CBW's Bookdown Template Documentation

2024-12-04

Contents

Chapter 1

CBW's Bookdown Documentation

Welcome to CBW's documentation for creating a workshop website using Bookdown. Bookdown is an R package that is used to build books, and in our case, the websites hosting CBW's workshops!

You only need to know markdown and whatever coding language you will be using to learn bookdown!

Note, this is the documentation to create a workshop using *bookdown*. If **Jupyter Book** suits you better, see here.

If you don't know which one to use, click here to learn more!

Chapter 2

Getting Started

Bookdown is an open-source R package that helps write books and articles. We will be building our bookdown-based workshop websites using this, specifically the gitbook template. (This is just the name of the specific template style, you will be working in a workshop template that CBW has prepared for you!)

If you're ready to start making a workshop website in bookdown, let's setup your device (PC, laptop)!

First, let's explain installations.

2.1 Installation

Since bookdown is an R package, you will need R. Plus, our ideal IDE (integrated development environment i.e. the platform we will be working in) is RStudio.

1. Download and install R (You need R 3.6.0+ installed for RStudio) here. Follow the instructions for your operating system (Linux/macOS/Windows). [Maybe we should have more installation instructions - a video?]

   Note: You can check if windows was installed properly on *macOS* by running the command `R` in terminal. On *windows*, –must test–.

   Note: We will not be using the R console, instead, we will be using RStudio!

2. Download and install RStudio here. Scroll down to find downloads for non-macOS.

3. Installl the bookdown R package: Open RStudio and in the console (in the bottom left window of RStudio) run the following command: `install.packages("bookdown")`.

We're ready to start working with CBW's bookdown workshop template now!

Chapter 3

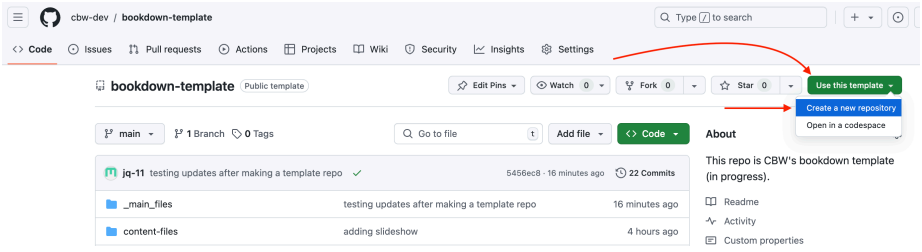
Creating [RC] and Deploying Your New Workshop

Certain aspects of the setup for workshops will be different depending on your role. Headers ending in "[RC]" are for Regional Coordinators. Headers without "[RC]" are assumed to be relevant to both RC and workshop teams.

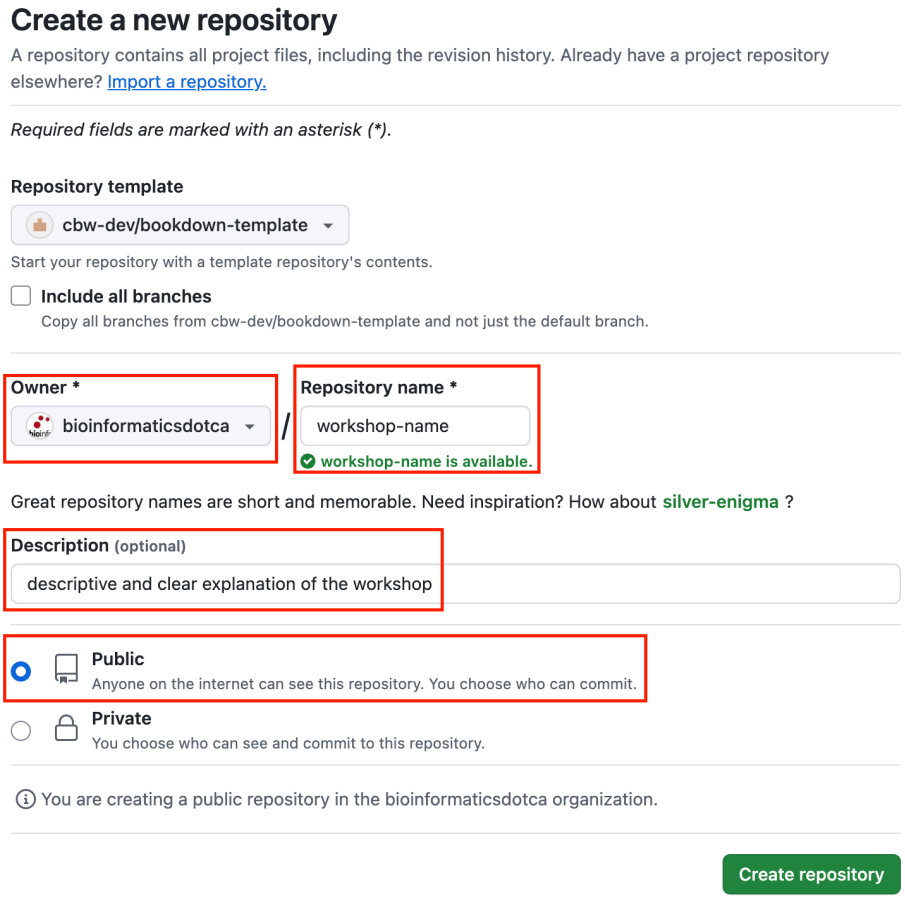
3.1 Workshop Setup [RC]

Regional Coordinators will be tasked with creating and slightly editing each new repository for each new workshop.

1. First, let's go to the bookdown template.
2. Click on the "Use this template" green button, which is to the left of the title of the repository "bookdown-template". Then, press the dropdown option: "Create a new repository", as seen below.



3. You will be brought to a "Create a new repository" page. Fill out the blanks as seen below. That is, change the owner to "bioinformatics.ca" [NOTE FOR TESTING PURPOSES: use cbw-dev], make it public, fill in the repository name and description according to CBW Guidelines. "Include all branches" does not need to be selected.



This may take a couple seconds to generate. After it loads, you will be brought to a new repository for the new workshop!

Now, let's turn this into a website - let's deploy!

3.1.1 Workshop Repo VS Workshop Website

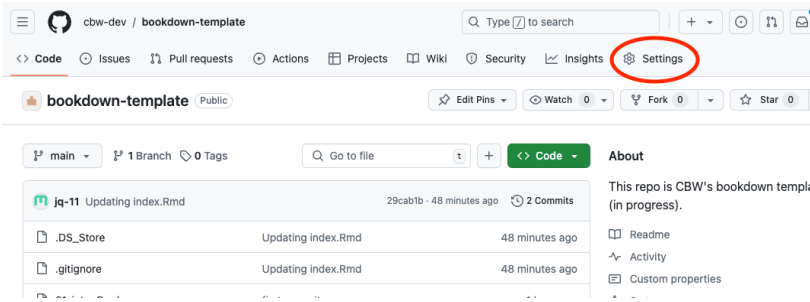
Now, you have made a repository that holds what GitHub needs to make our website (the basic workshop template). Essentially, the template has already been configured so that the html files that make up our website go into a folder

called `docs`. We need to tell GitHub to look at the `docs` folder to find our website files and make it available to see online (a.k.a deploy it).

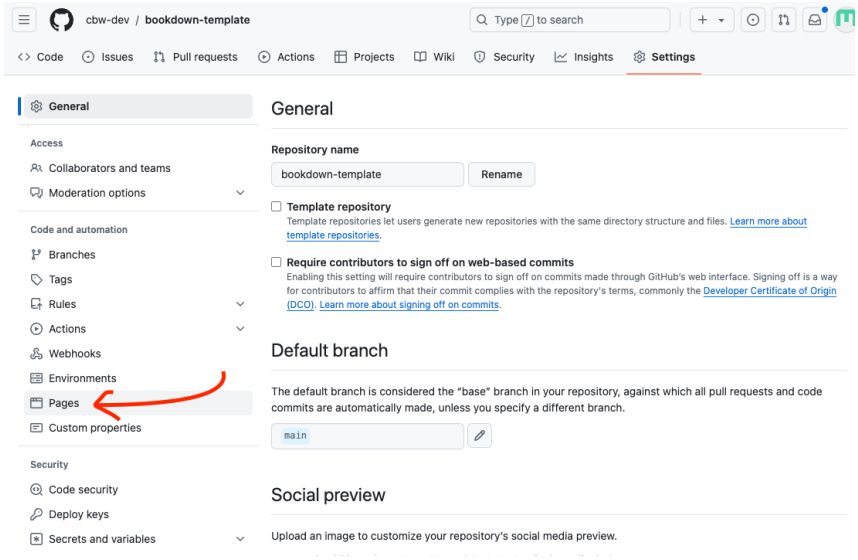
> Distinction:
>
> GitHub (ex. https://github.com/cbw-dev/bookdown-template) holds your repo, which has version control for all your files! The deployed website (ex. https://cbw-dev.github.io/bookdown-template/) has the workshop online.

3.1.2 How to Deploy Your Workshop Website

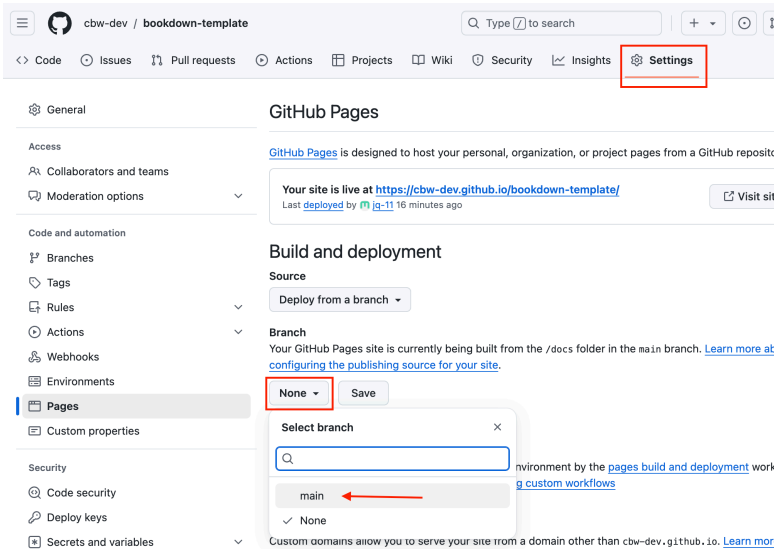
1. In the top navigation bar, select settings.



2. Then, go to the pages sidebar.

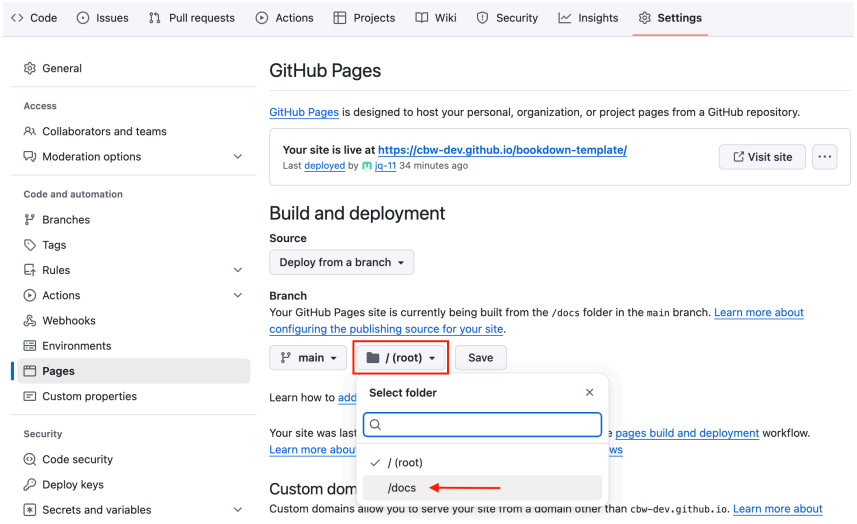


3. "Deploy from a branch" is already selected, which is what we want. We must change the branch from "none" to "main". Select the "None" drop-



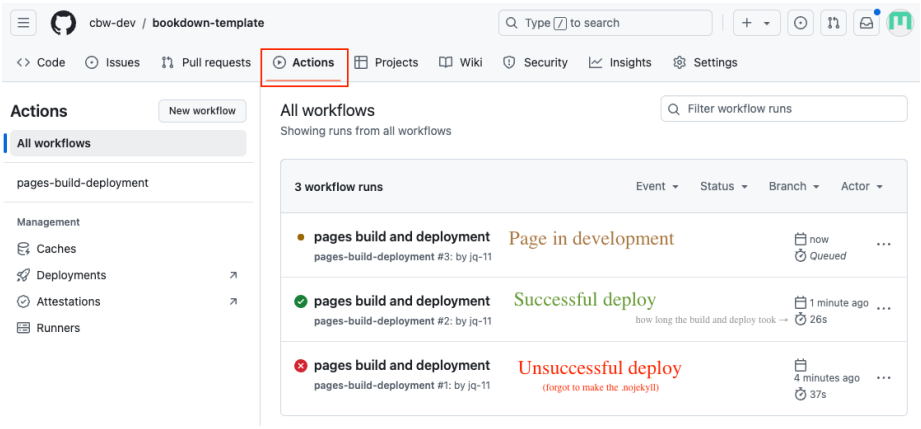
down button and select "main".

4. Then, change the folder from `/ root` to `/docs`. Then press save.



Great! Now we're waiting on the page to build and deploy, which should take less than a minute.

To see updates, go to the **Actions** page (found along the top navigation bar. This will help you understand how the deploy is working, and if it succeeded or failed.



You can click **pages build and deployment** for updates.

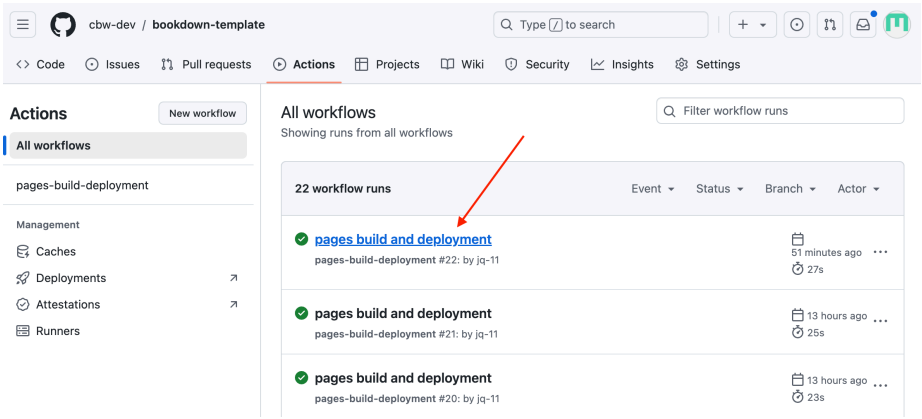
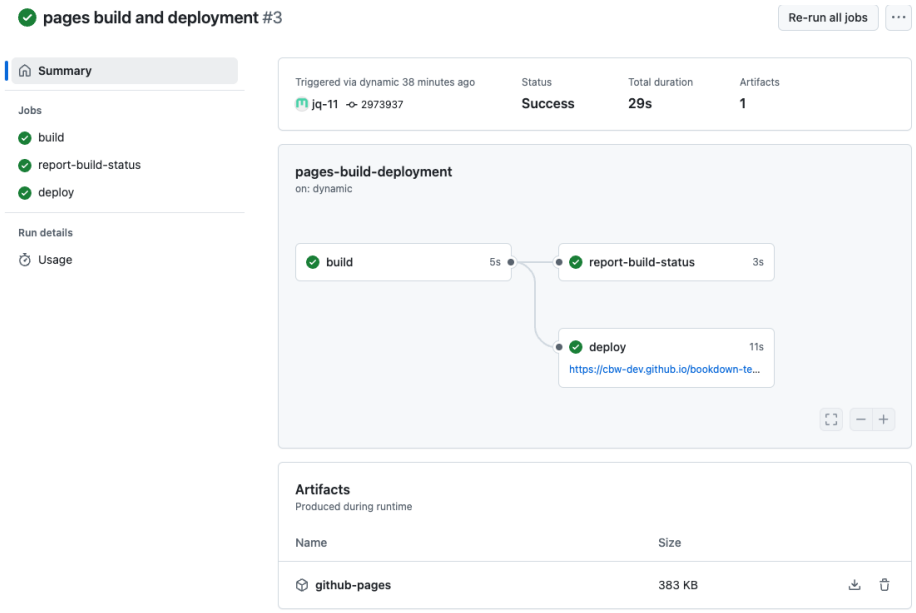


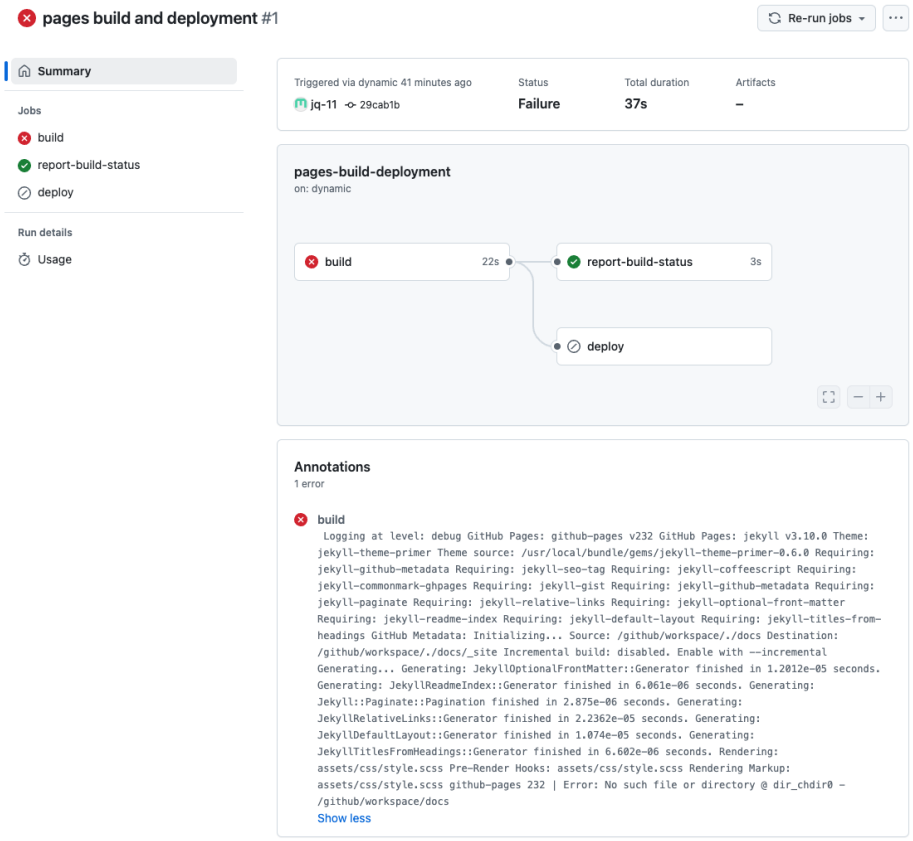
Figure 3.1: where to click for pages build and deployment information

A successful deploy will have a green checkmark next to it. You can inspect the 3 steps: build, report-build-status, deploy. Once it's done deploying, you can find the website at the link provided under "deploy"!

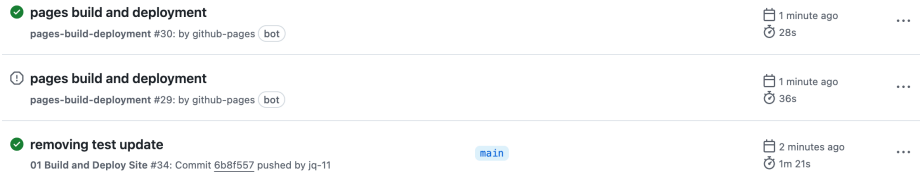


A failed deploy will have a red cross next to it. Clicking through the steps can help you determine what went wrong in the deploy.

Warning: A website can build properly, but may not deploy properly! It is a good idea to check after making big changes.



A Very Specific Build and Deployment Warning



This is a very specific (and unlikely) warning. It occurs when 1 deploy hasn't finished, but another deploy began. THIS IS NOT A CONCERN. This is a warning message you do not have to worry about!

3.2 Setting Up Team Access (Nia will fill this in)

Chapter 4

Command Line, SSH Connection & Git Clone

If you're familiar with the command line and have already established a SSH connection, continue to git cloning your workshop template locally.

4.1 Introduction to the Command Line

This is for those who have no (or extremely little) experience with the command line.

Using the command line, you can use text commands to interact with your computer's operating system. For us, we will be using it to move around our folders and to git clone our workshop into our computer, so we can work on it using RStudio!

> Note:
>
> Do not be worried about using terminal, espeically git commands in terminal! Once we are all setup, we will never have to touch the terminal and write these commands again!

- terminal (mac, unix based) vs command prompt (windows)

4.1.1 Terminal, Command Prompt and Windows PowerShell

We can use the command line using certain tools and applications. Terminal is a Unix-based (meaning Linux and macOS computer already have it) application that allows you to access the command line. Similarly, Command Prompt (CMD) and Windows PowerShell give access to the command line on Windows

computers. However, Terminal, Command Prompt and Windows PowerShell differ in what commands are accepted. The same commands we give to Terminal may not work in Command Prompt and/or Windows PowerShell.

> Note: Windows PowerShell tends to be more advanced than Command Prompt, and often can accept more commands that are accepted by Terminal than Command Prompt.

4.1.2 Common Commands (for us)

We won't need to know that many commands, but for easy navigation and understanding, here is what you (generally) need:

pwd pwd stands for "print working directory". For example, below our output is where in my folders the current .Rmd file that makes up this website is:

```
pwd
```

```
## /Users/jqiu/Documents/CBWgithub/cbw-dev-templates-docs/bookdown-docs
```

ls OR dir "ls" is a **Terminal** command that also works in **Windows PowerShell**. It short for "list" and outputs all the files and folders in the directory (folder) you are currently in. (Note: this code only shows 4 files to save space!)

```
ls
```

```
## 01-getting-started.Rmd
## 02-git-clone.Rmd
## 03-terminal-and-git.Rmd
## 04-files-and-build.Rmd
```

A similar command in **Command Prompt** is `dir`, which also outputs the files and folders in your current directory (along with timestamps)!

cd "cd" stands for "change directory". The command produces no output, but it allows you to go to a different directory than the one you're currently in. For example,

```
pwd # recall: pwd tells us where we currently are
cd img # img is a folder in bookdown-docs
echo "now switching directories" # outputs the following string
pwd
```

```
## /Users/jqiu/Documents/CBWgithub/cbw-dev-templates-docs/bookdown-docs
## now switching directories
## /Users/jqiu/Documents/CBWgithub/cbw-dev-templates-docs/bookdown-docs/img
```

> **Tip**:Typing "cd" and then hitting the `tab` key will give you the available directories you can go to from where you are, or what you

have currently typed in. If there is only one option, hitting `tab` will fill in your command with that option. (This works when typing in any file location into your command line, not only using "cd"). On macOS, the terminal will give you a list if there are multiple options. On Windows, both Command Prompt and Windows Powershell will fill in potential options, and you can hit tab multiple times untill you find your desired file destination.

File Location Shorthands When referring to file addresses, there are helpful shorthands! Here's a summary: - `.` = Current Directory - `..` = Parent Directory - `~` = Home Directory

Here's an example (recall, cd produces no output!):

```
pwd
echo -e # creates a line

echo "Current Directory Example"
cd .
pwd
echo -e

echo "Parent Directory Example"
cd ..
pwd
echo -e

echo "Home Directory Example"
cd ~
pwd
```

```
## /Users/jqiu/Documents/CBWgithub/cbw-dev-templates-docs/bookdown-docs
##
## Current Directory Example
## /Users/jqiu/Documents/CBWgithub/cbw-dev-templates-docs/bookdown-docs
##
## Parent Directory Example
## /Users/jqiu/Documents/CBWgithub/cbw-dev-templates-docs
##
## Home Directory Example
## /Users/jqiu
```

mkdir "mkdir [directory address]" stands for "make directory". Essentially, mkdir will make an empty directory (folder) at a specified location. For example: `mkdir test` would create a folder named "test" in our current directory. The following commands do the same thing. Note: `mkdir ./test` does the same thing.

rmdir "rmdir [directory address]" removes an *empty* directory. For example, `rmdir test` would delete the directory we just made!

rmdir -r OR rmdir /s "rmdir -r [directory address]" (Terminal & Windows PowerShell) and "rmdir /s [directory address]" removes a directory recursively, meaning it deletes all the contents of the folder as we all as the folder in itself. Be careful, you can not restore a directory you removed using "rmdir"!

Up [ ] and Down [ ] Arrows One of the most useful tips for using the command line is to use your up [↑] and down [↓] arrow keys. Using the up [↑] key gives you the previous commands you typed, and the down [↓] arrow returns you to your earlier commands.

4.2 Creating the SSH Connection

We need to create an SSH connection. You have already set this up if you have been git cloning, pulling from and pushing to GitHub. If you have, continue to git cloning. If you haven't, keep reading!

Essentially, we're doing these steps to update and receive updates from our GitHub repository, with security!

Follow the following 3 main steps. Each of these subheaders links to GitHub's official docs, if you would prefer to follow them instead! (Below is the simplified version of the instructions, if you've already been working with GitHub and SSH connection, consider using the official docs.) The official docs may be more up-to-date.

4.2.1 Generating a new SSH key

1. Open Terminal.

   Note: You can do these commands anywhere in your file explorer.

2. Copy and paste this text into your terminal. **Replace the email given below with your GitHub email address** (the email address you used to sign up for Github). Press enter to run the command.

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

You will get this output:

```
> Generating public/private ALGORITHM key pair.
```

Note: Output in your terminal begins with this ">" symbol, as seen above!

You will then be prompted with this message:

```
> Enter a file in which to save the key (/Users/YOU/.ssh/id_ALGORITHM):
```

Press enter. (This uses a default file and default file location.)

If you have already created a SSH key and you are asked to rewrite another key, look at the GitHub Docs for specific steps.

3. Type a secure passphrase (make up a password) when prompted with:

```
> Enter passphrase (empty for no passphrase): [TYPE YOUR PASSPHRASE]
> Enter same passphrase again: [TYPE THE SAME PASSPHRASE]
```

4.2.2 Adding your SSH key to the ssh-agent

1. In terminal, run the following command

```
eval "$(ssh-agent -s)"
```

You will get this output: `> Agent pid 59566`

2. If you're using macOS Sierra 10.12.2 or later additions, you need to modify your `~/.ssh/config` file.

    1. Check if you have a `~/.ssh/config` file: Run the following command:

    ```
    open ~/.ssh/config
    ```

    2. If you get the following ouput:

    ```
    > The file /Users/YOU/.ssh/config does not exist.
    ```

    Create the file using the touch command: run the command given below
    `touch ~/.ssh/config`

    3. Edit your `~/.ssh/config` file using the following instructions. (You can use any text editor you would like, such as vim). Below we use nano as a text editor.
        - Run `nano ~/.ssh/config`
        - Add the following lines to this file. (You should be able to immediately edit.)

        ```
        Host github.com
         AddKeysToAgent yes
         UseKeychain yes
         IdentityFile ~/.ssh/id_ed25519
        ```

        - Exit nano: `ctrl + X`
        - Type "Y" and hit enter to save changes, when asked the following

        ```
        Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
        ```

3. Return to terminal. Run the following command:

```
ssh-add --apple-use-keychain ~/.ssh/id_ed25519
```

4.2.3 Adding a new SSH key to your account

1. Copy the SSH public key: Run the following command to copy the content of the ~/.ssh/id_ed25519.pub file to your clipboard:

```
pbcopy < ~/.ssh/id_ed25519.pub
```

2. Go to your GitHub account on the GitHub website. Click on your profile picture (icon in the upper right). Then, select **Settings**.

3. Under the "Access" section, click **SSH and GPG keys**.

4. Click **New SSH key** or **Add SSH key**.

5. In the "Title" field, add a descriptive label for this key you are creating (ex. if this is your personal laptop, you can call the key: "Personal Laptop").

6. Select the type of key between "authentication" or "signing". (For our purposes, selecting authentication is fine).

7. in the "Key" field, paste (we are pasting what we copied in step 1).

8. Click **Add SSH Key**.

9. If you are prompted, confirm access to your GitHub account.

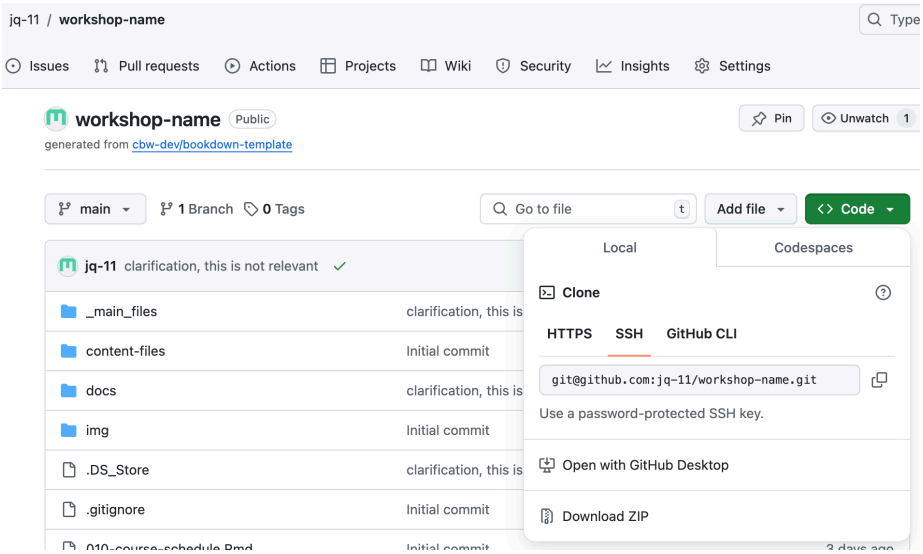
Finally, we're all done! We've created a SSH connection between your device and GitHub!

Thankfully, we only need to do these steps once! Additionally, most security questions are only asked the first time, so when you work on your workshop in the future, you will not have to redo these steps!

4.3 Getting the Template on Your Local Computer - Git Clone!

1. Navigate to where in your local file system you want to have your workshop in (hint: cd + `enter`).

2. Return to your workshop repository on GitHub. Find the ssh for your workshop repository: first, click the green button entitled `< > Code` ands see the drop down options. Click the SSH tab, as seen below, and then copy the text below it. The text should be something like git@github.c om:bioinformaticsdotca/WORKSHOP-NAME.git, as seen below.



Then put this command into your command line, within the folder you want the workshop folder to be in.

For example, if my organization is named: jq-11 and my workshop name is "workshop-name", I would enter this into my terminal.

```
git clone git@github.com:jq-11/workshop-name.git
```

4. You should be ready to go! With your given permissions, you should be able to git push and git pull fine!

   NOTE: Consider having only one team member (or perhaps your RC) make git pushes or control pull requests. To avoid merge conflicts, designate 1 team member to control actual changes to your workshop repo. Other team members can fork or create branches, and create a pull request that the designated team member can check and overlook.

But what do any of these files mean? Which ones do I edit? Which ones shouldn't I edit? How do I open this in RStudio? And how exactly is a page made from all these files??? It's time for you to go to the next page :D

Chapter 5

So What Do These Files Mean?

Ok now we have our workshop locally (on our computer), which is made up of all these files and folders?

Before we dive deep into what to do with these folders, let's explore how bookdown actually works and how to understand RStudio.

5.1 Bookdown Simple Explanation

Here is a general summary of how Bookdown creates html websites from .Rmd files.

Essentially, knitr renders and runs all the code, and the outpus are converted into markdown. After knitr, we eseentially have a bunch of only markdown files.

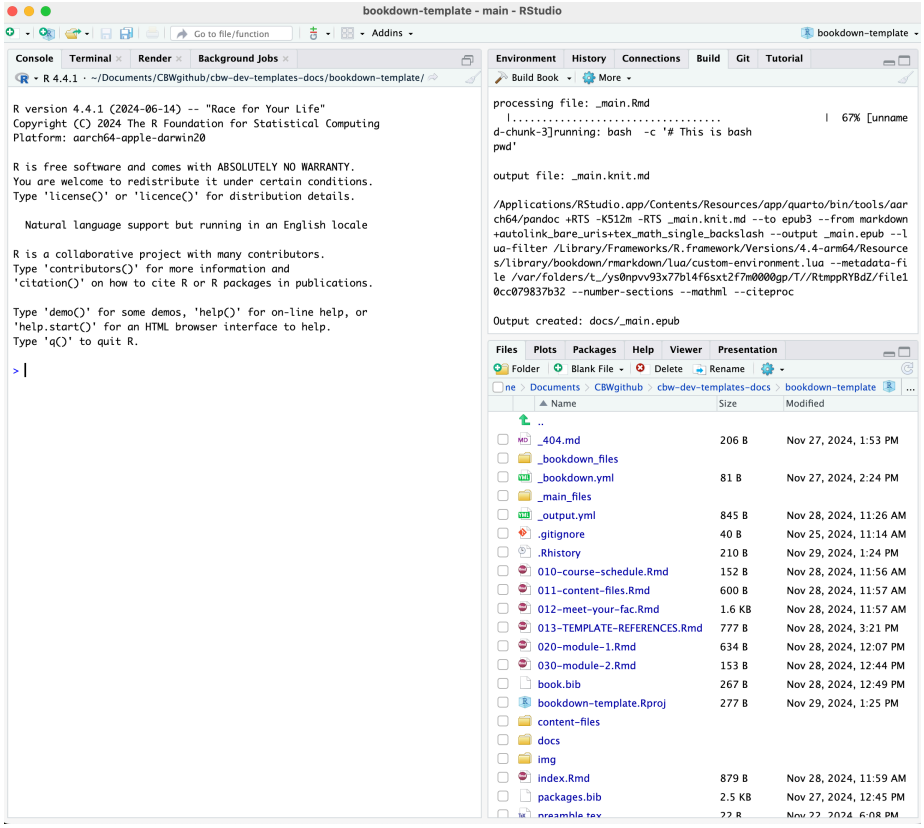
Pandoc translates this markdown into html, so that we get a website! It can be helpful to know when and how these packages work, to help debug later on!

Now, let's figure out RStudio. Skip to file setup if you already know how to use RStudio (and it's built in git control window).

5.2 Open in RStudio

Enter the folder you just git cloned using Findr/File Command, it should be titled "[workshop-name]". Right click on [workshop-name].Rproj and press "Open in RStudio". There is only one file with this file extension. The .Rproj file is what you will open each time you want to work on this workshop! You must explicitly **open the .Rproj** file to build properly!

A RStudio window should open up and look something like the image below.



5.3 Explaining RStudio

In the bottom left, have our console and other debug related windows (such as terminal!). Any code we run will appear in the console. We can access the terminal (just like editing in the Terminal app) under the "Terminal" tab.

In the bottom right, we have all of our files and subfolders. These files will be explained below. This window also contains helpful views, like "Viewer" and "Plots". We will touch on these later.

Try opening `index.Rmd`: a new pane will open in the top left that shows the contents of `index.Rmd`. This is where we will be editing our files! Notice, the "Knit" button.

In your top right, we have a different window with more different views. The most relevant windows to us are the "Build" and the "Git" windows.

No "Git" Window?

Try closing (and maybe even restarting RStudio) and then reopening it. A "Git" tab should appear to the right of the "Build" tab and to the left of the "Tutorial" tab.

5.4 Build the book

Try pressing "Build Book" within the "Build" window. Your build window is going to fill up with text, and soon, a website is going to pop-up as your new window. This is the website you will be editing to create your workshop!

By building the book, all of these files were compiled and converted to .html files, that create a website. Each time we make local changes to our files and we want them to appear in our website, we need to rebuild the book. Note that each time we build our book, the files we edited will be saved first (we don't have to save before building!).

5.4.1 Other Ways to Build Your Book

1. Build the book from the R console:

```
bookdown::render_book()
```

2. Press the keyboard buttons: `cmd + shift + B` (macOS) OR `ctrl + shift + B` (windows)

5.4.2 Knit Your Book

Building can take a long time. If you are editing just one file, you can press the "Knit" button that is at the top of the window with your file. This will run the code in the page, and show you what that page would look like in the website (as well as saving that file).

Note: Other pages in your website will not update.

A quicker way to knit is using the keboard controls

`cmd + shift + K` (macOS) OR `ctrl + shift + K` (Windows)

5.4.3 Preview Your Book

If you want live updates to your changes, you can preview the page as you edit the book when you save individual .Rmd files. You can start the server in a work session by using the RStudio add-in "Preview book", or from the R console (in the bottom left window):

```
bookdown::serve_book()
```

But which files do we edit? Well alas, it's time to discuss the file setup.

5.5 File Setup Explanation {#file-setup} (this section is in review, Neha can try looking at the files in the bottom right and read along to what we currently have)

Look at all these .Rmd files! Recall that .Rmd -> .md -> .html files (from the diagram earlier). Hence, most of the files we really need to edit or create more of are .Rmd files.

However, 2 other important files are the `_output.yml` files and the `_bookdown.yml` files. They help tell bookdown what we want and what to do, especially when making our website. Everything you need to edit here is made clear in the template, so do not worry too much about getting the ride code so that your project builds!

The only .Rmd that must have some configuration details is the landing page: index.Rmd. This is what fills up the beginning of the index.Rmd file, before the ---, which tells bookdown to stop looking for configuration information.

Let's move on to discussing the breakdown of the .Rmd files. Each new page is defined by a new header, which starts with #, each subheaders have increasingly more # symbols (##, ###, and more all create smaller subheaders). Try to have only 1 single # as a header on one page! (You may get warnings otherwise).

But then how does bookdown know which page goes next on our sidebar? The order of the sidebar is completely dependent on the alphaetical order of the files (see your bottom right!). Our template has numbering first, to help ease our understanding.

5.6 Push to GitHub via RStudio

Now, we know what our files mean and how to edit them. How do we get this onto GitHub? We can write git commands into our Terminal/Command Prompt, or alternatively (and more easily), RStudio has a built-in git interface.

Now, we will be able to see a Git window in the top right. Click "Git" to open this window.

Let's say we only edited `index.Rmd`, now we see the newly edited files. Changed files that need to be updated on GitHub will show up in this window, like how `index.Rmd` is seen above. (Do not worry too much about `.DS_Store` and `.gitignore` do.) Let's try to push this change to GitHub.

13. Select all the edited files.

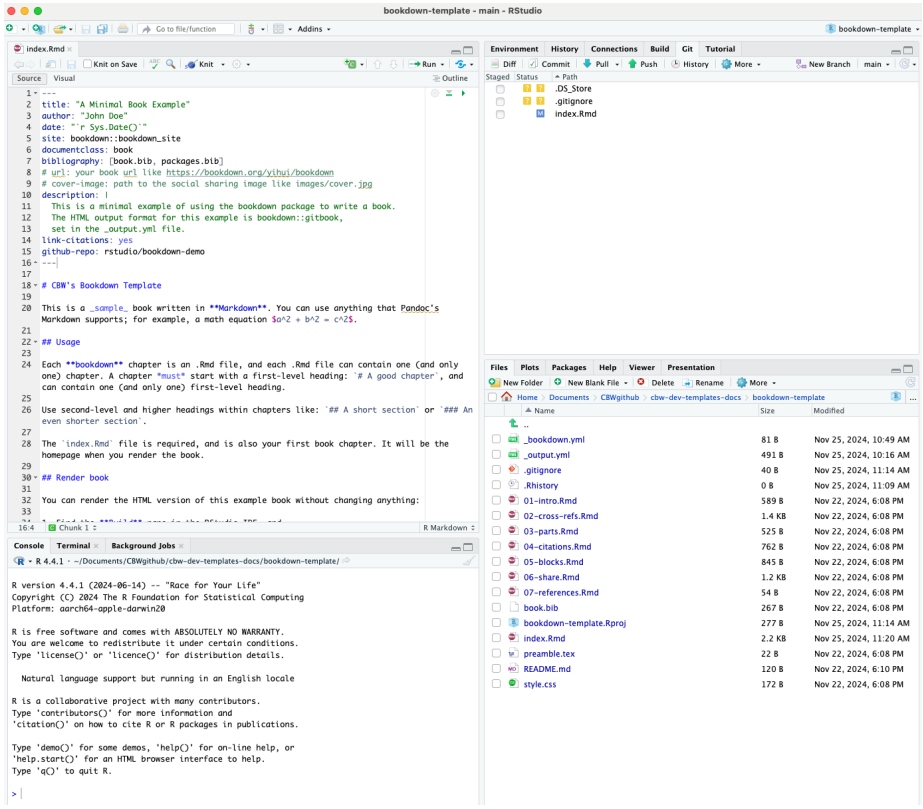
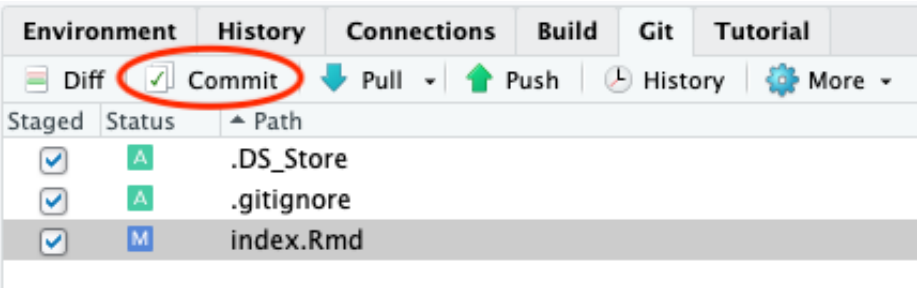
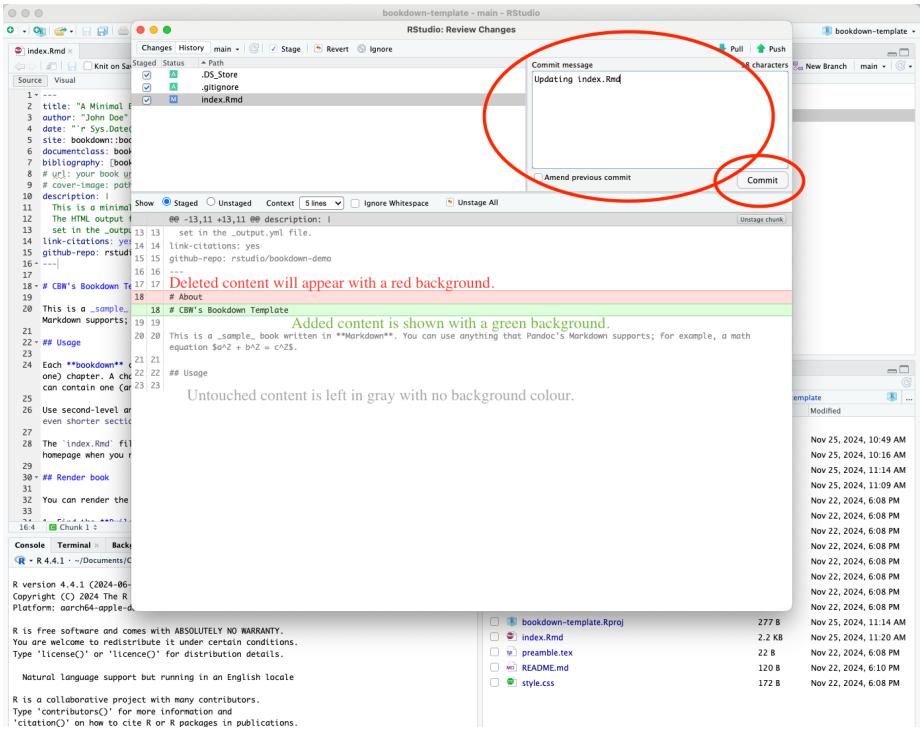


Figure 5.1: RStudio with Git window open



14. Then, click the Commit button, which appears above your selected items. A window pane will appear (shown below).



15. Add a commit message in the corresponding box, and then press commit below it.

16. A new window will show up, detailing your updates. Close this window and then press **Push** to push your updates to GitHub.

Now, we're done! We should see the updates on GitHub now. Also note, if we ever want to pull updates from GitHub, there is also a **Pull** button in the Git window within RStudio!

Git pushing puts your edits onto GitHub, *git pulling* takes the edits make on GitHub, and brings them to your local computer. For example, if one of your workshop team members made an edit, you want to have that edit on your computer before you start editing! It's a good idea to do this before you start editing, in case somehow your edits conflicts with their edits.

Git pushing will automatically update the website, you can see the updates and progress in the actions window we saw previously. (Check out your website on the web once it's done deploying!)

Chapter 6

Formatting Your Content - Markdown

Now we know which files to edit, but how can we edit these files? How do we format

6.1 Chapters

As mentioned earlier, headers are defined by a `#` before the title. Subheaders get increasingly more nested as add more `#` symbols before it. For example,

```
# Hello
```

would create a chapter title. Since there is only one `#` symbol, this would also create a new page. Again, try to keep only one chapter title per .Rmd file.

```
## Subheader
```

6.2 Subheader

This is what a subheader would look like.

```
### Subheader
```

6.2.1 Subheader

This is what a subheader with 3 `#` symbols would look like. You can add as many `#` symbols as you would like! There is also no limit to the the number of subheaders you can have.

An unnumbered subheader

Chapters and sections are numbered by default. To un-number a heading, add a `{.unnumbered}` or the `{-}` at the end of the heading. For example, the above subheader was written like this:

```
### An unnumbered subheader {-}
```

6.3 Cross-references

Cross-references are ways to link to different to different parts of your workshop website.

6.3.1 Chapters and sub-chapters

There are two steps to cross-reference any heading:

1. Label the heading: `# Hello world {#nice-label}`.
    - Leave the label off if you like the automated heading generated based on your heading title: for example, `# Hello world` = `# Hello world {#hello-world}`.
    - To label an un-numbered heading, use: `# Hello world {-#nice-label}` or `{# Hello world .unnumbered}`.
2. The reference your text as follows `[any text you want can go here](#cross)`. This will end up looking like: any text you want can go here.

6.4 Captioned figures and tables

Figures and tables *with captions* can also be cross-referenced from elsewhere in your book using `\@ref(fig:chunk-label)` and `\@ref(tab:chunk-label)`, respectively.

See Figure 6.1.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Don't miss Table 6.1.

```
knitr::kable(
  head(pressure, 10), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

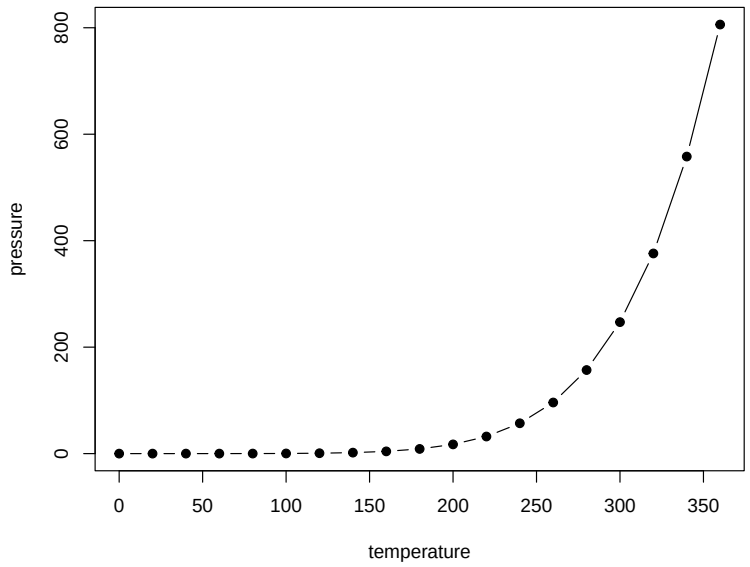


Figure 6.1: Here is a nice figure!

Table 6.1: Here is a nice table!

temperature	pressure
0	0.0002
20	0.0012
40	0.0060
60	0.0300
80	0.0900
100	0.2700
120	0.7500
140	1.8500
160	4.2000
180	8.8000

6.5 Parts

Notice how index.Rmd in our CBW Bookdown template has `# (PART) Introduction {-}` (followed by `# Welcome`). This creates the "Introduction" section on the sidebar.

There are already 2 main parts in the template: the introduction and modules sections.

If you want to add more parts, simply paste this: `# (PART) Introduction {-}` into a new .Rmd file, at the top of the file, before the main `# header`.

6.6 Footnotes

Footnotes are put inside the square brackets after a caret `^[]`. Like this one [1].

6.7 Citations (NOT SURE IF WE NEED WORKSHOPS NEED CITATIONS?)

Reference items in your bibliography file(s) using `@key`.

For example, we are using the **bookdown** package [Xie, 2024] (check out the last code chunk in index.Rmd to see how this citation key was added) in this sample book, which was built on top of R Markdown and **knitr** [Xie, 2015] (this citation was added manually in an external file book.bib). Note that the `.bib` files need to be listed in the index.Rmd with the YAML `bibliography` key.

The RStudio Visual Markdown Editor can also make it easier to insert citations: https://rstudio.github.io/visual-markdown-editing/#/citations

6.8 Equations

Here is an equation. Equations are written in latex.

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{6.1}$$

You may refer to using `\@ref(eq:binom)`, like see Equation (6.1).

6.9 Theorems and proofs

We also have specific syntax for theorems and proofs, for example, this code:

[1] This is a footnote.

```
::: {.theorem #tri}
For a right triangle, if $c$ denotes the *length* of the hypotenuse
and $a$ and $b$ denote the lengths of the **other** two sides, we have
$$a^2 + b^2 = c^2$$
:::
```

produces

Theorem 6.1. *For a right triangle, if c denotes the* length *of the hypotenuse and a and b denote the lengths of the* ***other*** *two sides, we have*

$$a^2 + b^2 = c^2$$

Labeled theorems can be referenced in text using `\@ref(thm:tri)`, for example, check out this smart theorem 6.1.

You can learn more about this here!

6.10 Callout blocks (will be added later)

The R Markdown Cookbook provides more help on how to use custom blocks to design your own callouts: https://bookdown.org/yihui/rmarkdown-cookbook/custom-blocks.html

Chapter 7

How to Render Code

Chapter 8

Brain Dump / FAQ

8.1 Danger Zones

- renaming your file with this project, doesn't change the .Rprog file!
- GIT TIPS PAGE ???
- BEST PRACTICES PAGE
- MAKE SURE TO EXPLAIN WHAT HAPPENS WHEN MERGE CONFLICTS APPEAR

Ex. Forgot to pull before editing? - Your git window will say (at the top of the window) your branch is ahead of the main branch - You will probably have to deal with a merge conflict - [this is pretty in-depth]

- update favicon.ico image in template with one with higher quality

8.2 Potential Errors & Bugs

- If you get a bug where your website builds into a website that looks like it is made using very simple html, you may have to change your permissions. If you see **"Permission denied"** in your warning messages, trying running this command `chmod -R u+w docs` in terminal, in the folder containing your docs folder [CLARIFY ?]
- **"Could not produce X output"** - by default, bookdown builds all possible formats: gitbook, pdf, epub versions. We only need the gitbook, so if you're having issues, change your build settings to only creating the gitbook [CLARIFY + INCLUDE IMAGES]
- A very common bug is **"missing X package"**, just install it using this command in your console (the bottom left window in RStu-

dio) `install.packages("missing package name, include these surrouding quotations")`

8.3 FAQ

Part I

DEVELOPERS GUIDE

Chapter 9

Build Site

9.1 How to edit _bookdown.yml

- add a new line, `output_dir: "docs"` to _bookdown.yml
- build the site
- add a `.nojekyll` file into the produced docs folder

9.2 How to edit _output.yml (RC)

- after *before:* change your workshop name link
- after *edit:* put the link to the workshop repo, and end the link with `/%s`
- save

9.3 Mandatory "index.Rmd" landing page

Each **bookdown** chapter is an .Rmd file, and each .Rmd file can contain one (and only one) chapter. A chapter *must* start with a first-level heading: `# A good chapter`, and can contain one (and only one) first-level heading.

Use second-level and higher headings within chapters like: `## A short section` or `### An even shorter section`.

The `index.Rmd` file is required, and is also your first book chapter. It will be the homepage when you render the book.

9.4 Build the book:

- "Build" button in RStudio IDE /OR/ bookdown::render_book()

- Preview the book: - updates on saves in viewer window bookdown::serve_book()

Before building

9.4.1 Render book

You can render the HTML version of this example book without changing anything:

1. Find the **Build** pane in the RStudio IDE, and
2. Click on **Build Book**, then select your output format, or select "All formats" if you'd like to use multiple formats from the same book source files.

Or build the book from the R console:

```
bookdown::render_book()
```

To render this example to PDF as a `bookdown::pdf_book`, you'll need to install XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.org/tinytex/.

9.4.2 Preview book

As you work, you may start a local server to live preview this HTML book. This preview will update as you edit the book when you save individual .Rmd files. You can start the server in a work session by using the RStudio add-in "Preview book", or from the R console:

```
bookdown::serve_book()
```

Chapter 10

Git Instructions

10.1 How to Make a Git Repo (RC)

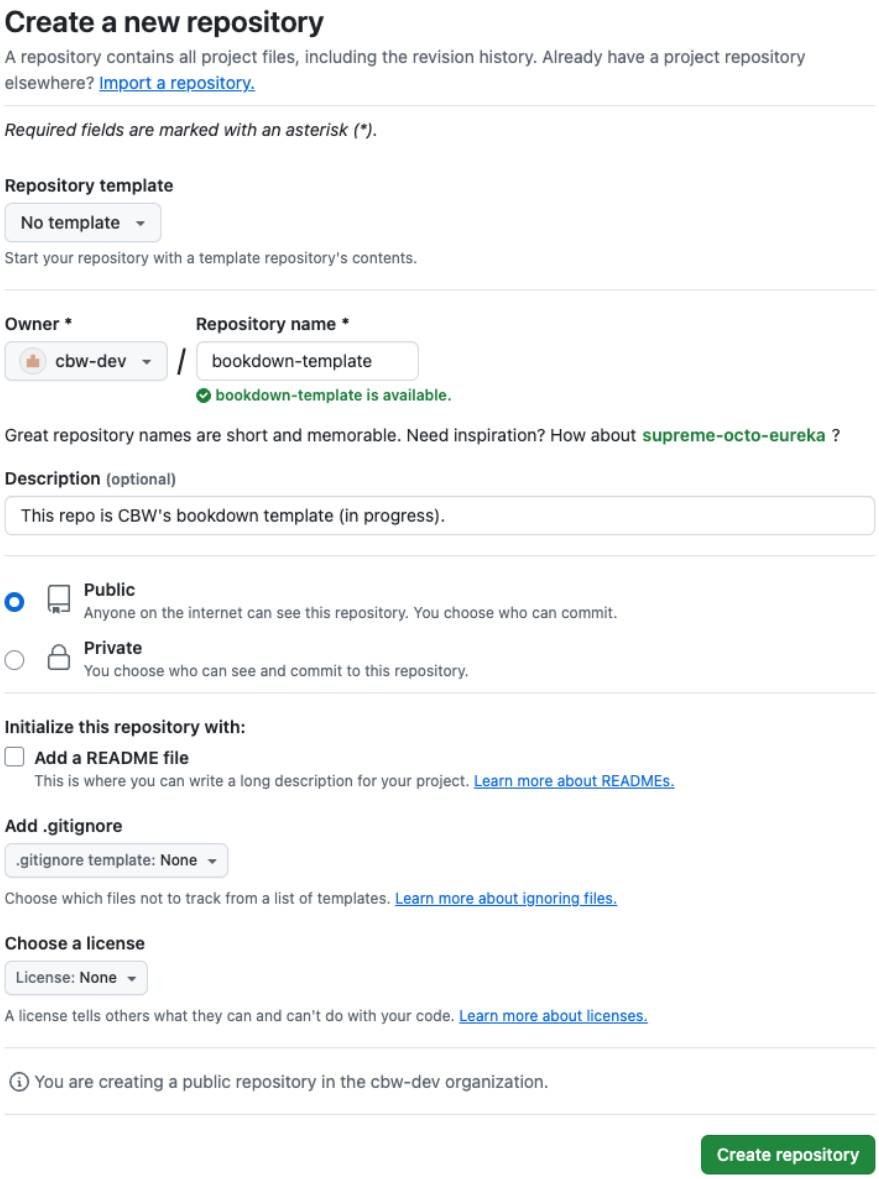
1. Go to https://github.com/cbw-dev (CHANGE?) and scroll to your repositories.

2. Click the green "New" button to the right of the repositories search bar.

3. Create the new repository. Give it a *name* and *description*. Select *Public* instead of private, as shown below.

   Warning

   MAKE NAMING CONVENTION



4. Click the green *Create repository* button at the bottom.

Now, we already have a local project. Now we want it on GitHub, so everyone on your team can make changes to the workshop! Let's make the GitHub connection (i.e let's add our local code to GitHub!)

10.1.1 How to Make the Git Connection (Adding your Local Repo to GitHub)

After the previous step, you will be brought to this page.The only things that will differ are the name of the repo.

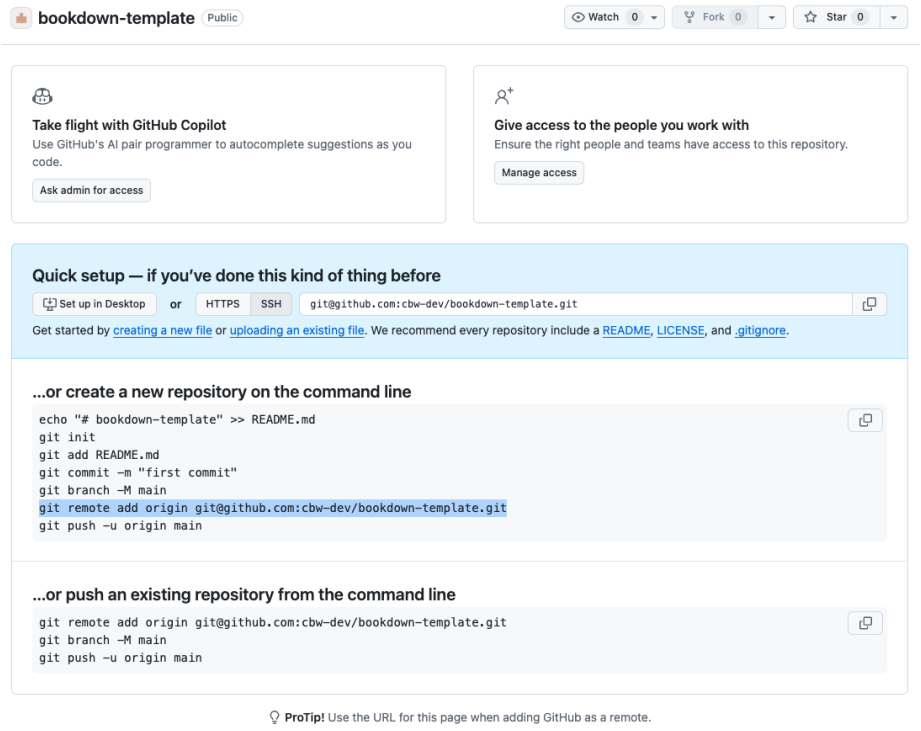


Figure 10.1: landing page after making a git repo

5. Open Terminal (Mac) or Command Prompt/Powershell (Windows).
6. Go to where we created the bookdown project.
7. Once inside the folder with the project. Let's make the git repo. First we initialize: `git init`. (Put this into terminal and press enter.)
8. Let's add all the files: `git add *`
9. Let's commit these files, with a descriptive message to help make it clear to others what we just did. For now, our message can be simple: `git commit -m "first commit"`. (Put this into terminal and press enter.)
10. Next, put this into terminal and press enter: `git branch -M main`.
11. **Important:** This step is why I highlighted that specific text above. Copy that command, and put it into terminal. Gen-

erally, it will look something like this: `git remote add origin git@github.com:cbw-dev/NAME-OF-YOUR-REPO.git`

12. Next, put `git push -u origin main` into terminal and press enter.

All the steps are shown below.

```
jqiu@ML8034-JQIU ~ % cd Documents/CBWgithub/cbw-dev-templates-docs/bookdown-template
jqiu@ML8034-JQIU bookdown-template % git init
Initialized empty Git repository in /Users/jqiu/Documents/CBWgithub/cbw-dev-templates-docs/bookdown-template/.git/
jqiu@ML8034-JQIU bookdown-template % git add *
jqiu@ML8034-JQIU bookdown-template % git commit -m "first commit"
[main (root-commit) 44aae1d] first commit
 15 files changed, 266 insertions(+)
  create mode 100644 01-intro.Rmd
  create mode 100644 02-cross-refs.Rmd
  create mode 100644 03-parts.Rmd
  create mode 100644 04-citations.Rmd
  create mode 100644 05-blocks.Rmd
  create mode 100644 06-share.Rmd
  create mode 100644 07-references.Rmd
  create mode 100644 README.md
  create mode 100644 _bookdown.yml
  create mode 100644 _output.yml
  create mode 100644 book.bib
  create mode 100644 bookdown-template.Rproj
  create mode 100644 index.Rmd
  create mode 100644 preamble.tex
  create mode 100644 style.css
jqiu@ML8034-JQIU bookdown-template % git branch -M main
jqiu@ML8034-JQIU bookdown-template % git remote add origin git@github.com:cbw-dev/bookdown-template.git
jqiu@ML8034-JQIU bookdown-template % git push -u origin main
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 11 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (17/17), 5.74 KiB | 2.87 MiB/s, done.
Total 17 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:cbw-dev/bookdown-template.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
jqiu@ML8034-JQIU bookdown-template %
```

Figure 10.2: all the git steps typed out into terminal with results

10.2 Updating GitHub via RStudio

Now, close your RStudio session, and reopen it.

Now, we will be able to see a Git window in the top right. Click "Git" to open this window.

Let's say we only edited `index.Rmd`, now we see the newly edited files. (Do not worry too much about `.DS_Store` and `.gitignore` do.) Let's try to push this change to GitHub.

13. Select all the edited files.

14. Then, click the Commit button, which appears above your selected items. A window pane will appear (shown below).

15. Add a commit message in the corresponding box, and then press commit below it.

16. A new window will show up, detailing your updates. Close this window and then press **Push** to push your updates to GitHub.

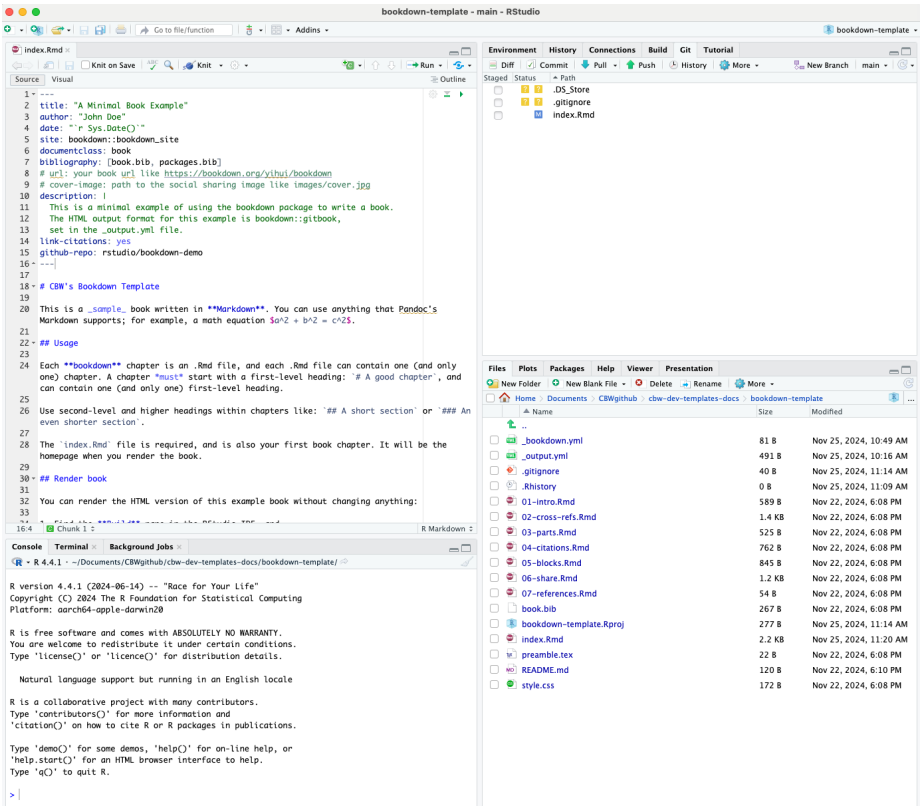


Figure 10.3: RStudio with Git window open

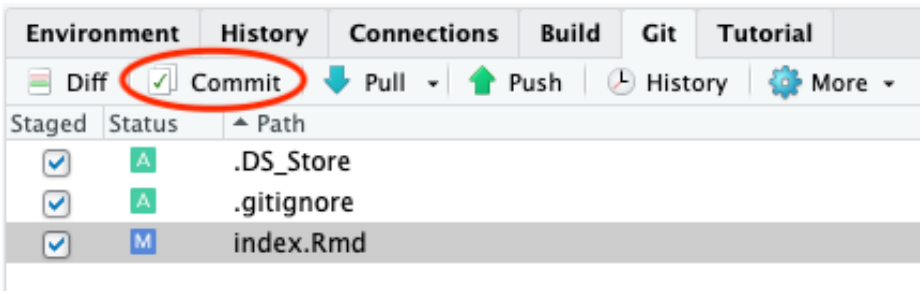


Figure 10.4: selected files in the git window and the commit button highlighted

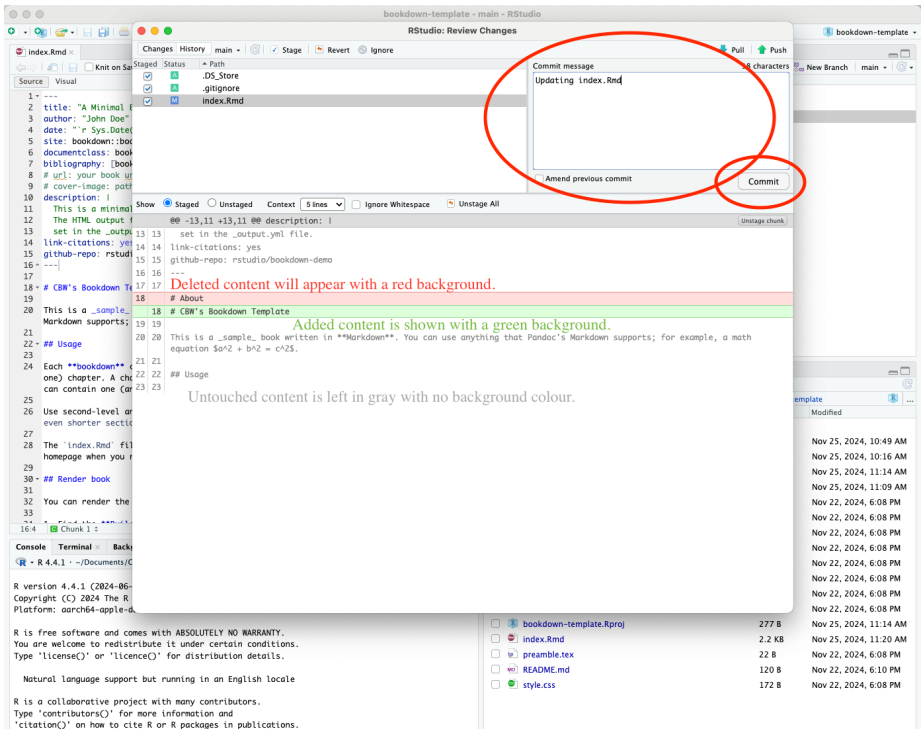


Figure 10.5: git commit window pane

Figure 10.6: post git commit window

Now, we're done! We should see the updates on GitHub now. Also note, if we ever want to pull updates from GitHub, there is also a **Pull** button in the Git window within RStudio!

Chapter 11

How to Deploy Your Workshop Website

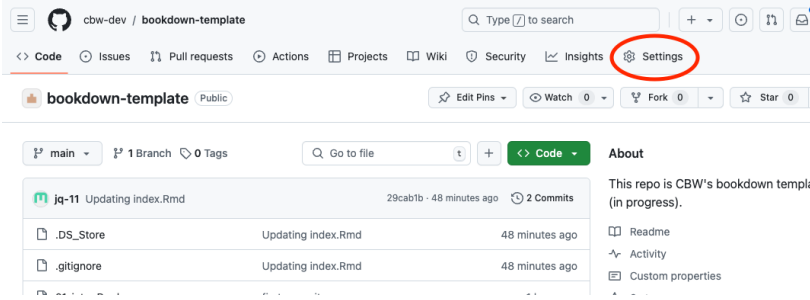
Let's recap.

We've made a bookdown project that builds into a website. We've reconfigured the output to go to a folder called "docs" (output_dir: "docs"). We've pushed our content onto github, and also made a ".nojekyll" file, which we placed into docs.

Now in our ./docs folder, we have a bunch of html files that make up our website. We want GitHub to look at these files in the docs folder and host the website for us!

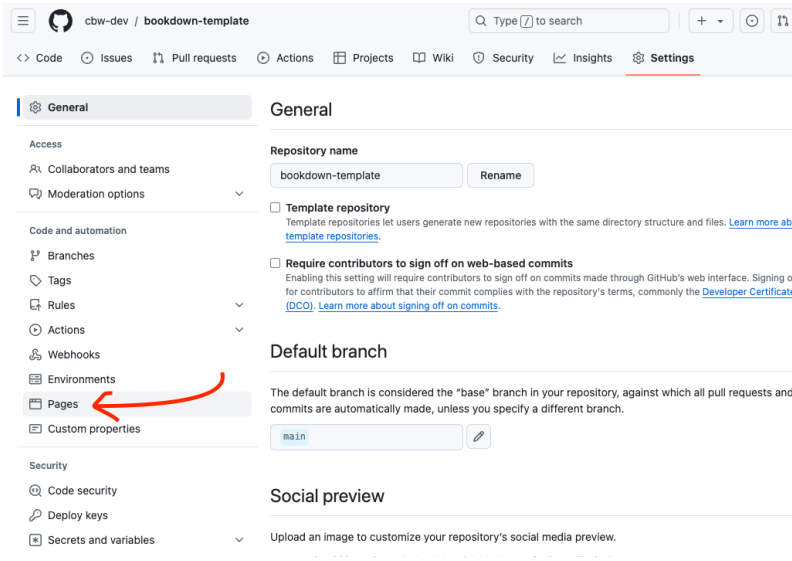
We deploy our website using GitHub pages. GitHub pages uses jekyll, so the .nojekyll file tells it to no longer rely on jekyll. Now, all we need to do is tell GitHub pages to deploy (create/update the website) from our docs folder.

1. Go to your repo on GitHub.

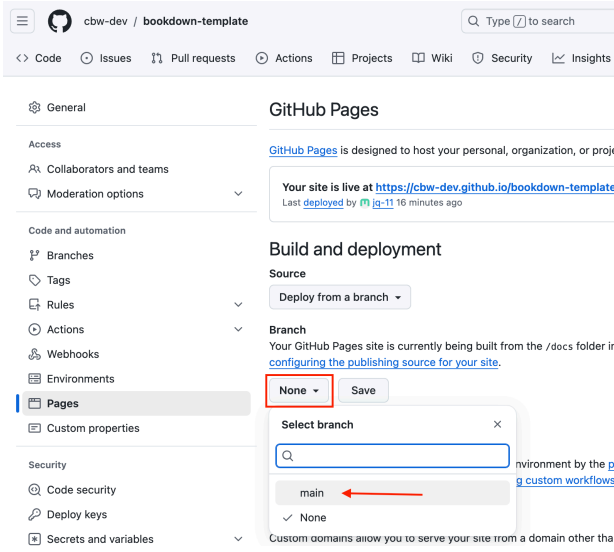


2. In the top navigation bar, select settings.

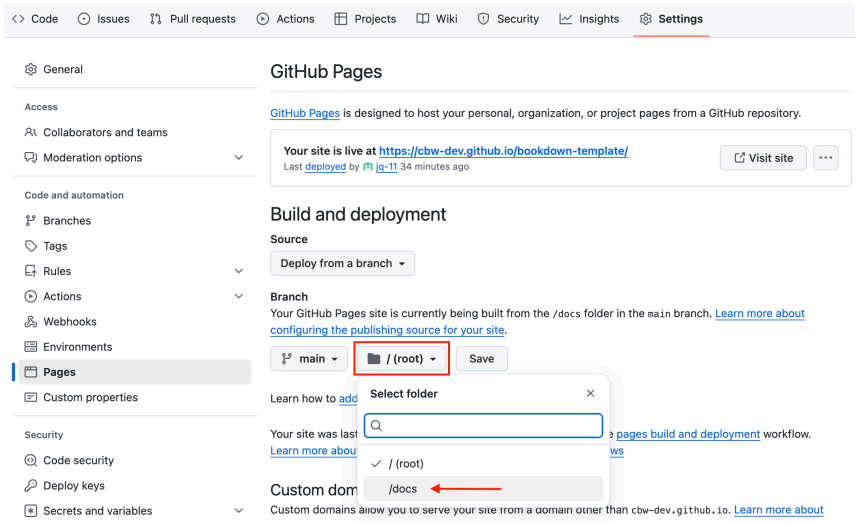
3. Then, go to the pages sidebar.



4. "Deploy from a branch" is already selected, which is what we want. We must change the branch from "none" to main.



5. Then, change the folder from `/ root` to `/docs`. Then press save.



Great! Now we're waiting on the page to build and deploy, which should take less than a minute.

To see updates, go to the **Actions** page (found along the top navigation bar. This will help you understand how the deploy is working, and if it succeeded or failed.

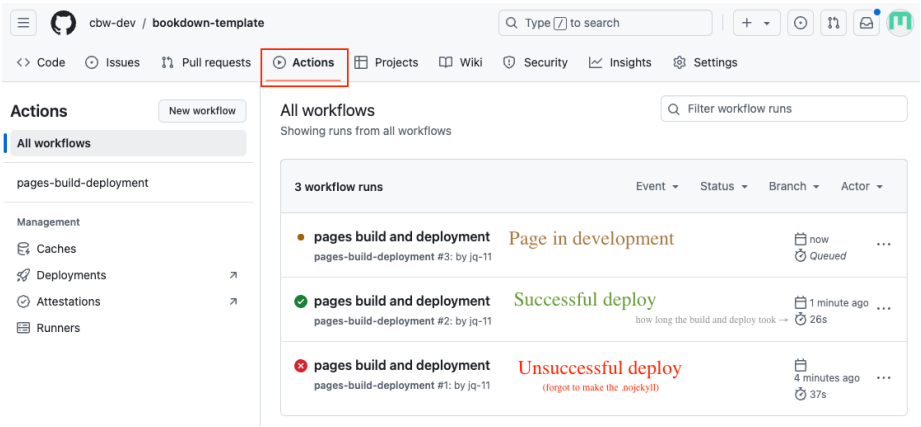
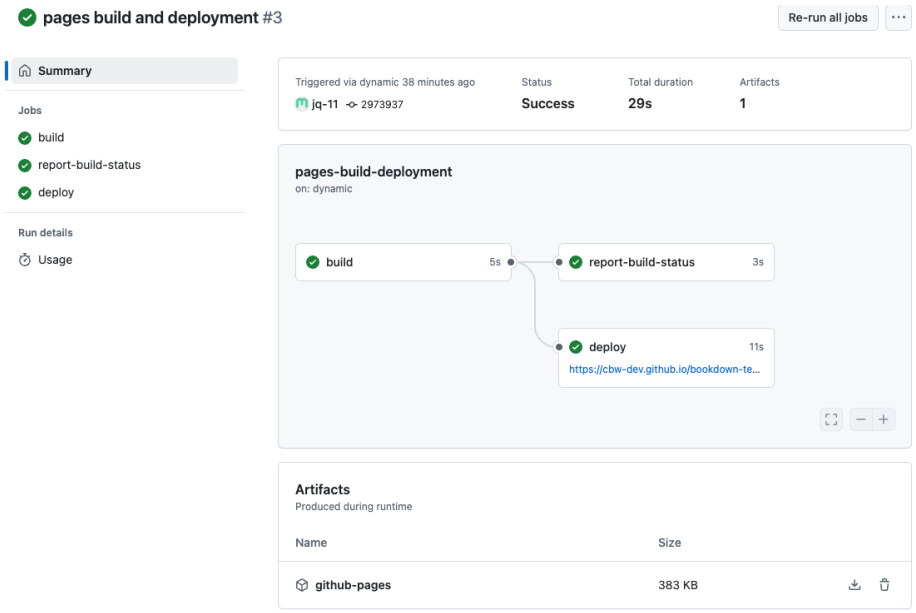
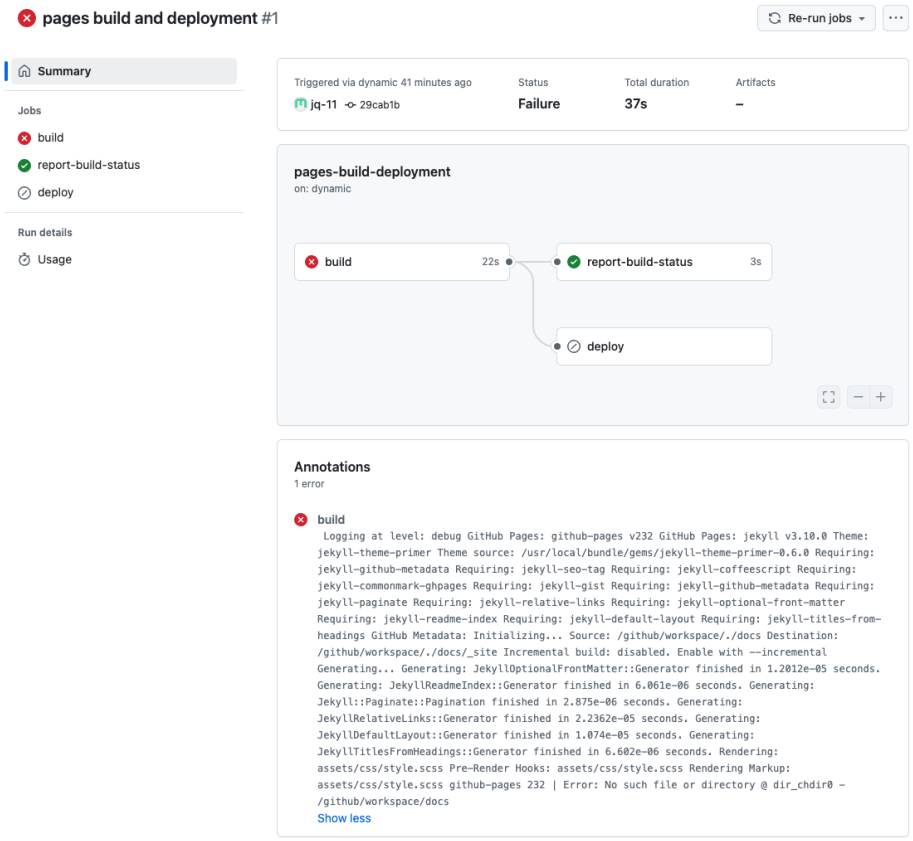


Figure 11.1: Image showing the different possibilities of deploy a github page

You can click **pages build and deployment** for updates. It will give you errors (which may not be very clear) or the link of your deployed page!


pages build and deployment #3
Re-run all jobs
Summary
Jobs
build
report-build-status
deploy
Run details
Usage
Triggered via dynamic 38 minutes ago
jq-11 2973937
Status
Success
Total duration
29s
Artifacts
1
pages-build-deployment
on: dynamic
build
5s
report-build-status
3s
deploy
11s
https://cbw-dev.github.io/bookdown-te...
Artifacts
Produced during runtime
Name
Size
github-pages
383 KB



Click around to explore more!

Bibliography

Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.org/knitr/. ISBN 978-1498716963.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*, 2024. URL https://github.com/rstudio/bookdown. R package version 0.40.