MIDDLESEX Community College

Tools and Technologies for Tech Writers 2023

Week 6

Markdown

Notices

This document was prepared as a handout for the Middlesex Community College Tools and Technologies for Technical Writers class, Winter semester 2023.

Prepared by Zoë Lawson, course instructor.

Contents

Paths

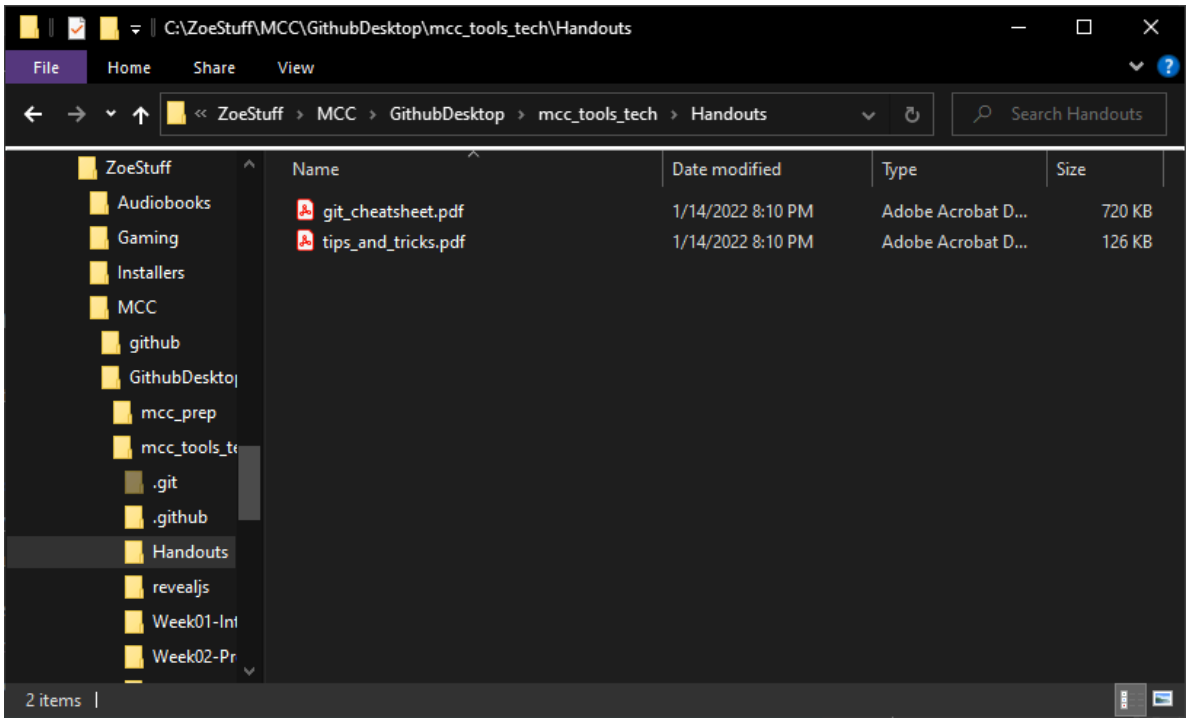
When working with computers, you often have to "write directions" on where to find a file. These are called 'paths'.

Computers have tons of information in them. This data and the tools to view and use the data is organized for humans into files and containers usually called folders or directories. The actual bits and bytes may be scattered all over a harddrive, but logically, computers present this data to us as files and folders. Files can be in folders, but folders cannot be in files. Folders can be nested inside of other folders.

Humans need to be able to find files and applications, so we need some sort of addressing system. Computers use bits and bytes and pointers n' things that are very hard to remember, unless you like remembering hex codes. So, someone figured out that you can write out the "path" to the file using folder names.

In the Unix world, someone decided to use forward slashes (/) as the separator between folder and file names. To avoid copyright infringement (or to be annoying) DOS (i.e. Windows) uses back slashes (\). This has caused all sorts of confusion and annoyance and bugs in the computer world ever since.

Most computer languages like to use forward slashes. Notice URLs, they use forward slashes. Notice all the paths in GitHub, they use forward slashes. Often, when linking to files for cross-references or images, you use forward slashes.



This is a screenshot of my GitHub Desktop folder location for this handout. You can see the path in the title bar: `C:\ZoeStuff\MCC\GithubDesktop\mcc_tools_tech\Handouts`. Since this is

a Windows system, it's using backslashes. In the Navigation Pane/Tree View, you can see all the folders and how they nest.

If I want to know the precise location of this handout on my computer, it is `C:\ZoeStuff\MCC\GithubDesktop\mcc_tools_tech\Handouts\tips_and_tricks.pdf`. This is called a *fully qualified path*. This tells me exactly where the file is, on my computer.

This is great...as long as I am working on my computer. If you want to find the file on your computer, it's not quite as useful.

Sometime people use *variables* in path descriptions. Common formatting conventions include italic text or surrounding with angle brackets. So I might write about this location as `<GitHub Desktop Repo Location>/mcc_tools_tech/Handouts/tips_and_tricks.pdf`. The string `<GitHub Desktop Repo Location>` indicates whatever directory I picked for cloning the repository. That could be the default of `C:\Users\User Name\Documents\GitHub` or the path I picked, `C:\ZoeStuff\MCC\GitHubDesktop`.

If you pick a formatting convention for variables, make sure you include some non-formatting related indicator. If you just make variables italic, a screen reader may not recognize the font change. If you include something not format related, such as adding the angle brackets (< >), it makes the difference more accessible.

This format of writing paths is extremely common in technical documentation. The only difficulty is standardizing how you refer to your variables so you don't have a mishmash of *Install_dir*, *Product Installation Directory*, and *Installation_Dir*.

While using string variables is great for tech doc, it's not super useful for computers. Yes, there are variables in programming languages, and you absolultely use them in paths in programming, but generally speaking, writing tech doc isn't programming. So there's another type of path you use all the time, a *relative path*.

A relative path is directions to a file based on where you're starting.

- `filename.txt` - the file is in the same folder as where you are starting.
- `directory/filename.txt` - the file is in the directory folder. The directory folder is in the same folder where you are starting.
- `../differentDirectory/filename.txt` - the file is in the differentDirectory folder, which is in the "parent" folder to where you are starting. With a .., you go "up" a folder.

For example, I'm starting in the Handouts folder and I want to get to the Week 2 homework folder. The Week 2 folder is not in the Handouts folder, so I go up a folder.

```
../
```

This puts me into the `mcc_tools_tech` folder. Goody! The Week 2 folder is in this directory, and I want to go into it.

```
../Week02-ProgressiveInfo
```

Now that I'm here in the Week 2 folder, there's the homework folder I want. So the relative path from the `Handouts` folder to the Week 2 Homework folder is:

```
../Week02-ProgressiveInfo/Homework
```

This relative path only works from certain locations, such as from the Handouts directory. However, this path will always work from the Handouts directory to the Week 2 Homework folder in the mcc_tools_tech repository, no matter what computer I'm on, or where you clone the repository.

Learning how to make relative paths between files is really useful. You use them all the time for cross-references and links to images.

Markdown basics

These are the basics for GitHub Flavored Markdown.

Paragraphs are just text.

- To break a paragraph, you have to have a blank line between.

```
this
will
render
as
a
single
paragraph

This is a new paragraph
```

- You can indent a paragraph, but you have to make sure you have the correct number of spaces. Start with at least five.
-

```
This is a paragraph with no preceding spaces.

     This is a new paragraph preceded by five spaces, so it should be
 indented.

  This is a new paragraph with only two preceding spaces, so it
 probably won't be indented.
```

You can do various types of inline formatting by wrapping the text with basic characters.

- Use a single underscore or asterisk to mark italic (or emphasized) text.

```
_This_ or *this* will be italic.
```

- Use two underscores or two asterisks to mark text bold (or strong).

```
**This** or __this__ will be bold.
```

- For monospace font, usually used for code samples, use the backtick `.

```
Make `this text` monospace.
```

- While not usually used for technical writing, you might find using two tilde for drawing a line through text.

```
Use the tilde to ~~strikethrough~~ text.
```

Links use square brackets and parenthesis. The link text goes into square brackets, followed by the URL in parenthesis.

```
[This text will be a link](http://www.middlesex.mass.edu)
```

Since markdown is just a text file, if you want to insert an image, you pretty much just make a link to it, just put an exclamation point before it.

```
![Put alt text here](../relative/path/to/image.png)
```

It may seem strange to put text in with an image, but it serves two important purposes:

- Alternative text - if something goes wrong and the image doesn't display, this text displays instead.
- Accessibility - if you use a screen reader, this text explains what the image contains.

Headings are identified by octothorps (aka pound sign aka hashtag). The symbol must be at the beginning of the line, and it must be followed by a space. The number of octothorpes equals the heading level.

```
# Heading 1
## Heading 2
### Heading 3
#### Heading 4
##### Heading 5
###### Heading 6
```

For unordered lists, use an asterisk, hyphen, or plus sign followed by a space.

```
* Bullet
- another bullet
+ another bullet
* these should all render the same
```

For ordered lists, use the number one followed by a period and a space.

```
1. This is the first numbered item.
1. This is the second numbered item.
1. It gets weird repeatedly using one plus a period and a space.
1. But it gets rendered as increasing numbers.
```

You can mix and match and nest the different types of list items. You will have to be careful with the spaces to indent properly.

```
1. This is a numbered item.
   This should be text aligned with the numbered item.
1. This should be the next numbered item.
   * with a bulleted list nested underneath
   * and another bullet
```

You can do simple tables. Columns are separated by pipes, |. A header row can be separated by a line of hyphens.

```
Heading 1 | Heading 2 | Heading 3
----------|-----------|----------
cell 1 | cell 2 | cell 3
The length of each cell | can be a dynamic thing |
```

Markdown resources

Markdown Guide	https://www.markdownguide.org/
CommonMark	https://commonmark.org/
GitHub Flavored Markdown	https://guides.github.com/features/mastering-markdown/ - Quickstart https://github.github.com/gfm - The specification.
Markdown Extra	https://michelf.ca/projects/php-markdown/extra/

All you need is a text editor. Notepad++ works, but it doesn't provide any type of rendering.

Since we're using GitHub Flavored Markdown, any `.md` file we upload into GitHub is rendered as HTML.

However, if you want to see real-time rendering of your lightweight markup, you can search for a markdown editor.

Originally, I had used https://typora.io/, but that was before it moved out of beta. It now costs $15.

I just tried https://ghostwriter.kde.org/ and it works well.

However, it doesn't have an easy-to-install version for the Mac. If you want a markdown specific editor for Mac, the most popular seems to be https://macdown.uranusjr.com/.

Week 6 Homework

1. Answer the questions in `relativePath.md`.

   Make a copy of the file with your name in it, and upload to the homework folder.
2. Document how to do a thing using GitHub Flavored Markdown.

   Follow the list of things that you will need to use when ever you document anything.

   - Include different headings.
   - Include different paragraphs.
   - Include ordered and unordered lists. (Preferably with a paragraph aligned with a list item.)
   - Include a table.
   - Include links (internal or external) (A linked TOC works)
   - Have some inline formatting.

   Make file(s) with the `.md` extension in the homework folder. You can make a subfolder if that's easier (i.e. you have multiple files).

Optional thought: DITA can take markdown as an input. You may be able to eventually combine assignments.