Markdown By Example

Tim Steinbach

```
_Emphasized text_
*Emphasized text*

__Strongly emphasized text__
**Strongly emphasized text**

Headline H1
===========

Headline H2
-----------

# Headline H1
## Headline H2
### Headline H3
#### Headline H4
##### Headline H5
###### Headline H6

    This
    Code block
    By four spaces

1. Ordered
2. List
3. Items

- Unordered
- List
- Items

+ Unordered
+ List
+ Items

* Unordered
* List
* Items

`Line of code`
``Line of code that includes a ` character``

![Alternate Text](path/to/image "Image title")

![Alternate Text][Image ID]
[Image ID]: path/to/image "Image title"

[Link Text](path/to/link "Link title")

[Link Text][Link ID]
[Link ID]: path/to/link "Link title"

<http://automatic-link-to-url.com/>
<Automatic@linktoemailaddress.com>
```

Markdown By Example

The Markdown-everything book

Tim Steinbach

This book is for sale at
http://leanpub.com/markdown

This version was published on 2012-11-24

This is a Leanpub book. Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing.

To learn more about Lean Publishing, go to http://leanpub.com/manifesto.

To learn more about Leanpub, go to http://leanpub.com.

Contents

Legal

All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.

All links are provided as a convenience and for informational purposes only. They do not constitute an endorsement or an approval of any of the products, services or opinions of the corporation, organization or individual. The author bears no responsibility for the accuracy, legality or content of the external site or that if subsequent links.

Sample

This file is only a sample ebook. It includes one free chapter of the book Markdown By Example[1] by Tim Steinbach. The entire book can be bought on the book's website.

If you enjoy this sample chapter, please consider supporting the author!

[1] http://markdownbyexample.com/

7 - Simple text

This chapter contains:

Relevant source folders:

- /readme/

Simple text

Since Markdown is supposed to make reading and writing files easy without stripping the author of the ability to create well formatted output, a Markdown file is all about its contents and as little distraction as possible from formatting. We will take a look at most of the formatting elements available for regular Markdown files. A full syntax documentation can be found at DaringFireball[2].

In this chapter we will create a README file for a fictional piece of software. By the end of it, we will have used quite a few different ways of formatting text without ever shifting our focus from content to the overhead of formatting that is usually required with HTML or TeX. At all times our README will be easy-to-read and still create a well-structured result, ready to be published with our software.

[2] http://daringfireball.net/projects/markdown/syntax

Headlines

The first step to our first Markdown file is - naturally - creating a new file. It should use the file extension .markdown. GitHub has its own .md extension, which can be used instead. Once we have created the file, it is time to edit it. The first thing we should do is create a headline for the entire document. The first and largest headline is called an H1, the second level is H2, and so forth up to H6. Headlines can be created in two different ways: H1 and H2 can just be underlined by equals signs or hyphens, such as this:

Code 7.1: Headlines

```
1 Headline 1
2 =========
3
4 Headline 2
5 ----------
6
7 Regular text
```

For this method of creating headlines, the number of underlining characters is irrelevant. While this is a great way of making headlines obvious in your document, you are limited to H1 and H2 headlines.

For H3 through H6 you would have to mix the second way of creating headlines with the aforementioned underlining method, or decide to use the following notation exclusively. Headlines can also be created by prefixing a number of pound signs (#) followed by a space to the title. The specific number of pound signs used defines the header level. Code 7.1 from above could also look like this, without changing the final result:

Code 7.2: Headlines using the pound symbol

```
# Headline 1

## Headline 2

Regular Text
```

Eventually, it is up to the author to decide which style is preferred. It should be noted that for the rest of this book, we will use the pound symbol method.

The following example is designed to include the most important portions of the Markdown syntax. First, we will emphasize words, some by printing them in bold face. Secondly, we will talk about ordered and unordered lists. After that an image will be added to our README, before making use of hyperlinks and code blocks.

Code 7.3: Headlines for README

```
# GoodTool

## What is GoodTool?

## Features

## Download

## Installation

## Screenshot

## API example

### Access to API factory (Java)

### Usage of API factory (Java)

## Contact
```

We can see that our software is called GoodTool, which is the title for the first headline. We have a number of second-level and two third-level headlines.

Text emphasis

We begin by filling the headline “What is GoodTool?” with contents. Let the description be the sentence “GoodTool makes system monitoring easier than ever before and is available for Windows and Linux operating systems”. Nothing is easier than this as we simply enter the sentence into our README file. But we would like to emphasize that this is the easiest monitoring tool ever by printing the words “ever before” in bold face. Additionally, the operating systems Windows and Linux should be emphasized, but not as much as the “ever before”.

Markdown once again allows us to choose between two different ways of emphasizing text. We can either use underscores (_) or asterisks (*). Surrounding text with one of those characters causes regular emphasis of the given text. Doubling the surrounding characters causes a stronger emphasis, usually bold face print.

Knowing this, it is now easy to accomplish our wishes from above with the following Markdown code:

Code 7.4: What is GoodTool?

```
## What is GoodTool?

GoodTool makes system monitoring easier than
**ever before** and is available for _Windows_
and _Linux_ operating systems.
```

While it is up to the author to choose underscores and/or asterisks, this book will use underscores for simple emphasis and double-asterisks for strong emphasis. This way the author's intention becomes very clear and sometimes it is hard to see the difference between single or double underscores. The above mentioned convention clearly avoids this.

Lists

Next up is the features headline. Here we would like to list some of the most important features of our software. Lists are generally divided into two categories: ordered lists and unordered lists. The former usually makes use of Arabic or Roman numbers, the latter can use any kind of character. For our features list we will not need an ordered list because it does not really matter in which order we list them (Markdown does keep the order of items in your list, but they do not need to be numbered). Unordered lists can be created by using one of the prefixes for unordered lists for each item the list shall contain. Asterisks (*), hyphens (-) and pluses (+) are completely interchangeable, but have to be followed by a space. The list of our features could therefore look like this:

Code 7.5: List of features

```
## Features

* Fast
* Secure
* Easy-to-work-with GUI
```

This will create an unordered list of our top three features. We could have also used hyphens or pluses. For the “Download” paragraph we will do just that, the actual links to our software will be added later, once we learn about hyperlinks.

Code 7.6: List of versions

```
## Download

- Version 1.0
- Version 0.9
- Version 0.8
```

So far, this is what our *README.markdown* should look like: (plus the additional headlines that we have not yet worked with)

Code 7.7: State of README after adding list of downloadable versions

```
# GoodTool

## What is GoodTool?

GoodTool makes system monitoring easier than
**ever before** and is available for _Windows_
and _Linux_ operating systems.

## Features

* Fast
* Secure
* Easy-to-work-with GUI

## Download

- Version 1.0
- Version 0.9
- Version 0.8
```

For the installation instructions included in our file, we would like to use an ordered list. This makes sense as the user will be guided through the process and should follow the instructions one step after another. Luckily, ordered lists are extremely easy in Markdown. All we need to do is use Arabic numbers as prefixes. Our installation instructions could then look like this:

Code 7.8: Installation instructions

```
## Installation

1. Download GoodTool
2. Execute .exe or .sh file
3. Follow screen instructions
```

The ordered list would automatically be converted into the corresponding output elements.

Images

Next, we will need to add a screenshot to our README file. This is probably the first thing you would not have added to an ordinary *README.txt* file. After all, a text file is supposed to contain nothing but text. The fact that Markdown files will have to be parsed and converted comes in handy here, as images will not be visible while editing our file but will be present in the resulting output. As we will see, the syntax for adding images is very similar to that of hyperlinks. Both somewhat connect to external content and therefore have almost the same notation.

An image link consists of three parts: the path to the actual image file, a text used in case the image cannot be found (alt text), and an image title. The image path is obviously required and so is the alt text, because the image file might have been moved or the given path may be wrong. An image's title attribute, however, is optional. The title is usually converted into a mouseover text in the resulting output of our Markdown file.

The screenshot of our GUI is called GoodToolGui.png and will be added to our README like this:

Code 7.9: Adding screenshot

```
## Screenshot

![GUI Screenshot](GoodToolGui.png "Screenshot")
```

An exclamation mark suggests that the following is an image. The alt text for our image is in the square brackets, while the parentheses contain the path to our image and the optional title.

If an image is to be used on several occasions there is an easy way of dealing with this: image references. Instead of using an image's path and title each time we would like to display said image, we can define it once and use references to the definition. References are usually defined at the very bottom of a document and have a syntax consisting of ID, path, and title. Our screenshot could have been defined as a reference:

Code 7.10: Screenshot as image reference

```
## Screenshot

![GUI Screenshot][screenshot]

[screenshot]: GoodToolGui.png "Screenshot"
```

We have replaced the parentheses in our image tag with the reference ID of the image we would like to use. At the bottom of our document we have then defined said reference ID with the path and title that we had previously used inside the parentheses of our image tag. Now we could very easily reuse the screenshot reference without having to remember the image's path. And should its path ever change - for example, if somebody moved all images into a separate folder - it will only have to be altered once. There is no need for a risky "search and replace all" operation in your text editor. Since we will only have a single image in our example, we opt for the first version of the image tag, which saves us from having to add the reference to the end of our file.

Hyperlinks

Hyperlinks (or just links) are references to other content. This other content might be an anchor in the same document or external contents such as an Internet URL to a website or an image. For now, we will only look at links to external content because Markdown does not directly support internal anchors. These have to be realized by adding custom HTML code to our file, something that will be covered in the chapter Markdown and HTML.

In our README file, we need links to external contents under two headlines: Downloads and Contact. While linking directly to files under the Downloads heading, we will link to our website and offer a link to our email address under Contact. Creating a hyperlink is very simple and should already seem familiar since it closely resembles the syntax used for images (and vice versa). Earlier, we added a list of downloadable versions. The last of the three, version 0.8, we will use to link directly to the 0.8 file.

Code 7.11: Directly linking to a file

```
## Download

- Version 1.0
- Version 0.9
- Version [0.8](http://rmrf.eu/
gt/0_8.zip "Link to 0.8")
```

As we can see, creating a hyperlink looks just like the image tag, except there is no exclamation mark prefix. This is the only difference between the two! Again, the link title is optional.

Similar to our screenshot example, we may also want to create references to our links and use those instead of directly embedding the URL into our link. Just like the reference to an image, a link reference works by creating an ID and defining it (usually at the end of the document). For the remaining two versions we define the following links:

Code 7.12: Linking directly and by reference

```
## Download

- Version [1.0][2]
- Version [0.9][1]
- Version [0.8](http://rmrf.eu/
gt/0_8.zip "Link to 0.8")

[1]: http://rmrf.eu/gt/0_9.zip "Link to 0.9"
[2]: http://rmrf.eu/gt/1_0.zip "Link to 1.0"
```

As we can see, the parentheses have been replaced by a second pair of square brackets. Inside these we find reference IDs, which have been defined not unlike an image would have be defined. It is not necessary to number the links but you can use any ID you wish. We could very well have named the links [Version10File] and [Version09File] instead. The only requirement with reference IDs is the fact that they must be unique within the document.

When using references, IDs must be unique among hyperlinks and pictures. There cannot be an image using the same ID as a hyperlink.

Sometimes it is not necessary to hide a link behind a title and the actual URL should be displayed to the reader. This might be the case when referring to a website or an email address. Under our Contact headline we will be using "Automatic Links", hyperlinks whose URL and title are identical. We would like the reader to know the URL to our website as well as our email address, instead of requiring them to click on the link title. This could be achieved by defining a link with both the title and target location to the same value, using one of the linking methods we have used before. But using automatic links, the work is done for us:

Code 7.13: Contact using automatic links

```
## Contact

Go to our website <http://rmrf.eu/>
or email to <steinbach.tim@googlemail.com>
```

By surrounding the URL and email address with angle brackets, we define them as automatic links. These will be turned into just what we indented - Links with identical title and location.

Code

The last thing we would like to add is an example on how to use our tool's API for programmers who would like to extend or simply control the application. First, we will provide a short one-liner that will show the reader how they can acquire access to our API. Code statements which are only a few lines long, can easily be expressed in Markdown by surrounding them with grave accents (‘) aka “backticks”. Generally, this is the preferred way of expressing code statements which can be fit into a single line. In the case that source code contains backticks, the ones defining the code statement will be doubled. For longer statements, there is a more readable way of creating code blocks - ordinary indentation. All lines that have been indented by exactly four spaces are considered part of a code block.

Code 7.14: Code blocks in Markdown

```
### Single line statements
`Code statement`
``Code statement that includes ` character``

### Code block
    Code Block
    Indented
    By Four
    Spaces
```

If there are multiple code blocks without any regular text in between them, they can be split with the end-of-block (EOB) character, since they would otherwise all be considered part of the same code block. This is important as Markdown takes pride in keeping the resulting HTML well-formed and structured, and having multiple independent code examples within the same block would not conform to being well-formed and structured at all. The EOB character is a circumflex (^) in an otherwise empty line.

Code 7.15: EOB character separated code blocks

```
### Multiple code blocks
    Code
    Block1

    Still Code Block 1
^
    Code Block 2

    Still Code Block 2
```

In our own README file we could create the API examples as follows: Accessing the API is a one-line statement, we will surround with backticks. The second example will be an actual use case of the API, which will contain more than one line. We will therefore prefer the indentation syntax for the example with multiple lines of code.

Code 7.16: API examples

```
## API examples

### Access to API factory (Java)

Get the API factory:
`GoodTool api = GoodTool.getAPI();`

### Usage of API factory (Java)

Set monitoring of only one CPU thread:
    GoodTool api = GoodTool.getAPI();
    api.setMonitoredThreads(1);
```

And this is it, the snippets will now be treated as code blocks. This is particularly important when applying CSS styles to code blocks. For more information, please read the HTML/CSS chapter.

Output

At this point, our README file is complete and this chapter is coming to an end. We have used headlines, used emphasis to highlight parts of our text, listed items in ordered and unordered lists, added images, links, and even source code. Now it is time to find out what the result of our (not so) hard work is. We run our Markdown file against bluecloth (one of the Ruby gems we installed earlier) and have a look. In order to run bluecloth and fetch the resulting HTML into a file (it sends the HTML directly into our Terminal session), we need to call it with our Markdown file being the first parameter and its result being streamed into a new file.

Code 7.17: Bluecloth will generate HTML code

```
1 $ bluecloth README.markdown > README.html
```

Provided the README file is in our current working directory (if not, cd into it), we will get a nice HTML file. Open it with your browser and it should look something like Figure 7.1.

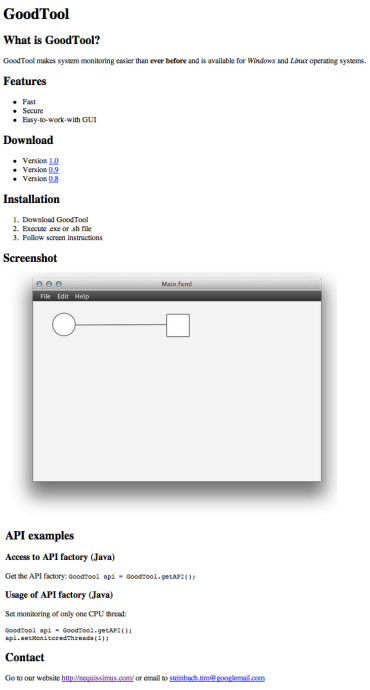

Figure 7.1: Resulting output

Not so bad after all, is it? And our Markdown file is still perfectly readable, unlike the HTML file created by bluecloth!

If you still have trouble with Markdown syntax, there is an additional example called “instructions” in the GitHub repository for this book. In this example there are two files, one with instructions and one with a

possible solution. At this point it should be noted that comments in Markdown use the same notation as HTML comments (because they are effectively HTML comments) as the file for you to work on is full of comments.

The next chapter will be a “cheat sheet”, a short recollection of the Markdown syntax, short and compact. This way there is no need to flip through all the past chapters trying to find a certain formatting syntax.

The following chapter will then use the README we just created and make modifications by using HTML and CSS directly. This will allow us to effectively design the output our Markdown converter produces. This chapter will also serve as an introduction to the rest of the book, which makes heavy use of not only Markdown but also HTML, CSS, and even JavaScript.