Texte auszeichnen mit Markdown

Überschrift

Mit Markdown schnell und einfach Texte auszeichnen

Ob Blog-Einträge, Kommentare in Foren oder Bugtracker auf Entwicklungsplattformen – immer mehr Software unterstützt Markdown zum Auszeichnen von Texten. Nach unserer Einführung wissen Sie nicht nur, warum der Überschrift ein # voransteht, sondern auch, wie Sie Textpassagen kursiv stellen, Aufzählungen und Tabellen eintippen und vieles mehr.

Von Sylvester Tremmel

kompakt

- GitHub, Reddit, Trello – zahllose Systeme verstehen Markdown; unsere Einführung erklärt die weit verbreitete Auszeichnungssprache.
- Markdown zu schreiben geht schnell und fällt leicht; wer die Sprache beherrscht, nutzt sie oft auch für die eigenen Texte und Notizen.
- Auch in kollaborativen Projekten wird dadurch niemand ausgeschlossen, lesbar sind Markdown-Texte ohne Kenntnis der Sprache und in jedem beliebigen Texteditor.

Mit Markdown ergänzt man reinen Text leicht um Formatierungsinformationen, die eine visuell schöne Darstellung erlauben. Der Clou: Markdown-Texte sind nicht nur bequem zu schreiben, sondern auch im Quelltext gut lesbar.

Dadurch benötigt man keine Textverarbeitung wie Microsoft Word oder LibreOffice Writer. Jeder beliebige Texteditor eignet sich, um Markdown zu verfassen und zu lesen. Die schöne Darstellung durch passende Software ist ein Bonus fürs Endergebnis, keine Voraussetzung beim Arbeiten an den Texten.

Weil Markdown so praktisch ist, kennen mittlerweile alle möglichen Systeme die Sprache: zahlreiche Notiz-Apps wie etwa „Drafts“ [1] oder „Obsidian“ [2]; komplexe Systeme zum Schreiben von Texten wie „Notion“ [3] oder „HedgeDoc“ [4] und Nischenanwendungen wie die Rezeptdatenbank „Tandoor Recipes“ [5]. Große Serveranwendungen wie GitHub oder GitLab und viele Systeme für Internetforen erlauben ebenfalls, die eigenen Beiträge und Nachrichten mit Markdown auszuzeichnen. Und in seinem ursprünglichen Zweck – dem Schreiben von Texten zur Publikation im Internet – glänzt Markdown ebenfalls: Zahllose Blogging-Systeme, Website-Editoren [6] und Dokumentations-Generatoren unterstützen die Sprache.

Die weite Verbreitung hat leider den Nebeneffekt, dass diverse Varianten der Sprache existieren (siehe Kasten). Die grundlegenden Features funktionieren aber überall gleich (oder zumindest sehr ähnlich) und was diese Einführung zeigt, sollte in den meisten Markdown-Implementierungen problemlos funktionieren. Der Text orientiert sich grob an CommonMark, weist aber auch darauf hin, wenn es andernorts nette Erweiterungen gibt oder etwas nicht ganz gleich funktioniert.

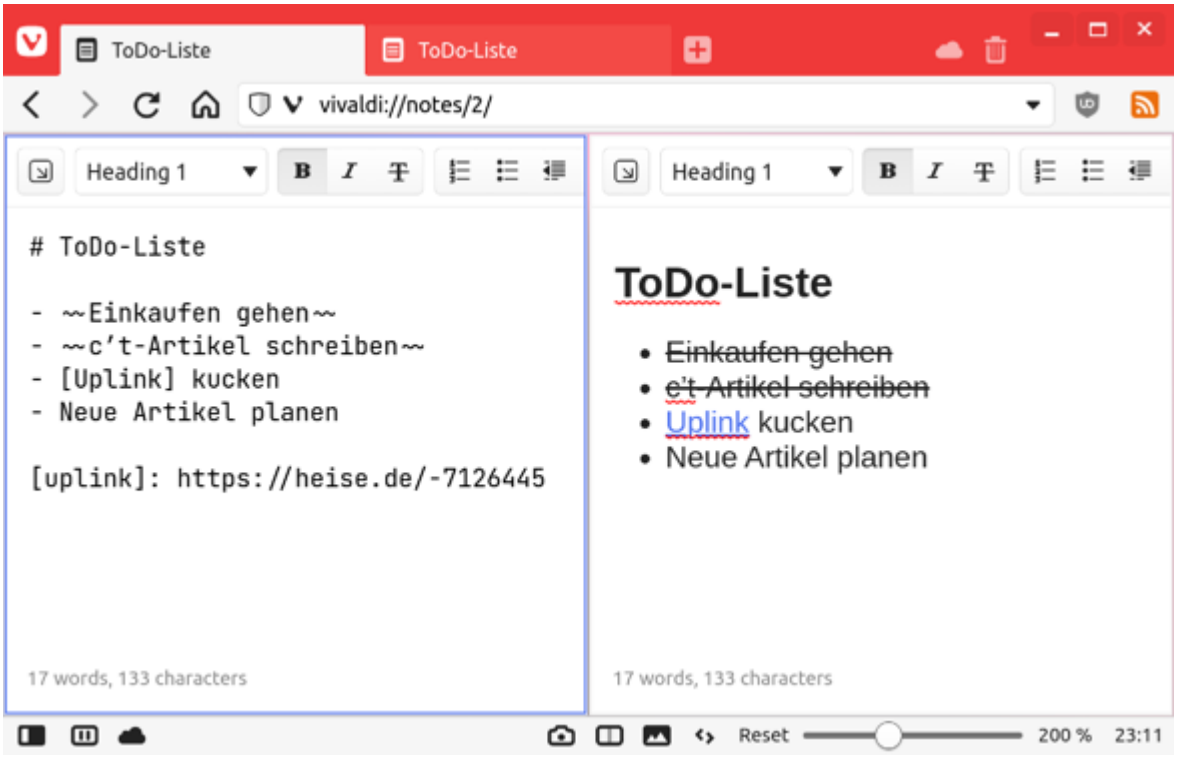
Pandoc’s GitHub-flavoured CommonMark?

Nicht überall, wo „Markdown“ draufsteht, ist das gleiche Markdown drin. Die ursprüngliche Version der Sprache wurde von John Gruber und Aaron Swartz aus der Taufe gehoben, um damit Artikel fürs Web zu schreiben. Mittlerweile findet Markdown allerdings in allen möglichen Systemen Verwendung und die immer neuen Einsatzzwecke luden zu allerlei Erweiterungen der Sprache ein. Hinzu kam, dass Grubers Dokumentation der Markdown-Syntax (alle Links unter ct.de/y5hr) an einigen

Stellen nicht ganz eindeutig ist oder von seiner Referenzimplementierung abwich.

Manche der entstandenen Varianten sind kaum dokumentierte Ad-hoc-Ideen von den Autoren der jeweiligen Software, andere – wie etwa „GitHub Flavored Markdown“ oder „Pandoc’s Markdown“ – sind weit verbreitete und sauber spezifizierte Erweiterungen der Sprache. Den Wildwuchs wieder einhegen soll „CommonMark“, eine erweiterte und präzisierte Markdown-Spezifikation. Das funktioniert nicht ganz: Die CommonMark-Spezifikation sollte möglichst kompatibel zu bereits existierenden Praktiken sein und geriet daher teilweise recht komplex. Infolgedessen weichen verschiedene Markdown-Implementierungen in Details von CommonMark und auch voneinander ab. Außerdem gibt es weiterhin viele Features, die über CommonMark hinausgehen und von System zu System unterschiedlich funktionieren.

Ungeachtet dieser Einschränkungen ist CommonMark viel wert, weil es eine relativ verlässliche Basis darstellt: Wer die dort spezifizierten Features kennt und nicht auf abstruse Weise miteinander kombiniert, der hat mit jeder vernünftigen Markdown-Implementierung Freude. Und wenn ab und an eine Kleinigkeit auf dem jeweiligen System nicht funktioniert, weiß der erfahrene Markdown-Autor, das Problem zu umschiffen. Je nach Anwendungsfall und Bedarf kann man dann immer noch nachschlagen, welche über CommonMark hinausgehenden Möglichkeiten die jeweilige Software bietet.



Markdown kann man an den verschiedensten Stellen nutzen, hier verschönert es den Inhalt der Notizen-Funktion im Browser Vivaldi.

Jeder Text ist Markdown

Markdown zu schreiben fällt grundsätzlich leicht, weil jeder Text Markdown ist – ungültige Syntax gibt es nicht. Wenn ein Markdown-Textabschnitt nicht anderweitig interpretiert werden kann, dann ist er eben genau das: ein Textabschnitt, genauer gesagt ein Absatz:

```
Ein Absatz in Markdown

Und noch einer, der sich über
zwei Zeilen erstreckt.
```

Einfache Zeilenumbrüche behandelt Markdown wie Leerzeichen, sodass man den Quelltext auch manuell umbrechen kann – wie beim zweiten Absatz in diesem Beispiel –, ohne dass diese Umbrüche die Ausgabe beeinflussen. Um einen neuen Absatz zu beginnen – der auch in der Ausgabe als neuer Absatz gerendert wird –, muss man eine Leerzeile einfügen – was auch im Texteditor deutlicher ist.

Um einen Zeilenumbruch auch in der Ausgabe zu produzieren, reicht es, im Markdown-Text die vorhergehende Zeile mit einem Backslash oder mehr als einem Leerzeichen zu beenden:

```
Ein Absatz mit einem\
Zeilenumbruch und␣␣
noch einem.
```

Um einen neuen Textabschnitt einzuleiten, schreibt man drei Sternchen (***), Bindestriche (---) oder Unterstriche (___). Markdown interpretiert dergleichen als „thematischen Bruch“, den die meisten Systeme als horizontale Linie rendern. Wer mag, darf auch mehr Zeichen setzen oder sie mit Leerzeichen auflockern:

```
*******

- - - -
```

Zweierlei Überschriften

Um eine Textzeile als Überschrift zu markieren, stellt man ihr ein oder mehrere Doppelkreuze (#) und ein Leerzeichen voran. Die Anzahl der Doppelkreuze (maximal sechs) gibt die Ebene der Überschrift an:

```
# Überschrift

## Unterüberschrift
```

Überschriften dürfen auch mit Doppelkreuzen enden, was manche Autoren schöner finden:

```
### Unterunterüberschrift ###
```

Markdown ist das egal, es ignoriert Doppelkreuze am Ende einfach und achtet auch nicht darauf, ob ihre Anzahl zur Zahl der öffnenden Zeichen passt.

Übrigens hilft ein Backslash, wenn Markdown ein Zeichen interpretiert, das einfach Textinhalt sein soll:

```
\# Normaler Text, beginnt mit #

# Überschrift, endet mit \#
```

Daneben kennt Markdown noch eine zweite, sehr augenfällige Syntax für Überschriften:

```
Überschrift
===========

Unterüberschrift
---------------
```

Solche Überschriften nennt man Setext-Überschriften, nach einem Textauszeichnungssystem von 1991, aus dem die Syntax übernommen wurde. Setext-Überschriften sehen zwar im Texteditor schön aus, sind aber eher mühsam zu tippen. Markdown begnügt sich zwar auch mit nur einem einzelnen Gleichheitszeichen oder Bindestrich (oder jeder anderen Anzahl), allerdings geht dann schnell die bessere Optik verloren. In jedem Fall kann man mit dieser Syntax nur zwei Überschrift-Ebenen auszeichnen.

Syntax für Faule

Bei einem anderen Feature lehnt sich die Markdown-Syntax ebenfalls an Bekanntes an: Zitate leitet man – wie in E-Mails – mit Größer-als-Zeichen ein. Auch das Verschachteln funktioniert:

```
> Zitat, das sich über
> mehrere Zeilen
> erstreckt.
>
>> Unter-Zitat mit
>> zwei Zeilen und einer
>>
>> # Überschrift
```

Die Leerzeilen vor dem (Unter-)Zitat und der Überschrift dürfen laut CommonMark-Standard entfallen, aber manch andere

Markdown-Implementierung besteht darauf. Grundsätzlich hilft es bei unerwarteten Ergebnissen oft, Elemente mit Leerzeilen zu trennen.

Wer seinen Text manuell umbricht, muss allerdings nicht jeder Zeile einzeln ein > voranstellen. „Markdown erlaubt Dir faul zu sein“, steht schon in der originalen Dokumentation von John Gruber. Bei Zeilen, die nur den aktuellen Absatz fortsetzen, darf man auf die Markierung verzichten. Das folgende Codebeispiel hat daher die gleiche Bedeutung wie das vorhergehende:

```
> Zitat, das sich über
mehrere Zeilen
erstreckt.
>
>> Unter-Zitat mit
zwei Zeilen und einer
>>
>> # Überschrift
```

Ebenfalls kaum gewöhnungsbedürftig – und ebenfalls mit einer eingebauten Abkürzung für Faule – ist die Art und Weise, wie Markdown Listen auszeichnet. Man stellt dem Text für einen Listeneintrag schlicht einen Listenmarker voran. Die Sprache erlaubt dafür Sternchen, Bindestriche oder Plus-Zeichen für unnummerierte Listen und Zahlen mit Punkten für nummerierte Listen:

```
* Listenpunkt
* noch ein Listenpunkt
* und ein weiterer

1. Eins
2. Zwei
3. Drei
```

Statt dem Punkt nach einer Zahl darf man auch eine Klammer setzen (1), 2), …). Übrigens beachtet Markdown nur die erste Zahl einer nummerierten Liste. Man kann also auch drei Punkte

mit 3., 10. und 07. nummerieren und erhält als Ausgabe eine Liste, die 3., 4., 5. hochzählt. Das hilft, wenn Listenpunkte immer wieder an neue Positionen wandern: Man nummeriert einfach jeden Punkt mit 1. und Markdown kümmert sich darum, dass am Ende eine ordentliche Aufzählung herauskommt.

Andere Zählvarianten kennt CommonMark nicht, aber viele Markdown-Implementierungen erlauben auch römische Zahlen (i., ii., …), Buchstaben (A), B), …) oder vollständige Umklammerungen ((a), (b), …). Was funktioniert, kann man einfach von Fall zu Fall ausprobieren.

Eine ebenfalls häufig verfügbare Ergänzung zu CommonMark sind Aufgabenlisten. Dafür folgen auf den Listenmarker zwei eckige Klammern, die ein Kästchen andeuten. Erledigte Einträge erhalten ein X in ihrem Kästchen:

```
- [ ] Aufgabe 1
- [x] Aufgabe 2
- [ ] Aufgabe 3
```

Bei der Ausgabe werden dann statt der Klammern ordentliche Kästchen und Häkchen angezeigt.

Wenn ein Listeneintrag mehrere Zeilen umfassen soll, dann rückt man die Folgezeilen einfach passend ein. Über Einrückungen werden Listen auch verschachtelt:

```
+ Listenpunkt und
+ ein langer Listenpunkt,
  der über mehrere
  Zeilen geht
+ Noch ein Punkt
  1. Ein Unterpunkt
  2. und noch einer
     > mit einem Zitat

     im Listenpunkt
```

Wie erwähnt ist die Leerzeile vor dem Zitat laut Standard optional, wird aber von so manchem System verlangt. Die

Leerzeile danach ist in jedem Fall nötig, weil Markdown sonst eine „faule“ Zitatfortsetzung erkennt.

Die genauen Regeln für die Tiefe von Einrückungen sind aus mehreren Gründen sehr kompliziert. Auf der sicheren Seite bleibt, wer Einrückungen so wählt, dass alle Marker einer Liste untereinander liegen und auch der Text jeder Zeile auf Linie beginnt – also so, wie es oben beim zweiten Listenpunkt und zweiten Unterpunkt der Fall ist.

Wie erwähnt ist auch bei Listenpunkten Faulheit gestattet und man kann die Einrückung komplett weglassen. Das funktioniert aber nicht für Überschriften und dergleichen, sondern nur, wenn eine Zeile schlichten Text aus der Zeile darüber fortsetzt:

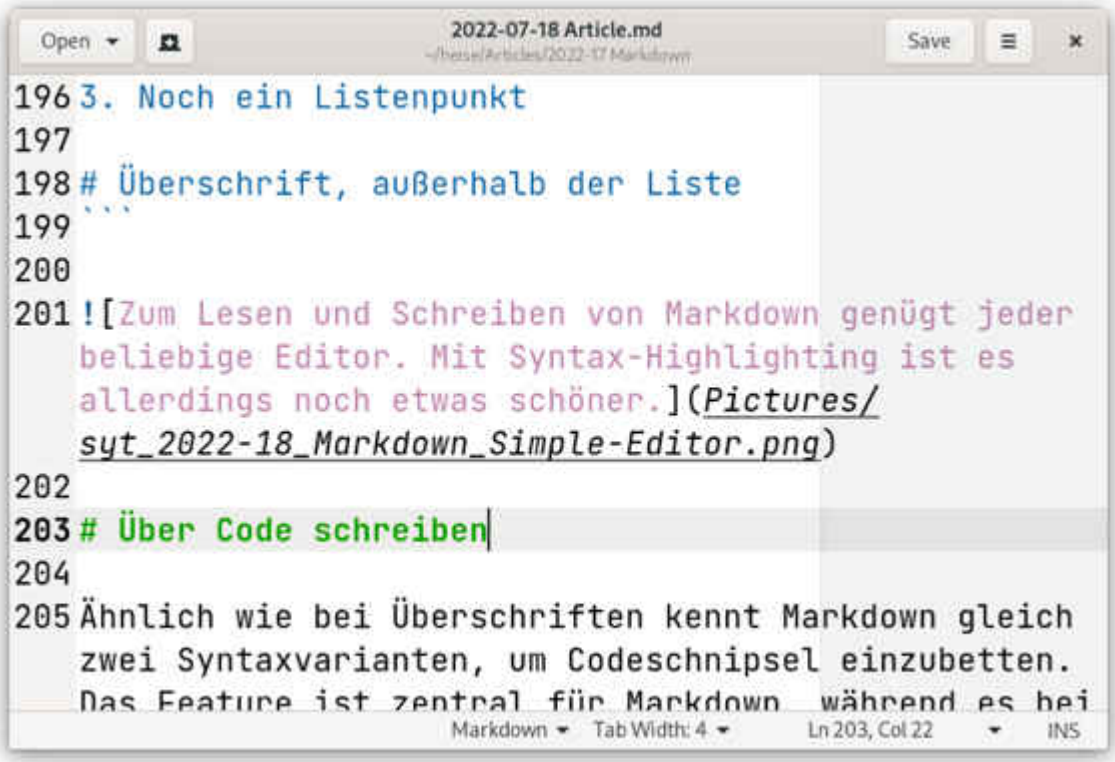
```
1. Ein langer Listenpunkt,
über zwei Zeilen.

2. Ein Listenpunkt mit

   # integrierter Überschrift

3. Noch ein Listenpunkt

# Überschrift, außerhalb der Liste
```



Zum Lesen und Schreiben von Markdown genügt jeder beliebige Editor. Mit Syntax-Highlighting ist es allerdings noch etwas schöner.

Über Code schreiben

Ähnlich wie bei Überschriften kennt Markdown gleich zwei Syntaxvarianten, um Codeschnipsel einzubetten. Das Feature ist zentral für Markdown, während es bei anderen Textverarbeitungen und Auszeichnungssprachen eher ein Nischendasein fristet. Die Sprache kommt häufig in der Softwareentwicklung zum Einsatz und Programmierer müssen eben allenthalben Code dokumentieren, kommentieren oder publizieren.

In der ersten und simpleren Variante rückt man Codeschnipsel einfach vier Leerzeichen weit ein. Den Inhalt eines Codeblocks fasst Markdown nicht an, sodass man dort nach Herzenslust #, * und alle anderen Zeichen verwenden kann:

```
    Codezeile
    Noch eine Codezeile
```

```
    # Keine Überschrift, sondern
    # ein Kommentar im Code
```

Eingerückte Code-Blöcke sind der Grund, warum man die meisten anderen Markdown-Elemente (wie Zitate, Überschriften et cetera) zwar mit Leerzeichen einrücken darf, wenn man will, aber höchstens drei Leerzeichen weit. Rückt man ein Element vier Leerzeichen weit ein, macht Markdown daraus einen Codeblock.

Überhaupt wird es schnell unübersichtlich, wenn man eingerückte Codeblöcke mit Elementen wie Listenpunkten kombiniert, die ebenfalls Einrückungen erfordern. Man kann solche Codeblöcke durchaus in Listen verschachteln und mit anderen Listeninhalten mischen, aber beim Schreiben artet das rasch in mühsames Leerzeichen-Abzählen aus.

Hier hilft die zweite Syntax für Codeschnipsel, die Codeblöcke nicht einrückt, sondern oben und unten mit expliziten Zeichen umzäunt („fenced code blocks“). So ein Zaun besteht aus drei oder mehr Gravis (`, oft auch „Backticks“ genannt) oder Tilden (~):

````
```
Codezeile
Noch eine Codezeile

# Keine Überschrift, sondern
# ein Kommentar im Code
```
````

Der abschließende Zaun muss mindestens so lang sein wie der öffnende. In aller Regel schreibt man sie gleich lang; diese Regel dient aber dazu, Codezeilen mit zum Beispiel drei Gravis im Codeblock zu erlauben, ohne dass sie fälschlicherweise den Codeblock abschließen: Man nutzt dann einfach vier oder mehr Gravis als Zaun – oder eben Tilden.

Umzäunte Codeschnipsel haben noch einen weiteren Vorteil gegenüber ihren eingerückten Pendants: Sie ermöglichen

sogenannte Info-Strings. Das sind Metadaten zum Codeblock, die man direkt nach dem öffnenden Zaun in dieselbe Zeile schreibt. Der Inhalt dieser Metadaten hängt vom konkreten System ab, für das der Markdown-Text verfasst wird. In aller Regel ist das erste (und oft einzige) Wort der Name der Programmiersprache, die im Codeschnipsel zum Einsatz kommt. Das dient zum Beispiel dazu, ein passendes Syntax-Highlighting zu aktiveren:

```
```javascript
alert('Hello World!');
```
```

Wer statt einem ganzen Codeblock nur schnell ein paar Wörter im Fließtext als Code markieren will, kann sie einfach links und rechts in je einen oder mehrere Gravis einschließen. Tilden funktionieren dafür nicht. Wenn der markierte Code selbst eine Folge von Gravis enthält, dann nutzt man zum Umfassen schlicht eine Anzahl an Zeichen, die nicht im Code als Sequenz auftritt (und trennt sie, falls nötig, mit Leerzeichen vom Inhalt):

```
Ein Text mit `Code`,
etwas ``Code samt ` Gravis`` und
nur einem Doppelgravis: ` `` `.
```

Fett formatiert

Ganz ähnlich wie kurze Codepassagen kann man Textstellen kursivieren und fetten, indem man sie mit einfachen beziehungsweise doppelten Sternchen oder Unterstrichen umgibt:

```
*Kursive* und **fette** Wörter,
sowie __fette__ und _kursive_.
```

Kursivierung und Fettung sind dabei lediglich die üblichen Darstellungsformen, die Auszeichnungen bedeuten eigentlich „betont“ und „stark betont“, was man sogar verschachteln darf:

```
Normal, _kursiv und **fett-kursiv**_.
Auch ***fett-kursiv***.
```

Etwas kompliziert werden die Regeln, wenn die Auszeichnungen nicht an Leerzeichen enden. Um einen Teil eines Wortes zu kursivieren, eignen sich zum Beispiel nur Sternchen (Wort*teil*kursivierung). Unterstriche funktionieren hier nicht, damit Markdown nicht häufige Konstruktionen wie Dateinamen mit Unterstrichen versehentlich auszeichnet. Wenn es zu einer falschen Auszeichnung kommt, hilft wieder der Backslash:

```
Ein Name_mit_Unterstrichen und
ein Name\*mit*Sternchen.
```

CommonMark spezifiziert keine weiteren solchen Formatierungen. Allerdings gibt es wieder einige Ergänzungen, bei denen es sich lohnt, einfach auszuprobieren, ob ein System sie unterstützt: Mitunter kann man mit einfachen Tilden Text tief- (~1~) und mit Zirkumflexen hochstellen (^2^). Recht häufig markieren doppelte Tilden Text als gelöscht; er wird dann durchgestrichen angezeigt. Manche Systeme erlauben auch, Text mit doppelten Gleichheitszeichen hervorzuheben. Er bekommt dann etwa einen gelben Hintergrund:

```
Text mit einer ~~Löschung~~ und
einer ==Hervorhebung==.
```

Links und verlinkte Bilder

Links setzt man in Markdown mit einer Reihe von Syntaxvariationen, die aus Klammern bestehen. Eine Möglichkeit ist, den zu verlinkenden Text in eckige und das Linkziel in runde Klammern zu setzen:

```
Webseite des [Magazins für
Computertechnik](https://ct.de).
```

Was man als Linkziel notiert, ist Markdown egal. Häufig handelt es sich um URLs oder Dateipfade – was in der Ausgabe funktioniert, hängt vom jeweiligen System ab. Zusätzlich zum Ziel kann man auch einen Link-Titel angeben. Den schreibt man einfach, von (mindestens) einem Leerzeichen getrennt, dahinter

und fasst ihn in Anführungszeichen oder runde Klammern ein:

```
[Linktext](Linkziel "Linktitel")
```

Bei einer Ausgabe als HTML-Code dient der Linktitel als Inhalt des title-Attributes des Links.

Linkziel und -titel direkt nach dem verlinkten Text anzugeben kann unübersichtlich werden, besonders wenn man sehr viele Links oder Links mit sehr langen URLs in einen Text einbettet. Abhilfe schaffen Referenzenlinks, die nach dem verlinkten Text nur ein kurzes Label angeben. Um es von einem Linkziel zu unterscheiden, notiert man das Label in eckigen statt runden Klammern.

Irgendwo anders im Markdown-Dokument (auch davor) definiert man eine passende Linkreferenz. Die wiederholt das Label (ebenfalls in Klammern) und gibt nach einem Doppelpunkt das Linkziel und den optionalen Titel an:

```
Webseite des [Magazins für
Computertechnik][CT].

[CT]: https://ct.de "c’t-Website"
```

Wenn das Label mit dem Linktext übereinstimmt, darf man es auch weglassen und die Referenz funktioniert trotzdem; die eckigen Klammern des Labels sind dann optional. Praktischerweise ignoriert Markdown Groß- und Kleinschreibung bei der Label-Zuordnung:

```
Webseiten der [ct][] und von [heise].

[CT]: https://ct.de
[Heise]: https://heise.de
```

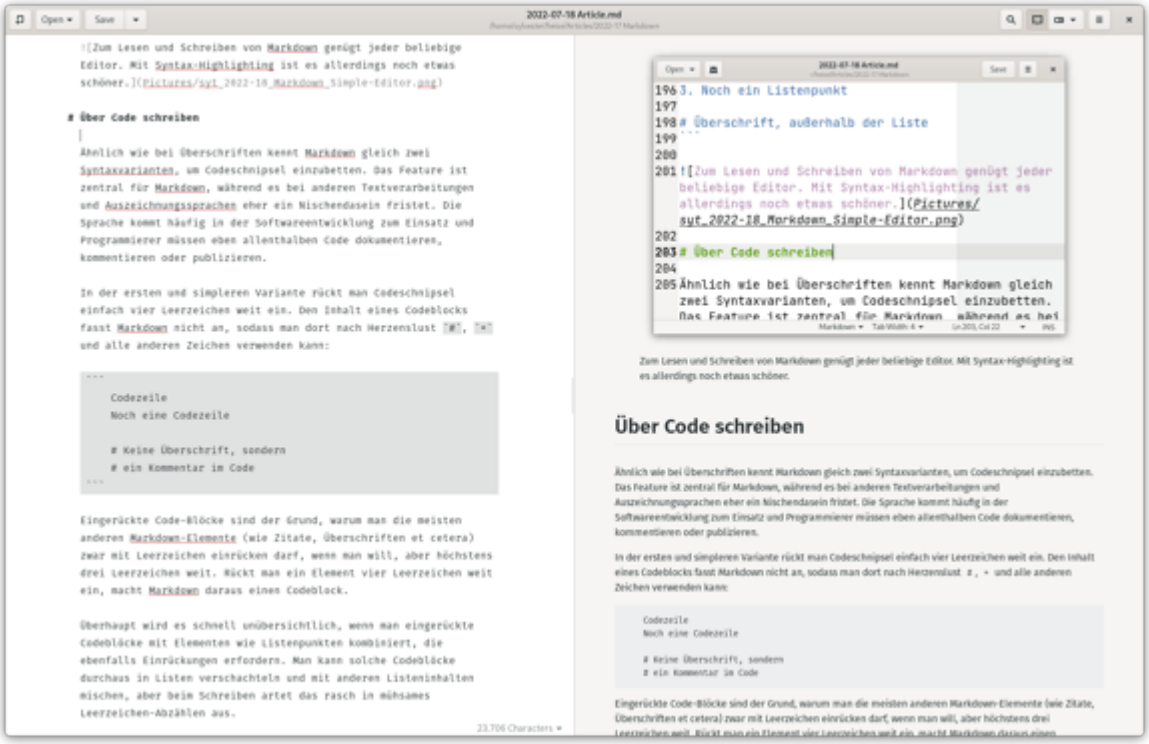
Als letzte Linkvariante gibt es noch eine Syntax für den Fall, dass Linktext und -ziel identisch sind. Um dann nicht https://ct.de schreiben zu müssen, kann man auch einfach das Linkziel in spitze Klammern setzen: <https://ct.de>. Einige Markdown-Systeme erfordern nicht

einmal das und verlinken automatisch alles, was sie als URL erkennen.

Eng verwandt mit Links sind Bilder – zumindest aus Markdown-Sicht: Eine Bilddatei kann man nicht sinnvoll in eine Textdatei einbetten und muss sie daher wie ein Linkziel referenzieren. In Markdown geschieht das über dieselbe Syntax, der man lediglich ein Ausrufezeichen voranstellt:

```
![Beschreibung](Pfad/zum/Bild-1.png)
```

Mit der Syntax von Referenzenlinks funktioniert das ebenso, nur die Variante mit spitzen Klammern gibt es nicht für Bilder. Anstelle des Linktexts steht eine Bildbeschreibung. Üblicherweise wird die als Alt-Text des Bildes ausgegeben, sollte also den Bildinhalt erklären. Allerdings weichen einige Markdown-Systeme davon ab und geben Bilder, die für sich alleine in einer Zeile stehen, anders aus. Die Beschreibung rendern diese Systeme dann als Bildunterschrift. Wieder probiert man am besten einfach aus, wie das jeweilige System diesen Fall handhabt.



Wer lange Texte in Markdown schreibt, sollte sich darauf spezialisierte Editoren ansehen. Die können – wie hier das

Programm „Apostrophe“ – zum Beispiel die layoutete Vorschau neben dem Markdown-Quelltext anzeigen.

HTML einbetten?

Das deckt die wesentlichen Aspekte von Markdown und CommonMark ab – bis auf einen. Wie eingangs erwähnt, war Markdown ursprünglich (nur) dafür gedacht, Texte für das Web zu verfassen. Der Output einer Markdown-Implementierung sollte daher HTML sein und schon im Eingabetext vorhandenes HTML konnte die Software einfach durchreichen. Das ist im Grunde immer noch der Fall; wer HTML beherrscht, kann es einfach im Markdown-Text notieren:

```
<div class="test">

*Kursiver Text*

</div>
```

Ob das Ergebnis wie erwünscht ausfällt, ist allerdings eine andere Frage. Zwar produzieren die meisten Markdown-Anwendungen tatsächlich HTML (zumindest als Zwischenschritt), aber häufig filtern sie ihre Eingabe – etwa aus Sicherheitsgründen – und man kann keine (beliebigen) HTML-Tags in das Ergebnis schleusen.

Außerdem sind die genauen Regeln zum Mischen von Markdown- und HTML-Code komplex. Im obigen Beispiel sind etwa die Leerzeilen unabdingbar. Durch sie wird die mittlere Textzeile als Markdown und nicht als HTML aufgefasst, sodass die Sternchen Wirkung entfalten – zumindest in manchen Implementierungen. Weil solche Sprachmischungen schnell implementierungsabhängige Ergebnisse produzieren, die man kaum noch durchschaut, sollte man sie eher als Notlösung betrachten: Kann man ausprobieren, wenn man mit Markdown-Syntax alleine einfach nicht ans Ziel kommt.

Tabellen

Als Beispiel für so einen Fall nennt die originale Markdown-Spezifikation Tabellen. Das ist zum Glück meistens nicht mehr korrekt, denn HTML-Tabellen sind lästig zu schreiben und schwer zu lesen. Viele Markdown-Implementierungen unterstützen eine simplere und viel übersichtlichere Syntax:

```
| Magazin | Zyklus   | Domain       |
| ------- | -------- | ------------ |
| c’t     | 14 Tage  | ct.de        |
| iX      | 4 Wochen | ix.de        |
| Mac & i | 8 Wochen | mac-and-i.de |
```

Allerdings sind Tabellen kein Teil der CommonMark-Spezifikation und es gibt zahlreiche Syntaxvariationen. Meistens darf man zum Beispiel die äußeren senkrechten Striche (|) auch weglassen. In der Regel kann man auch bestimmen, wie Tabellenspalten ausgerichtet werden sollen, indem man Doppelpunkte in die Trennlinie zwischen der Kopfzeile und dem Rest der Tabelle setzt. Beides zusammen sieht so aus (die erste Spalte wird linksbündig, die zweite zentriert und die dritte rechtsbündig ausgegeben):

```
Magazin | Zyklus   | Domain
:------ | :------: | -----------:
c’t     | 14 Tage  | ct.de
iX      | 4 Wochen | ix.de
Mac & i | 8 Wochen | mac-and-i.de
```

Außerdem bietet Markdown wieder mal eine Abkürzung für Faule: Man muss die Tabellenspalten nicht im Quelltext ausrichten, damit sie korrekt interpretiert werden. Das vereinfacht das Tippen erheblich, geht aber deutlich zulasten der Lesbarkeit:

```
Magazin | Zyklus | Domain
:- | :-: | -:
c’t | 14 Tage | ct.de
iX | 4 Wochen | ix.de
Mac & i | 8 Wochen | mac-and-i.de
```

Neben Tabellen gibt es noch viele weitere Markdown-Ergänzungen, die sich weder in CommonMark noch der originalen Spezifikation der Sprache finden. Dazu gehören zum Beispiel Fußnoten oder Textabschnitte, die – häufig am Anfang des Dokuments – Metadaten enthalten. Manche Systeme erlauben auch Emojis und sogar mathematische Formeln in LaTeX-Syntax im Markdown-Text.

Wer Bedarf an solchen Erweiterungen verspürt, konsultiert am besten die Dokumentation der Software, um die es geht. Im Regelfall ist das der schnellste Weg, um herauszufinden, was im konkreten Fall möglich ist und welche Syntaxvariante das System verlangt. Um einfach nur einen Blick über den Tellerrand zu werfen, eignet sich die Dokumentation von Pandoc (siehe ct.de/y5hr), einem System, das zahlreiche Ergänzungen in zahlreichen Syntaxvarianten versteht [7].

Fazit

Wer die Markdown-Basics beherrscht, dem steht in vielen verschiedenen Systemen eine einheitliche, praktische und schnelle Eingabemethode zur Verfügung. Zwar unterscheiden sich Implementierungen in Details, aber wenn gelegentlich etwas nicht wie erwartet funktioniert, dann hilft in der Regel ein Backslash, eine zusätzliche Leerzeile, um Elemente zu trennen, oder man vereinfacht die Dokumentstruktur etwas: Codeblöcke in Listen in Zitaten in Listen bringen nicht nur so manche Markdown-Implementierung in die Bredouille – sie sind auch einfach schwer zu verstehen.

Wie zahlreiche andere c’t-Artikel entstand auch dieser in einem Markdown-Editor. Das ging wie immer flott und einfach von der Hand und ist auch im Quelltext exzellent zu lesen – obwohl es wirklich verschärfte Bedingungen sind, mit Markdown-Syntax über Markdown-Syntax zu schreiben. *(syt@ct.de)*

1. Literatur
2. Stefan Wischner, Blitznotizen, c’t 13/2021, S. 82

3. Achim Barczok, Notiznetz, c’t 6/2021, S. 84
4. Liane M. Dubowy, Organisationstalent, Schreiben und organisieren in Notion, c’t 12/2021, S. 166
5. Andrijan Möcker, Tippgemeinschaft, HedgeDoc: Gemeinsam texten mit Markdown-Pads, c’t 13/2022, S. 164
6. Jan Mahn, Nach Art des Hauses, Rezeptdatenbank selbst hosten mit Tandoor Recipes, c’t 2/2022, S. 164
7. Anna Simon, Website-Wasservogel, Mit Pelican einfach schnelle Webseiten generieren, c’t 12/2022, S. 164
8. Jan Mahn, Formatautomat, Automatische Textumwandlung mit pandoc, c’t 7/2018, S. 168

Markdown-Spezifikationen: ct.de/y5hr