# Major Workshop Application for ABLE 2022

## Application Checklist

☒ Application deadline: November 30th.

Application is a PDF or MS Word document that includes:

☒ Workshop title and name(s)of presenters

☒ Abstract describing the proposed workshop (300 words maximum)

☒ Participant handouts (see note to reviewers below)

☒ A list of equipment and supplies needed for a class of 25 students

☐ Notes for the Instructor (will be completed post-workshop)

☒ Sample results (available from workshop repository)

☒ A description of how you will present the laboratory to conference participants

☐ Student/TA evaluations of this exercise as a learning experience (not applicable)

## Notes to the Reviewers

1. I think the current title of the workshop is the best one to attract participants. I would appreciate your suggestions for improving it.
2. Not all of the participant handouts/exercises are finished. I am re-writing and field testing them currently. If a handout is incomplete, I put **DRAFT** below the title, and either described what is missing or included links to external documentation of the procedures to illustrate what the workflow will be.
3. To show the versatility of Markdown, I composed the handouts for this workshop in a Markdown document, then converted it to MS Word for final formatting and submission. The original “.md” file is available to participants from this workshop’s [ABLE 2022 GitHub repository](https://github.com/adanieljohnson/ABLE_2022_Workshop).

# Major Workshop Proposal: Building an Open-Access Writing Resource Guide Using GitHub and Online Tools

## Presenter

A. Daniel Johnson, Wake Forest University, Department of Biology, 1834 Wake Forest Road, Winston-Salem NC 27109, USA. ***johnsoad@wfu.edu***

## Abstract

Scientific writing helps students learn to state problems and present claims precisely, summarize evidence to support those claims, and demonstrate their reasoning. Regular practice with coaching-oriented feedback is the most effective way to develop those process skills. In 2021 we presented our model for teaching scientific writing in multi-section introductory biology courses. One of our core tools is a standardized Scientific Writing Resource Guide that supports both students and instructors across multiple courses.

To make our Resource Guide easier to maintain and move between formats, we converted it into Markdown, a lightweight but versatile markup language that lets authors write well-structured text once, then output it in multiple formats. Markdown text can be converted directly to HTML for web pages, exported as Word documents, or organized into rich online books.

Before the workshop participants will register for a free GitHub account (a no/low cost, secure collaborative space that works seamlessly between institutions.) During the workshop participants will copy a template for our Guide to their account, use Markdown code to edit or create pages, then convert their work into formatted individual documents. We will demonstrate how to use R Studio to assemble Markdown documents into books, and GitHub to manage authorship and document approvals in a shared writing project.

Participants will leave with their own copy of the Science Writing Resource Guide which they can revise to match their courses, tools for writing web-ready documents offline in Markdown, and a GitHub account that they can use to build books or share pages with others. Those interested can learn how to contribute to our public edition of the Writing Resource Guide, and how to launch book projects of their own.

**Keywords**: scientific writing, writing guidelines, collaborative online project, multi-course resource, markup language

## Introduction

### About the Writing Resource Guide

Scientific writing helps students learn to state problems and present claims precisely, summarize evidence to support those claims, and demonstrate their reasoning. Regular practice with coaching-oriented feedback is the most effective way to develop those process skills. For many years, one of our essential tools for teaching scientific writing has been a standardized Writing Resource Guide that supports students across multiple courses.

In 2021 we revised and updated our [Writing Resource Guide](https://github.com/adanieljohnson/SWP_student_writing_guide) with the aim of supporting two audiences: undergraduates learning the craft of scientific writing, and biology instructors who either teach scientific writing to undergraduates or supervise teaching assistants who do. Our Resource Guide focuses on writing a lab report that models a journal article because the same components are used in most other forms of scientific communication too. The general format conforms to the ***Council of Science Editors*** (8e) standards, with some modifications to make writing easier for students just starting out.

No single guide can meet the needs of every possible audience. A comprehensive resource addressing multiple STEM writing styles would be too cumbersome for students to use. Instead we have released our Guide as a starting template [under terms of a Creative Commons BY-NC-SA 4.0 license](http://creativecommons.org/licenses/by-nc-sa/4.0/). Instructors can edit, extend or modify it to fit their particular needs and requirements, so long as they attribute its original source and do not use it commercially.

To make editing simple, we wrote the Guide in Markdown, a lightweight markup language that is **extremely easy to use**. Markdown lets authors write text once then convert it into multiple file formats. This makes it easier to reuse the same text in many places.

Markdown takes less than 20 minutes to learn most of the syntax needed for routine writing tasks, and the marked-up text remains easy to read. Markdown can be converted to HTML5 so it plays well with most LMSs. It also can be converted directly into MS Word or PDF documents. With some additional free tools, Markdown documents can be compiled into rich online books.

### About This Workshop

**Before the workshop** participants need to install GitHub Desktop and register for a free GitHub account. GitHub is a no/low cost, secure space for collaboration and code sharing that is popular with data scientists and coders. GitHub is becoming a popular spot for hosting blogs, static websites, and open-access book projects to (to see an example, check out this book, [Using Markdown inside R](https://bookdown.org/yihui/rmarkdown/).) The GitHub Desktop application greatly simplifies installation, creating an account, and handling repositories.

Participants also need to install R and R Studio.

In the first half of the workshop participants will use GitHub to copy the repository containing our Resource Guide to a personal GitHub account. Then they will create and edit new Markdown documents using R Studio as a plain text editor. Finally they will push their new files to their personal repository. In the second half of the workshop, participants will use R Studio and standalone tools to render edited pages into HTML and MS Word formats. I will demonstrate how to use R Studio plus Bookdown to turn a collection of Markdown files into a book that can be hosted online or printed.

Participants will leave this workshop with a complete copy of the Science Writing Guide that they can revise to match their local needs, and a personal GitHub account from which to share work with others, or launch a book project (say, a new lab manual) of their own.

# Pre-Lab Assignment: Install the Required Software

Give yourself enough time to install the software. GitHub Desktop takes 10-15 minutes to install. Registering for a new GitHub account takes 5-10 minutes. R Studio Desktop takes 10-15 minutes to install if you do not have it already. If something does not work, we will fix it at the beginning of the workshop.

## Install GitHub Desktop

Go to GitHub and follow their instructions for installing GitHub Desktop on your computer: [Downloading and Installing GitHub Desktop](https://desktop.github.com/)

If you have time, take a look around the application: [Getting Started With GitHub Desktop](https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop)

## Register for a Free GitHub Account

GitHub Desktop should ask if you want to log into an existing account, or create a new one. If it does not do that automatically, go to the [GitHub home page](https://github.com/) and follow the prompts to register. **You do not need a private account for this workshop**. If you choose to start using GitHub regularly, it only costs $50/year for a private account. GitHub definitely is a good investment; the backup service alone saved me several times when I accidentally deleting important files.

## Install R and R Studio Open Source Edition

You need to download and install R **before** R Studio Desktop on your computer.

1. [Download and install R](https://cloud.r-project.org/).
   * If you are a Windows user: Click on “Download R for Windows”, then click on “base”, then click on the Download link.
   * If you are macOS user: Click on “Download R for (Mac) OS X”, then under “Latest release:” click on R-X.X.X.pkg, where R-X.X.X is the version number. For example, the latest version of R as of November 25, 2019 was R-3.6.1.
   * If you are a Linux user: Click on “Download R for Linux” and choose your distribution for more information on installing R for your setup.
2. [Download and install R Studio](https://www.rstudio.com/products/rstudio/download/).
   * Scroll down to “Installers for Supported Platforms” near the bottom of the page.
   * Click on the download link corresponding to your computer’s operating system.

For more detailed instructions, consult [Installing R and R Studio, Step by Step](https://moderndive.netlify.app/1-getting-started.html)

# Exercise 1: Copying the Resource Guide Repository From GitHub

## Background

GitHub stores collections of files related to one project as folders called **repositories**. When you installed GitHub, you created a dedicated spcace inside your computer for storing repositories. This is where your own working copy of the Writing Resource Guide will live.

Users can create multiple repositories for different projects. Private repositories can only be seen by specific GitHub users that the project owner allows. Public repositories are available for anyone to view or copy, but they cannot add new files, or revise or remove existing files without the repository owner’s express permission.

What makes GitHub so useful is that you can make a copy of any public repository (plus any private ones you are given rights to see) and store it in your own GitHub account. Your copy contains all of the files that were in the project’s original repository.

There are two kinds of copies: **clones**, and **forks**. A clone is a completely independent copy. GitHub will tell you if there have been changes made to the original source repository, but those changes are not added to your copy, and any changes you make to your version do not affect the source. We usually clone repositories when we want to use the materials independently from their original authors.

A fork is a collaborative copy that remains connected to its parent. All of the users are working on the same set of files at the same time. Unlike a Google document though, changes made by collaborators have to be approved and merged by the owner of the repository. Once changes are approved, they are “pushed” to all collaborators’ copies of the repository. Forked repositories are useful when multiple authors are working on one project. Every change is documented along with who made it. Individuals can work on separate files (and even different versions of the same file) without interfering with each other. Earlier versions of every file are available from the file archive.

In this exercise you will be cloning the **ABLE 2022 Workshop** repository. This is your sandbox copy; you cannot damage the original source files. You can edit any page or add new pages. If you make a mistake in a document and cannot fix it, just download a new copy. Later you will fork the SWP Writing Resource Guide. You’ll also be using the **Commit** and **Push** commands. Do not worry about what they mean right now; we are coming back and looking at them in detail in Exercise 3.

## Procedure

1. Use a web browser to log into your GitHub account.
2. [Click here to go directly to the ABLE workshop repository](https://github.com/adanieljohnson/ABLE_2022_Workshop). Alternatively, go to the Search window at the top left, and enter **adanieljohnson/ABLE\_2022\_Workshop**. This is the shortcut to the project repository.
3. What you see will be something similar to the screenshot below. Click on “Code” to open the window with the link to clone this repository.

Main page of ABLE Workshop repository

1. Copy the URL. You will need it in the next steps.

Orange arrow: URL to clone the repository.

1. Open GitHub Desktop. Go to **File > Clone Repository.**

GitHub Desktop

1. Paste the URL you just copied into the window. Do not change the local path right now.

1. You should now have a copy of the files on your local hard drive. Use “View in Finder” to double-check.
2. Before continuing, you want to save your new repository and files to your GitHub account. On the main page of GitHub Desktop, you should see a list of files that have changed since they were last saved to your account. Give your update a SHORT name (yes, “Update” is fine.) Then click on **Commit** to formally save the changes locally.

Committing changes to your local copy of the files.

1. To upload a permanent copy in your GitHub account, click on **Push to origin**.

Pushing a copy of your changes to your main GitHub account

1. Now that your repo is saved, we can play. Open the folder named **Writing Resource Guide.** Inside are files with the “.md” extension. These are Markdown files for the full Resource Guide.
2. Open the folder named **Sample Files**. It contains examples of .md files that have been rendered into HTML and MS Word formats. Pick one of each type, open them, then open the corresponding file with the .md extension.
3. Using **Handout for Exercise 1** as your “.md to .html/.docx dictionary,” try to translate the markup in the .md files.
   * How hard or easy is it to read the **original text** in the .md file?
   * How does the markup translate into a web page?
   * How does the marked up text translate into a Word document?
4. If you make any changes to your .md files, be sure to commit them before going on to Exercise 2.

# Handout for Exercise 1: Introduction to Markdown

## Markdown Syntax

Markdown was created as a way for writers to mark up text quickly to show formatting without having to embed full HTML tags. The goal with Markdown is to separate the structure of a document (headers, paragraphs, bullets, etc.) from the words of the text. By using a single set of marking conventions (called the **syntax**) marked-up text can be converted to dozens of different formats.

Here is some text formatted so you can see the the markup:

### Level 4 Header - Formatting  
  
There are two main kinds of text formatting:  
\* Inline formatting (\\_italics\\_, subscripts like H\~2\~SO\~4\~, etc.)  
\* Block- or paragraph-level formatting (headings, sub-headings, etc.)

…and what the same text looks like rendered:

#### Level 4 Header - Formatting

There are two main kinds of text formatting:

* Inline formatting (*italics*, subscripts like H2SO4, etc.)
* Block- or paragraph-level formatting (headings, sub-headings, lists, etc.)

## Markdown Codes for Inline Formatting

The table shows how to mark text, and what it will look like when rendered.

| Inline Formats | How to Mark Them | How They Appear |
| --- | --- | --- |
| Italics | \_italicized\_ word | *italicized* word |
|  | \*italicized\* word | *italicized* word |
| Bold | \_\_bolded\_\_ word | **bolded** word |
|  | \*\*bolded\*\* word | **bolded** word |
| Bold Italics | \_\_\_marked\_\_\_ word | ***marked*** word |
|  | \*\*\*marked\*\*\* word | ***marked*** word |
| Superscripts | Super^script^ed letters | Superscripted letters |
| Subscripts | Sub~script~ letters | Subscripted letters |
| Horizontal rule | \*\*\* | Draws a line across page \_\_\_\_\_ |
| Inline code (not rendered) | `code block` | code block |
| Escape a special character | \\*code block\\* | \*code block\* |
| Links to web pages | [text](link) | [RStudio](https://www.rstudio.com) |
| Links with URL | [link](link) | <https://www.rstudio.com> |
| Links to embed local images | ![alt text or image title](path/to/local\_image). | ![Image on my desktop](/home/desktop/image\_on\_desktop.png) |
| Links to embed image from web | ![alt text or image title](https://upload.wikimedia.org/ wikipedia/commons/thumb/6/6c/ Biology\_students\_in\_lab. jpg/320px-Biology\_students\_in\_lab.jpg). | Image on Wikimedia Commons |

## Codes for Block Level Formatting

### Headers

Header levels are indicated with 1-6 hash marks followed by a space.

| Block Formats | How to Mark Them |
| --- | --- |
| Level 1 Header | # Level 1 Header |
| Level 2 Header | ## Level 2 Header |
| Level 3 Header | ### Level 3 Header |
| Level 4 Header | #### Level 4 Header |
| Level 5 Header | ##### Level 5 Header |
| Level 6 Header | ###### Level 6 Header |

How they render:

# Level 1 Header

## Level 2 Header

### Level 3 Header

#### Level 4 Header

##### Level 5 Header

###### Level 6 Header

### Paragraphs

Regular paragraphs need to be separated by a blank line, or they will run together. For example, a text written without spacing lines like this:

There are several kinds of text formatting to explore.  
 Inline formatting text.  
 Block- or paragraph-level formatting.  
 Lists and quotes.

Looks like this when rendered:

There are several kinds of text formatting to explore. Inline formatting text. Block- or paragraph-level formatting. Lists and quotes.

Text written WITH spacing lines like this:

There are several kinds of text formatting to explore.  
  
 Inline formatting text.  
  
 Block- or paragraph-level formatting.  
  
 Lists and quotes.

Renders like this:

There are several kinds of text formatting to explore.

Inline formatting text.

Block- or paragraph-level formatting.

Lists and quotes.

Blockquotes are written after the “>” symbol. This text:

> "I thoroughly disapprove of duels. If a man should challenge me,  
 I would take him kindly and forgivingly by the hand and lead him  
 to a quiet place and kill him."  
>  
> --- Mark Twain

Renders like this:

“I thoroughly disapprove of duels. If a man should challenge me, I would take him kindly and forgivingly by the hand and lead him to a quiet place and kill him.”

— Mark Twain

Blocks for writing code that is not converted are written between three backticks. To include literal backticks, use more backticks outside.

``` Block of text or code that should NOT be rendered, like this \_\_bold\_\_ word. ```

### Hidden Comments

There are two ways to add comments to Markdown that will not display in the rendered text:

[//]: # (This comment in Markdown gets removed from both .html, and .docx)  
  
<!--- This comment in Markdown is retained in .html, gets removed from .docx-->

## Lists

Bulleted or unordered lists need to be separated from the preceding paragraph by a blank line. The items start with \*, +, or -, and you can nest one list within another list by indenting the sub-list by four spaces. For consistency, it is best to make it a habit of using one character for main bullets, and the other two for sub-bullets.

A coded list looks like:

\* First item  
\* Second item  
 + First sub-item  
 - First sub-sub-item  
 - Second sub-sub-item  
 + Second sub-item  
\* Third item

The output renders as:

* First item
* Second item
  + First sub-item
    - First sub-sub-item
    - Second sub-sub-item
  + Second sub-item
* Third item

Ordered list items start with numbers, but the rule for nesting are the same as with unordered lists.

1. First item  
2. Second item  
 1. First sub-item  
 1. First sub-sub-item  
 2. Second sub-sub-item  
 2. Second sub-item  
3. Third item

The output renders as:

1. First item
2. Second item
   1. First sub-item
      1. First sub-sub-item
      2. Second sub-sub-item
   2. Second sub-item
3. Third item

The two formats can be mixed together:

1. First item  
 + Sub-item  
 + Sub-item  
2. Second item  
3. Third item

Renders as:

1. First item
   * Sub-item
   * Sub-item
2. Second item
3. Third item

The challenge when using numbered lists in Markdown is that they sometimes default to starting with #1. There is no simple and consistent way to force a list to start with a particular number. I usually fix this by editing the HTML or Word document directly, or just pre-number lists myself.

### Task Lists

Task lists are unordered lists with editable checkboxes instead of bullet marks. To create a task list with open check boxes, begin each item with a dash, space, then opening and closing square brackets separated by one space (- [ ] ) in front of task list items. To select a checkbox, add an x in between the brackets (- [x] ).

How to enter it:

- [x] Write the manual pages  
- [ ] Update the LMS  
- [ ] Email students

The rendered output looks like this:

* ☒ Write the manual pages
* ☐ Update the LMS
* ☐ Email students

## Math Expressions

Inline LaTeX equations are written using standard LaTeX syntax, and enclosed by pairs of $ signs. This code:

$f(k) = {n \choose k} p^{k} (1-p)^{n-k}$

Renders an inline version of the equation as:

Equations in display style are written with a pair of double dollar signs. This code:

$$f(k) = {n \choose k} p^{k} (1-p)^{n-k}$$

Renders as:

The first link below is for a tutorial on writing LaTeX equations. The second link is for a web page where you can write LaTeX equations using an online visual editor.

* [Writing LaTeX equations](https://www.overleaf.com/learn/latex/Mathematical_expressions)
* [Online equation builder](https://latex.codecogs.com/legacy/eqneditor/editor.php)

MS Word handles equation rendering well, but other Markdown formatting tools sometimes fail to render correctly. Always check any formatted equations you have in a .md file the first time you try out a new rendering tool.

## Handy HTML Bits

There are some things that Markdown does not handle well. For example, subscript and superscript markup may not work on the first character or number in a word (like 3H) or for a whole word. Some other handy items are not in the GHFM syntax at all. A few basic HTML codes can fill in these gaps.

| If you need to… | Insert this HTML snippet | For this result |
| --- | --- | --- |
| Force a superscript for a whole word or first character | <sup>super</sup>script | superscript |
| Force a subscript for a whole word or first character | <sub>sub</sub>script | subscript |
| Strike through text | <strike>This</strike> word | This word |
| Add an extra space between items | &nbsp; |  |
| Add an extra line between items | <br> |  |
| Add a horizontal rule between lines | <hr/> | \_\_\_\_\_ |
| Add Greek letters | &alpha;, &eta; | α, η |

This page is a [good source for other HTML shortcuts](https://www.w3schools.com/html/html_formatting.asp). You csan find more [information about special symbols here](https://www.w3schools.com/html/html_symbols.asp).

## Learning More

These web resources can help you build your Markdown writing skills. More are listed in the Notes for Instructors.

* [GitHub’s Introduction to GHFM](https://docs.github.com/en/github/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax)
* [Making Tables in Markdown](https://www.pluralsight.com/guides/working-tables-github-markdown)
* [Web-based Table Maker](https://www.tablesgenerator.com/markdown_tables)
* If you want to take a **very** deep dive, this is a link to the full [Official Specification for GHFM](https://github.github.com/gfm/)

# Exercise 2: Creating and Editing Markdown Pages

## Background

Markdown was designed to be easy to add while writing. Unlike HTML, Markdown does not require any special code or starting text at the beginning of a new document. The only requirement is that the saved text file should have **.md** as the extension.

There are many ways to write Markdown files. You can:

* Write texts entirely inside R Studio;
* Use a standalone plain text editor on your computer;
* Write text files inside GitHub via a browser.

Which will work best for you depends on your personal preferences, and how you plan to use the files. You may end up using some of all three methods. The goal is to find a workflow that fits how you use documents most often.

R Studio happens to be a very good plain text editor. If you plan to manage your files entirely through R Studio, it makes sense to write your text files there too. If you need to share your documents in many different places or formats, or are not sure where they will go, writing them in a plain text editor is probably easiest.Both of these workflows require you to commit and push changes to GitHub from time to time.

If you want to be 100% sure you never lose a single word, and don’t mind a few extra steps, writing directy in GitHub might be best. Files written in GitHub via a browser are saved immediately to their repository: no pulling or pushing needed.

Personal Tidbit: most of my writing currently ends up on our LMS so it needs to be plain HTML. I write occasional handouts that need to be in MS Word (.docx) format. I prefer to write drafts in a commercial plain text editor that shows Markdown codes in color. I work across multiple computers, so even my early draft files go into a GitHub repository. That way I can access them anywhere.

When it is time to generate output, the commercial text editor I use can convert Markdown directly to plain HTML that I then can copy/paste straight into our LMS. To generate MS Word documents and online books I use R Studio, which plays well with GitHub repositories. I rarely make PDFs anymore, because they can be hard for screen readers to translate.

In this exercise you will try writing and editing Markdown texts using R Studio as a plain text editor, and a standalone text editor.

## Procedure:

1. Go to the file containing the Table of Contents page for the Writing Resource Guide. Find 1-2 pages you think you might want to edit or contribute to. Also find 1-2 topic that you do not see in the current pages that you think should be in the Guide.

### Editing in R Studio

1. Open R Studio, and choose\_\_File > Open\_\_ and navigate to the **ABLE 2022 Workshop** repository.
2. Locate one of the pre-existing .md files you wanted to edit, and open it.
   * First look at the current structure. If needed, compare it to the markup guide in the handout from earlier. Do you see how the structure of the rendered file is encoded in the Markdown tags? (Remember, you can click the **Preview** tab at the top of the window to switch from Code to Preview modes.)
   * Look for a section you want to edit. Make some changes, save, then see how your changes look using **Preview** mode.
3. Next you will start a new file. Choose **File > New File > Markdown file**.

Start a new .md file

1. Enter some starting text. One or two words is enough.
2. Save the file in your **ABLE 2022 Workshop** repository. Give your file a name that reflects the topic you did not see. Include the .md extension. **TIP:** when you name files, do not include spaces. Write all file names in CamelCase (FileName1.md) or Snake\_Case (File\_Name\_1.md)
3. Now you have a starting file that you can edit any way you wish.
   * If you want to see how changes to the Markdown code affect it, click the **Preview** button at the top of the window.
   * If the .md file will not display, you probably have an error in the format. Look at the bottom of the page for a tab called **R Markdown**. There usually is an error message telling you what went wrong.
4. When you are finished editing your two pages, save them both in the **ABLE 2022 Workshop** respository folder.

### Editing Markdown With a Plain Text Editor

1. Close R Studio and save any open files.
2. Navigate to the your **ABLE 2022 Workshop** repository folder.
3. Click on one of the .md files to select it. This time you want to open the file in a plain text editor.
   * For Windows, the default installed text editor is **Notepad++**.
   * For MacOS, **TextEdit** is the default installed editor.
   * If you have no idea what text editor you have installed, just double-click the file. Your operating system will try to open the file with the appropriate program.
   * You can open and edit .md files with any text editor you have installed ([Atom](https://atom.io/), [Brackets](https://brackets.io/), [Bluefish](https://bluefish.openoffice.nl/index.html), [BBEdit](https://www.barebones.com/products/bbedit/), etc.)
4. After you have edited an existing .md file, try creating a new one from scratch. Once again, create a new file for a topic you think is missing from the Table of Contents of the Resource Guide.
5. When you have finished, save both of your edited .md files.

### OPTIONAL: Writing Directly in GitHub

The workflow for creating and editing .md files in a browser is a bit different. These instructions only work when you are inside GitHub online.

1. Using a browser, log into GitHub (not Desktop). Go to the repository containing your cloned copy of the Resource Guide. At the top of the list, click on **Add file**. Choose **Create new file**.
2. Give your file a name that reflects the topic you did not see. Include the .md extenstion.
3. Enter some starting text. One or two words is enough.
4. At the bottom of the page is the **Commit** dialog. GitHub requires you to describe what you did when you save a file. That information goes in the first box, and is required. It might seem like extra trouble at first, but when you are making changes in critical files, these commit comments can help you track what changes you made and when. If there are multiple authors working on a file, each author’s changes should say specifically what they worked on.
5. Commit the file directly to the master (branches and pull requests are not important until you are working on more complex projects.) **TIP:** when you name files, do not include spaces. Write all file names in CamelCase (FileName1.md) or Snake\_Case (File\_Name\_1.md)
6. Now you have a starting file that you can edit any way you wish. If you want to see how changes to the Markdown code affect it, simply click the **Preview** tab at the top of the window.
7. Next, go back to the Table of Contents and find a page you think you might want to edit. Locate the corresponding pre-existing .md file, and open it.
   * First look at the current structure. If needed, compare it to the markup guide in Handout 3. Do you see how the structure of the rendered file is encoded in the Markdown tags? (Remember, click the **Preview** tab at the top of the window to switch from Code to Preview modes.)
   * Look for a section you want to edit. Make some changes, commit the file to your local repository, then see how your changes look using **Preview** mode.

### Tips and Tricks

* Always start Markdown pages with a Level 1 header that is the title of the document. I reserve the Level 1 header just for that purpose.
* It is tempting to remove spaces between lines in .md files. BAD idea! Markdown relies on blank lines to keep track of blocks of text. When Markdown is rendered, extra blank lines are deleted automatically.
* When editing inside GitHub, pages may extend beyond the screen. If so, look at the upper right of the editor window for a drop-down that says **No wrap**. Click it and choose **Soft wrap**. This keeps sentences and paragraphs from running off the right side of the screen.

# Exercise 3: Updating Your Local and Master Copies of the Resource Guide

## Background

You have edited and saved your .md files locally. Now the local repository on your computer is ahead of your GitHub account! We need to synchronize the two sets of files.

GitHub has two functions that help you keep local and online repositories in sync: **pull** and **push**. These commands can be confusing, but they are at the heart of what makes GitHub so powerful.

When you pull a copy of a repository, GitHub compares the versions of the files in the repository on its servers and your local computer. If the server version does not match the local version exactly, GitHub will try to download the newer version and update the older version on your local computer. If there are newer versions in BOTH locations, GitHub will ask you to pick one to be the new “correct” version.

Say you have been editing several .md files using the text editor on your computer. When you commit the files, you are telling GitHub “these are the newest versions, and what everyone should be using from now on.” GitHub dutifully stamps them with a time and location code, and stores them locally. Now you need to push the files, that is, tell the main GitHub server to update its copies.

When you push files, the server compare its copy of each file to the data stamp on the one you pushed. If there are changes in the server copy that are not in yours, GitHub will stop and tell you that you need to pull the changes already logged by the server and merge them with your local changes first. **Then** you can push the newest version of each file back to the server.

Why does GitHub use so many different commands? Why not just save a file instantly and everywhere, like Google does? If you have ever had someone change a Google document you just finished editing back to what you deleted, you know why. GitHub’s strategy is designed for code writers and editors. There has to be one correct version, and one person with the authority to make that decision. Committing a file only saves it to the local copy of the repository you happen to be working in. Pushing a file changes it on the main server, and on all of the forked copies.

In this exercise we’ll be pulling and pushing files between your server and local repository copies.

## Procedure

1. Open GitHub Desktop. If you are not logged in to your account, do so now.
2. In the upper left of the work pane, click on Current Repository. Pick the **ABLE 2022 Workshop**. You now should see a list of files that changed. The file you created and the existed file you edited should be listed.

Setting the repository to synchronize.

1. In the upper part of the work pane, click on **Fetch Origin**. This does the same thing as issuing a Pull command; it pulls information about the server version of the repository, so it can compare the two versions.
2. You should see a list of files that have changed since they were last saved to your account. Give your update a SHORT name. Then click on **Commit** to save the changes locally.

Committing changes to your local copy of the files.

1. To update the copy in your GitHub account, click on **Push to origin**.

Pushing a copy of your changes to your main GitHub account

1. If you click on Fetch Origin again, you should get a message saying **All files up to date**. This means your files in the local copy and the server copy of the repository all match.

### Tips and Good Practices

**How often should you commit and push?** It is up to you really. Generally you should commit every 30-60 minutes, or after you complete a large set of edits. You may only push changes to GitHub once or twice a day, though more often does not hurt anything. It is a good idea to push to GitHub when you finish working for the day.

**Why commit regularly?** GitHub creates a record of every Commit and Push command in your account that includes a detailed list of which files were changed and how. If you are editing a file and accidentally delete some important text, you can look back in that file’s history to reconstruct the missing pieces. The more often you Commit, the finer grained the record of changes.

# Exercise 4: Generating Your Output Documents

## Background

Markdown is converted to other document formats using [Pandoc](https://pandoc.org/index.html), the “universal document converter.” It can read many different document file types, then convert one format to another. Some file types are fussy, but Markdown and Pandoc play very nicely together.

There are several ways to access Pandoc. We’ll explore two: using a standalone web-based converter (for clean HTML), and R Studio (for MS Word). I will demonstrate a third route, using the command line.

## Converting Text.md to Text.html

Most plain text editors (except for R Studio) can convert Markdown to very compact HTML. If your text editor does not, this web-based Pandoc converter works well.

1. Go to the **MD to HTML Converter** at [https://ashkanph.github.io/md-to-html/.](https://ashkanph.github.io/md-to-html/)
2. In the upper right corner, use **Browse** to find your local GitHub folder, and choose one of your recently edited .md files.
3. The .md file will be converted to .html and displayed.
4. Use the **Save as HTML** button to write a copy of the file back to your local GitHub repository.
5. If you open the version of the file with the .html extension in your plain text editor, you will see it has been rewritten using HTML markup. You can copy and paste the HTML code into most LMS pages. Alternatively, you can share the file itself with others; they can open it in any web browser.

**Why not use R Studio for HTML?** When R Studio generates a standalone HTML preview, it embeds the raw code for every image plus a LOT of extra styling code. For example, the HTML file for the handouts for this workshop that was generated using this converter was 63 Kb. The same HTML file generated using R Studio was 6 Mb. This problem goes away when R Studio creates books; the HTML files are very compact, because they reference separate styling and image files rather than embedding them.

## Converting Text.md to Text.docx

R Studio converts .md files to well-formatted Word documents.

1. Open R Studio, and go to the workshop files. Open one of the files you edited or created.
2. At the top of the screen, open the pulldown menu next to **Preview** and choose **Word document**.

Converting Markdown to Word format

1. R Studio will process the file through Pandoc and generate a pre-formatted MS Word .docx file, then save it to the same directory where the .md file is located.
2. Click on the .docx file to open it in MS Word.

MS Word files are formatted using a standard template. If you want your documents to be formatted a different way, you will need to create a template file. You can then import the template styles into your newly generated documents to make them look however you like.

## Demonstration 1: Using Pandoc Via the Command Line

It is very easy to install and run Pandoc from your computer’s command line. If you would like to try it yourself:

1. Go to the [Pandoc installation page](https://pandoc.org/installing.html) for instructions to download and install to your computer.
2. Use the following commands to convert from .md to .html or .docx format. The commands are the same, except for the extension on the output file.

pandoc -s /filepath/Input\_Filename.md -o /filepath/Output\_Filename.html

pandoc -s /filepath/Input\_Filename.md -o /filepath/Output\_Filename.docx

Pandoc is fussy about rendering PDFs. First, it does not tolerate markup errors, and may refuse to convert a file. Second, you must install a pdf-engine then call it in the command. Look at the example below:

pandoc -s --pdf-engine=xelatex /filepath/Input\_Filename.md -o /filepath/Output\_Filename.pdf

The pdf-engine defines what the format will look like. This particular PDF engine’s default layout and fonts look outdated and can be hard to read. Changing them requires some deeper coding work, which is why I do not recommend using Pandoc for creating occasional PDFs. It is quicker to render a file in MS Word then save it as a PDF.

# Demonstration 2: Using Bookdown to Compile Documents Into an Online Book

R has several powerful libraries for creating online materials. We used **bookdown** (which leverages **knitr** and Pandoc), which was created specifically for authoring full length technical books. The [Home page for the R Bookdown package](https://bookdown.org/) has examples of books that were created this way. Two particularly useful books you might want to look at are:

* [Guide to using Bookdown](https://bookdown.org/yihui/bookdown/)
* [Using Markdown inside R](https://bookdown.org/yihui/rmarkdown/)

There is not enough time for you to set up your own GitHub-hosted book. Instead, we’ll take a short tour through the back end of the **SWP Writing Resource Guide** and see how R Studio can convert a collection of separate .md files into an integrated book.

# Links to Participant Resources

### Getting Started

* [Workshop Repository and Writing Resource Guide](https://github.com/adanieljohnson/ABLE_2022_Workshop)
* [WHY and HOW to write well-structured, reusable texts](https://www.portent.com/blog/content/content-with-github-markdown.htm)

### GitHub

* [Install GitHub Desktop](https://desktop.github.com/)
* [GitHub home page](https://github.com/)
* [Getting Started With GitHub Desktop](https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop)
* GitHub’s [Atom Text Editor](https://www.portent.com/blog/content/atom-markdown.htm) can be set up as a Markdown-native editor.

### Markdown-like add-ons

* [Mermaid](https://mermaid-js.github.io/mermaid/#/) is a Javascript based diagramming and charting tool. It uses Markdown-inspired text definitions to create and modify diagrams dynamically.

### Markdown Syntax

* [GitHub’s Introduction to GHFM](https://docs.github.com/en/github/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax)
* [Quick Guide to Mastering Markdown](https://guides.github.com/features/mastering-markdown/) \*[Extended Syntax for Markdown](https://www.markdownguide.org/extended-syntax/)
* [Emoji cheat sheet](https://github.com/ikatyang/emoji-cheat-sheet/blob/master/README.md). There are two ways to add emoji to Markdown files: copy and paste an emoji into the Markdown-formatted text, or type emoji shortcodes. GHFM does not support emoji shortcodes, so copy/paste from this cheat sheet is your best option.
* [Making Tables in Markdown](https://www.pluralsight.com/guides/working-tables-github-markdown)
* [Web-based Markdown Table Maker](https://www.tablesgenerator.com/markdown_tables)
* [HTML shortcuts](https://www.w3schools.com/html/html_formatting.asp)
* [HTML special symbols](https://www.w3schools.com/html/html_symbols.asp)
* [Writing LaTeX equations](https://www.overleaf.com/learn/latex/Mathematical_expressions)
* [Online equation builder](https://latex.codecogs.com/legacy/eqneditor/editor.php)
* [Official Specification for GHFM](https://github.github.com/gfm/), if you want to see the complete documentation.

### File Format Converters

* HTML5 for the web:
  + [MD to HTML Converter](https://ashkanph.github.io/md-to-html/)
  + [Dillinger](https://dillinger.io/)
  + [Markdown It](https://markdown-it.github.io/)
* MS Word documents:
  + [Pandoc Converter](https://pandoc.org/demos.html)

### R and R Studio; Bookdown Package

* [Install R](https://cloud.r-project.org/).
* [Install RStudio](https://www.rstudio.com/products/rstudio/download/).
* [Installing R and R Studio, Step by Step](https://moderndive.netlify.app/1-getting-started.html)
* [Using Markdown inside R](https://bookdown.org/yihui/rmarkdown/)
* [Home page for the R Bookdown package](https://bookdown.org/)
* [Guide to using Bookdown](https://bookdown.org/yihui/bookdown/)

### Commercial Products

* [Marked 2](https://marked2app.com/) is an inexpensive desktop application for previewing Markdown. Mac only.
* [Markdown Pad](http://markdownpad.com/) is similar to Marked 2, but for Windows.
* [Typora](https://typora.io/) lets you write and view Markdown immediately. In beta, so will be commercial eventually.
* [Caret](https://caret.io/) is a newer cross-platform tool.

# Information About the Presentation

## Equipment Needed

Participants will need to have a laptop that can connect to the internet, and access to a web connection. Tablets are not going to work; GitHub Desktop does not work as an app. It may be helpful to have 2-3 loaner laptops in the event someone only has a tablet.

The workshop can be conducted in a computer lab, but it would be better if participants used their own computers, so they leave with the full set-up needed to continue using the materials at home.

## Notes for the Instructor

I plan to use this section to summarize feedback and answer questions received during the workshops. Topics I expect will be covered here includes:

* Technical issues: potential bottlenecks or challenges in converting files; other formats; proprietary software options.
* Pedagogical issues: the topics that instructors think are needed in the Resource Guide; questions about the writing instructional strategy used.
* Suggestions for improvement from the participants.

## Sample Results

The working template version of the Resource Guide that participants will clone to their own accounts for editing, a copy of the .md file used to create the handouts, and all other files for the workshop will be [available from the workshop’s public GitHub repository](https://github.com/adanieljohnson/ABLE_2022_Workshop).

## Flow of the Workshop

Participants will learn to use Markdown, GitHub and GitHub Desktop, and Pandoc text converter in the context of a collaborative writing project.

**Markdown** is a lightweight but very versatile text markup language that lets authors write well-structured texts once then reuse them multiple ways. It is extremely easy to learn (<20 minutes for 90% of what you will use routinely), and text with Markdown tags remains easy to read. I estimate that >80% of the new materials I write now start out in Markdown.

There are several “flavors” of the original Markdown language standard. I chose the GitHub “flavor” of Markdown (GHFM) for the workshop because:

* GHFM adds functions that other flavors do not have.
* There are multiple tools for converting GHFM into other formats.
* GHFM is nearly identical to the Markdown syntax used by R Studio to write online books.
* GHFM is the lingua franca for GitHub.

**GitHub** is best known as a place where data scientists share code, but it works equally well as a no/low cost, secure project space that operates seamlessly between institutions. GitHub also is becoming a destination for hosting blogs, static websites, and open-access book projects.

Many ABLE members already know **R and R Studio**, but likely think of them more as a platform for statistics and numerical data analysis. When combined with GitHub and Markdown, R Studio becomes a powerful text editor, integrated development environment, and project management space.

Prior to the workshop, participants will register for a free GitHub account and connect GitHub Desktop to their GitHub account. They also will install R and R Studio. These installations are straightforward, but in the event participants have trouble, they can work with a partner while we complete their setup in the first few minutes of the workshop.

During the first 60 minutes of the workshop, participants will copy the text files for the Writing Resource Guide from a sandboxed repository I have set up for this workshop to their personal GitHub accounts. Using both R Studio and a plain text editor, they will edit an existing page, then create a new page for a topic they think is needed. By break time, participants will have completed their writing and saved their modified files back to their personal accounts.

After the break (assuming 30 minutes), participants will spend the next 60 minutes converting their edited Markdown pages into well-structured documents. They will learn to generate files in two formats:

* HTML5, suitable for use on the web, using [Ashkanph](https://ashkanph.github.io/md-to-html/) .md to .html converter.
* MS Word documents, using either R Studio or the standalone [Pandoc Converter](https://pandoc.org/demos.html).

There is not enough time to set up a complete GitHub-hosted book. Instead, I will use the remaining ~30 minutes to demonstrate how R Studio’s *knitr* and *bookdown* R libraries can compile individually edited pages into a GitHub-hosted e-book.

Participants will leave with a complete copy to our Science Writing Resource Guide which they can use in their courses. Their computer will be set up and ready to use GitHub Desktop and R Studio to continue editing their copy of the Guide, and they will have a GitHub account that they can use to build books or share pages with others. They also will have the resources needed to convert their work into MS Word or HTML documents.