Introducing Markdown and Pandoc

Using Markup Language and Document Converter

Thomas Mailund

Apress®

Introducing Markdown and Pandoc

Using Markup Language and Document Converter

Thomas Mailund

Apress®

Introducing Markdown and Pandoc: Using Markup Language and Document Converter

Thomas Mailund
Aarhus N, Denmark

ISBN-13 (pbk): 978-1-4842-5148-5 ISBN-13 (electronic): 978-1-4842-5149-2
https://doi.org/10.1007/978-1-4842-5149-2

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail editorial@apress.com; for reprint, paperback, or audio rights, please email bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484251485. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author

Thomas Mailund is an associate professor in bioinformatics at Aarhus University, Denmark. He has a background in math and computer science. For the past decade, his main focus has been on genetics and evolutionary studies, particularly comparative genomics, speciation, and gene flow between emerging species. He has published *R Data Science Quick Reference*, *The Joys of Hashing*, *Domain-Specific Languages in R*, *Beginning Data Science in R*, *Functional Programming in R*, and *Metaprogramming in R*, all from Apress, as well as other books.

About the Technical Reviewer

Germán González-Morris is a polyglot software architect/engineer with 20+ years in the field, with knowledge in Java(EE), Spring, Haskell, C, Python, and Javascript, among others. He works with web distributed applications. Germán loves math puzzles (including reading Knuth) and swimming. He has tech-reviewed several books, including an application container book (Weblogic), as well as titles covering various programming languages (Haskell, Typescript, WebAssembly, Math for coders, and regexp). You can find more details at his blog site (`https://devwebcl.blogspot.com/`) or twitter account (`@devwebcl`).

CHAPTER 1

The Beginner's Guide to Markdown and Pandoc

Markdown is a markup language. The name is a pun, but where the humor might be atrocious, the language is not. The Markdown language lets you write plain text documents with a few lightweight annotations that specify how you want the document formatted. Such annotations are the defining characteristics of a markup language. Markup languages separate the semantic or content part of a document from the formatting of said document. The content of a document is the text, what should be headers, what should be emphasized, and so on. The formatting specifies the font and font size, whether headers should be numbered, and so on.

Markup languages have a stronger focus on semantic information than direct formatting as you would do with WYSIWYG (what you see is what you get) formatting. With markup languages, you might annotate your text with information about where chapters and sections start, but not how chapter and heading captions should be formatted. Decoupling the structure of a text from how it is visualized makes it easier for you to produce different kinds of output. The same text can easily be transformed into HTML, PDF, or Word documents by tools that understand the markup annotations. And because writing the text and formatting it are

T. Mailund, *Introducing Markdown and Pandoc*,
https://doi.org/10.1007/978-1-4842-5149-2_1

separate steps, you can apply one or more text documents to the same transformation program to get a consistent look for related documents, or you can transform the same document into multiple output formats so the same document can be put on a web page or in a printed book, for example. Most WYSIWYG editors can export to different formats, but they usually do not let you output to the same document type with different formatting, for example, output PDF files in A4, 6″ x 9″, and 7″ x 10″ with point size 11 in the first two and 12 in the last. With a Markup language, this is relatively easy.

Among markup languages, Markdown is one of, if not the, simplest. The annotations you add to a text are minimal, and most likely you will already have seen most of them if you occasionally use plain text files. For example, where you would use italic or boldface in Word, you would write ∗italic∗ and ∗∗boldface∗∗ in Markdown, and most likely you have seen this notation before. In my misspelled youth, I frequently used TeX/LaTeX and HTM-L/SGML/XML. I know people who cannot concentrate on the text body if it is full of markup information. With Markdown, the markup annotation is almost invisible, and they have no problem working with that. With Markdown you can generate documents in other markup languages, so you do not need to know them. If you want the full power to format your documents the way you want them, then I still recommend that you learn the other languages. You can use that knowledge to create templates (see Chapter 9), and then you only need to use, for example, LaTeX or HTML when writing the templates. You can then still keep your document in Markdown. One exception, where you still want to use LaTeX, is if you need to write math in your document. Then you need to write it in LaTeX; see Chapter 6.

You need a program for translating Markdown into other file formats. The tool I will use in this book is Pandoc. Pandoc supports basic Markdown and several different extensions. It also lets you define templates and stylesheets to customize the transformed files. Pandoc can

do more than translate Markdown files into different output files. It can translate from and to several different formats. I will only describe how you translate from Markdown to other formats. If you have an existing text in Word, for example, and you want to try out Markdown by editing that document, then you should be able to generate a Markdown file from the Word file, edit the Markdown format, and then translate the Markdown document back to Word.

CHAPTER 2

Why Use Markdown and Pandoc

If you are used to WYSIWYG editors such as Microsoft Word, you might reasonably ask why you should use Markdown files. You can write your document and format them any way you like, and you can export your document to different file formats if you wish. For short documents that you only need to format once and to one file format, you do not need Markdown. I will argue that Markdown is still an excellent choice for such documents, but it is for more advanced applications where it really shines.

For applications that are just as easy to handle with a WYSIWYG editor, plain text can be a better choice in situations where you need to share documents with others. A de facto file format for this is Word files, but not everyone has Word. I don't. I can import Word files into Pages, which I have, and export to Word, but I don't know what that does to the formatting. Everyone has an editor that can work on plain files, and with a plain text file, you know exactly what you are editing. If the text and the formatting are separated, then someone with more artistic skills can handle the formatting while I can write the text. One argument for Word might be tracking of changes. This is an important feature, but with plain text files, you can put them under real version control, for example, GitHub, and that is superior to version tracking.

If you need your document in different formats, for example, you might need to include your text in a printed progress report and also have it on a

T. Mailund, *Introducing Markdown and Pandoc*,
https://doi.org/10.1007/978-1-4842-5149-2_2

web site, then you can export the document to as many file formats as you need. If you need different typography for the different file formats, you might have to do substantial manual work. You might need to change all the document styles by hand, and in the numerous occasions where you need to make changes to your text, you need to change the styles for each file format more than once. If you separate style and text, you avoid this problem altogether.

Using a markup language to annotate your text makes it easier for you to distinguish between the semantic structure of a document and how it is formatted. In the Markdown document, you markup where headers and lists are, for example, but not how these should be formatted in the final output. The formatting styles are held in different files and you can easily transform your Markdown input into all the output file formats and styles you need. Furthermore, someone else can work on the style specification while you concentrate on the text. Your Markdown doesn't have to be in a single file either. You can split it into as many as you want, and then different authors can work on separate pieces of the text without worrying about how to merge files afterward. With version control, you can even work on the same file in parallel up to a point.

Separating Semantics from Formatting

Most documents have a semantic structure. Texts consist of chapters and sections, plain text and emphasized text, figures and citations, quotes, and lists. When we read a document, these semantic elements are visualized by different fonts, bold and italic text, different font sizes, and we do not directly see the semantic structure. Because we don't immediately see the structure, it is easy to forget that it is there.

Most word processors separate semantics from formatting. If you take care to use the formatting section when working on a Word document, then the semantic information needed to change styles, that is, the visual

representation of all semantic units (e.g., headings) is readily available. Separating the semantics of a document from its formatting is not an exclusive property of markup languages. However, when the separation of text and semantics is not enforced, there is a potential for error. If you decide to change the font size of level-two section headers, for example, you can easily do this, but you can equally easy highlight a single section header and reformat that, changing only that single header. That makes this particular header different from all the rest, and if you later modify the formatting of level-two headers, you won't be changing this one header. Great if this is on purpose; not great if this is not what you wanted.

With WYSIWYG editors, you can separate semantics from formatting, but it is easy to break this separation. With markup languages, you can also define some text elements as special and their format different from related items, but you have to do this explicitly so you cannot easily do this by mistake. Keeping the core text consisting of semantic elements and separate from formatting is vital in many situations. If you want to translate your text into both paper documents and web pages, you typically want the format to be different in the two resulting documents. If the core text only contains the semantic structure, this is quickly done, by having a different mapping from semantic elements to formatting information, typically called *templates* or *stylesheets* (see Chapter 9). With different stylesheets for different output formats, the formatting is tied to the output text rather than the input text (see Figure 2-1).