -f markdown -o my-output.pdf clt14-markdown-und-ne-prise-pandoc.mmd
```

Allerdings gefällt mir das Ergebnis noch nicht besonders, denn es enthält z.B. noch kein Syntax-Highlighting meiner Beispiel-Codeblocks, für deren Erstellung ich mir so große Mühe gegeben habe... Zudem soll das vorbereitete Literatur-Verzeichnis ins fertige Dokument automatisch eingebaut werden. Das verlangt nach ein paar ergänzenden Kommandozeilen-Parametern.

Tatsächlich verwendete ich folgendes Kommando zum Erzeugen des vorliegenden PDFs:

```
pandoc                              \
  ----smart                         \
  --toc                             \
  --highlight-style=espresso        \
  --normalize                       \
  --latex-engine=lualatex           \
  --from=markdown                   \
  -V geometry:paperwidth=4.13193in \
  -V geometry:paperheight=21.0cm    \
  -V geometry:vmargin=14.5mm        \
  -V geometry:tmargin=39pt          \
  -V geometry:bmargin=55pt          \
```

```
  -V fontsize:10pt                \
  --filter=pandoc-citeproc        \
  --csl=mhra.csl                  \
  --biblio=mybiblio.bib           \
  --output=./build-full/froscon2015-markdown-und-ne-prise-pandoc.pdf \
    froscon2015-markdown-und-ne-prise-pandoc.mmd
```

Wie man sieht, kann man die Geometrie-Maße sowohl in Inch (`in`), als auch in Zentimeter (`cm`), Millimeter (`mm`) und Punkten (`pt`) angeben.

Kapitel 10

Weitere, detaillierte Dokumentation

Eigentlich ist Markdown nicht nur eine 'Syntax für Text'. Zu Anfang seiner Geschichte war es zusätzlich ein Tool, welches aus ebendieser Syntax HTML erzeugen konnte. Aber diese vergleichsweise kleine Aufgabe hat inzwischen `pandoc` übernommen – zusätzlich mit einigen weiteren, viel größeren... Heutzutage ist Markdown allerdings in erster Linie als Text-Auszeichnungsformat bekannt.

Der vorliegende Artikel deckt selbstverständlich das Thema nicht erschöpfend ab. Manche `pandoc`-Begriffe hat er nur erwähnt, jedoch nicht erklärt: *Was ist denn z.B. ein 'pandoc_title_block'?!*

Folgende Adressen können als Ausgangspunkt für eigene Recherchen in der Welt der weiteren Markdown- und `pandoc`-ereien dienen:

- Download des ursprünglichen `markdown`-Tools:
  `http://daringfireball.net/projects/markdown/`
- Dokumentation der Markdown-Syntax:
  `http://daringfireball.net/projects/markdown/syntax`
- Download (und Dokumentation) von `pandoc`:
  `http://johnmacfarlane.net/pandoc/`
- Pandoc-verständliche Syntax-Erweiterungen für Markdown:
  `http://johnmacfarlane.net/pandoc/demo/example9/pandocs-markdown.html`

- Vim-Plugin für `pandoc`:
  `https://github.com/vim-pandoc/vim-pandoc`
- EBook über EBook-Publishing mit Markdown & Pandoc, von Jan Ulrich Hasecke:
  "Das ZEN von Pandoc"[1]

[1] http://literatur.hasecke.com/sachbuecher/das-zen-von-pandoc

Kapitel 11

EBook-Publishing bei Leanpub.com

- Ein Internet-Verlag, der eine modifizierte `markdown+pandoc`-Toolchain verwendet und exzellente Konditionen sowohl für Autoren wie auch für Leser/Käufer bietet. Die dort erhältlichen Formate sind PDF, MOBI und EPUB: **Leanpub.com**[1]

- *"PDF-KungFoo with Ghostscript & Co."* (englisch) – mein EBook bei Leanpub als 'work in progress' mit Minimal-Preis $US 0,00: **https://leanpub.com/pdfkungfoo**
  Bitte trotzdem einen selbstgewählten, freiwilligen Betrag bezahlen. Er geht, nach Abzug von 10% plus 50 Cent pro Buchkauf (das ist der Verlags-Anteil) als Spende an die EFF *(Electronic Frontier Foundation)*.

- *"PDF-KungFoo Workshop bei der Ubucon 2013"* (deutsch) – ein kollaborativ erstelltes EBook mit Minimal-Preis $US 0,00: **https://leanpub.com/pdfkungfoo-ws1-deu**
  Bitte trotzdem einen selbstgewählten, freiwilligen Betrag bezahlen. Er geht, nach Abzug von 10% plus 50 Cent pro Buchkauf (das ist der Verlags-Anteil) als Spende an die EFF *(Electronic Frontier Foundation)*.

[1] https://leanpub.com/authors/