Using Quarto and RStudio to build Data fellowship module materials

MACEPA Data Fellowship - Training Materials

Table of contents

Starting Point

These resources are intended to help us build and maintain materials and modules for the Data Fellowship training materials. These can also serve as a useful resource when wanting to use Quarto and GitHub pages for other aspects of our work. Therefore these resources will continue to grow and evolve with us - send suggestions for content or improvements through the issues tab on the GitHub Repo.

System Requirements

Before you can start using Quarto to build out webpages or reports you need to ensure the following are installed on your computer

- R - https://www.r-project.org/
- Rstudio - https://posit.co/download/rstudio-desktop/
- Quarto - https://quarto.org/docs/download/
- Git - https://git-scm.com/downloads

Ensure you have a GitHub **profile** and are a member of **PATH-Global-Health organization**

If you haven't installe these packages before ensure the `usethis` `gh` `renv` and `quarto` packages are installed by typing this in the RStudio console:

```
install.packages(c("usethis", "gh", "renv", "quarto", "gitcreds"))
```

Configure Git with RStudio

After installing Git and setting up a GitHub account, the next step is to configure Git in RStudio. If you've linked your GitHub account to R already then skip ahead to **?@sec-stage**

1. Open RStudio.
2. Configure Git in RStudio:

- Go to **Tools** > **Global Options**.
- In the left-hand sidebar, click **Git/SVN**.
- Make sure the path to the Git executable is correct (this should automatically detect where Git was installed). For Windows, it might look something like `C:/Program Files/Git/bin/git.exe`.
- Click **Ok**

3. We want Git to know who we are so it can associate changes with you. Enter the following code in your console and replace the user name and email to those linked to your github account

```
usethis::use_git_config(user.name="Jane Doe", user.email="jane@example.org")
```

Linking to GitHub

When its time to send our files to GitHub, we need GitHub to know who we are and that we have permission to write to our repositories. We can establish this authorisation through either a **personal access token** or a **SSH key**.

We can generate a PAT through GitHub directly through https://github.com/settings/tokens and click "Generate token". Look over the scopes; I highly recommend selecting `repo`, `user`, and `workflow`. Copy the generated PAT to your clipboard. Or leave that browser window open and available for a little while, so you can come back to copy the PAT.

Another option is to create this directly in R using:

```
usethis::create_github_token()
```

Recommended scopes will be pre-selected if you used `create_github_token()`.

Now in R call, `gitcreds::gitcreds_set()` to get a prompt where you can paste your PAT:

```
> gitcreds::gitcreds_set()

? Enter password or token: ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
-> Adding new credentials...
-> Removing credentials from cache...
-> Done.
```

You should be able to work with GitHub now, i.e. push and pull!

What stage are you building from?

Before getting started we need to understand from what point we are building resources from - the following starting points cover the most common starting points we tend to find ourselves in

Starting Point	Description
Building from scratch	Start here if you want to learn about setting up a completly new resource and with tips and steps about using GitHub for the first time too
Building from an existing GitHub Repository	Start here if you are collaborating on resources and want to add additional materials to an already existing module

1 Building From Scratch

These resources are intented to be used if you are building module resrouces from scratch and need to set up a new Quarto Project, GitHub Repository and Associated GitHub Pages to host the materials.

See the table of contents to the right of this page for a summary of the steps involved in this process. Ensure you have followed the System requirements and have set up your GitHub profile and linked it in RStudio as detailed in the Start Point section.

1.1 Step 1: Set up new Quarto Book Project in RStudio

We suggest using the Quarto Book format for hosting training materials - there are several other Quarto output styles that you can work with but this what has worked well for this style of content delivery.

1. Open **RStudio**.
2. Go to the menu: `File` → `New Project...` → `New Directory` → `Quarto Book`.
3. **Name the project** and select the location where you want to save the project files on your computer. (I like to use a folder called github and all my repos exist in separate folders but pick what works for you!)
4. Ensure the checkbox for `Create a Git repository` is selected (this initializes Git locally in the project).
5. And also tick the checkbox for `use renv with this project`. This will create a local environment for the project and store all your package dependencies in an renv.lock file, this is an important aspect for hosting our site on gh-pages later on.
6. Click **Create Project**.

RStudio will open up your new project. The Quarto Book project structure will include:

- **_quarto.yml:** A configuration file where you can define the book's structure, theme, output format, and other settings.
- **index.qmd:** The landing page or introduction to the book. This file can serve as the cover page with a description of your project.
- **qmd files for chapters:** A few example .qmd files will be provided as placeholders for different chapters, which you can edit or replace with your content.

- An **renv** folder: This directory contains your project-specific package library, and an `renv.lock` file will be created to lock down package versions and dependencies, ensuring the project remains reproducible and code can be executed on publishing to our online portal.

1.2 Step 2: Stage initial commit to GitHub

Before linking this project to GitHub, we need to make sure the initial project files are committed to the local Git repository.

- Head to the `Terminal` tab next to the `console`.
- In the terminal, check which files are ready to be staged using:

```
    git status
```

- This will show the files that have been modified or are new and need to be added to the repository. It will also tell us which branch we are working on in brackets. If this is the `master` branch lets change it to be called `main`.

```
git branch -m master main
```

- To add all files to the staging area (the files you want to include in your commit - here this will just be our default Quarto Book Project files which is okay, run:

```
git add .
```

- The `.` adds all the files in the current directory
- After staging the files, you'll need to commit them. The commit message should describe what changes or additions you're committing.To commit the changes, use: The `-m` flag allows you to add a message in quotes (`" "`) describing the commit.

```
git commit -m "Initial commit for Quarto website"
```

1.3 Step 3 Push the local project to GitHub using `usethis::use_github()`

We now want to link our local repository to GitHub and specially we want it to be part of the PATH-Global-Health GitHub organisation. We can use the following code to do this, run this in your console:

```r
usethis::use_github(
organisation = "PATH-Global-Health",
visibility = "public"
)
```

This command will:

- Create a new GitHub repository.
- Link your local project to this repository.
- Push the project files to GitHub.

This should then open up the repository automatically in you browser.

1.4 Step 4: Setting up gh-pages

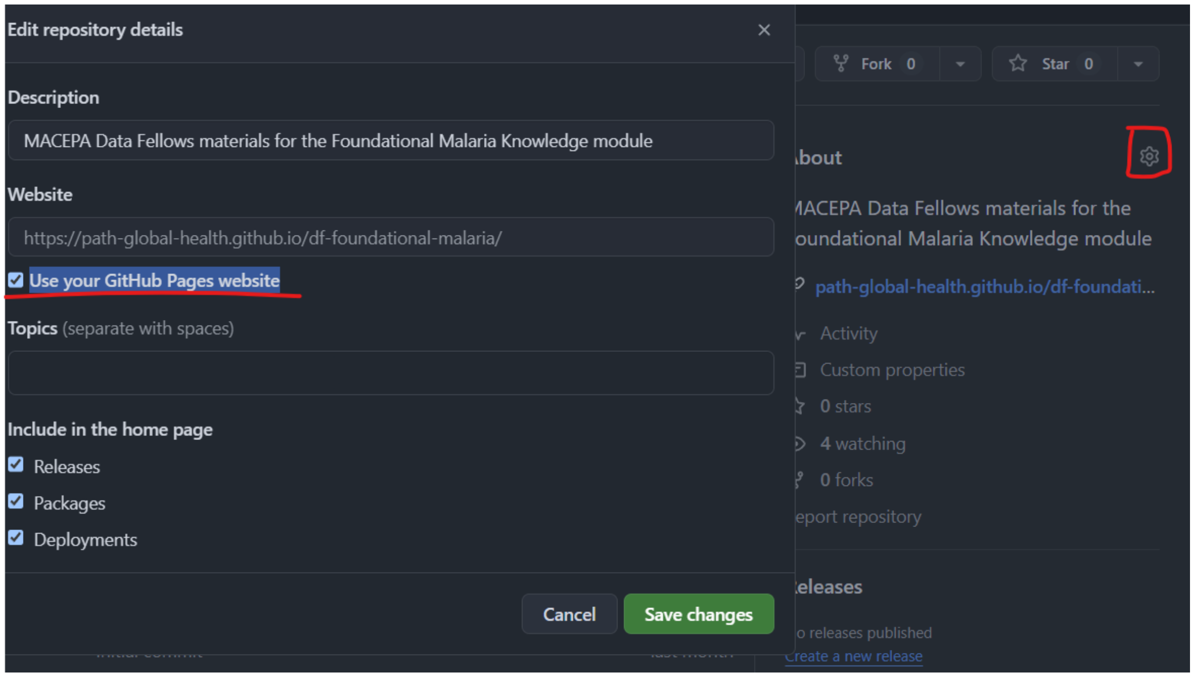
Once our repository is on GitHub, we can configure the GitHub Pages site - which is where our module resources will be hosted. Use the `usethis::use_github_pages()` function to set the publishing branch for GitHub Pages.

To publish from the `gh-pages` branch, run:

```r
usethis::use_github_pages(branch = "gh-pages")
```

If we head to our GitHub repository online we want to add some details to the repo page:

Head back to the `<> Code` tab and in the `About` section on the top right open the settings wheel - Under `Website` check the box next to: [x] "Use your GitHub Pages website" as shown in the image below.


Edit repository details

Description

MACEPA Data Fellows materials for the Foundational Malaria Knowledge module

Website

https://path-global-health.github.io/df-foundational-malaria/

☑ Use your GitHub Pages website

Topics (separate with spaces)

Include in the home page

☑ Releases

☑ Packages

☑ Deployments

In addition we can add a short description in this section as in the above image e.g. "MACEPA Data Fellows materials for the [insert module title]".

1.5 Step 5: Automate Deployment with GitHub Actions

This is something I've found works best for me and my workstyle when creating these modules. Instead of ever rendering my work locally and then publishing this to GitHub I include a GitHub Action command so that when I commit and push changes to the repository GitHub will automatically render the new outputs to the `gh-pages` site.

Manually building and deploying our project every time we make a change can be time-consuming and prone to error. So by configuring GitHub Actions, we can automate the entire publishing process. Whenever we push changes to the repository (e.g., updated content, code adjustments), GitHub Actions will automatically trigger the workflow to build and deploy our site. Which saves us time and reduces manual effort. This also helps ensure that everyone is working on the most recent version of the materials, with automatic deployment occurring in the background.

1.5.1 Set up

More detials on setting up GitHub actions can be found here: https://quarto.org/docs/publishing/github-pages.html.

1. In your Quarto project directory, create a folder called `.github/worflows`
2. Inside `.github/workflows/`, create a file called `quarto-publish.yml` - You do this from within RStudio by heading to the `files` pane and into the `workflows folder` → `new blank file` → `Text file` and this opens up in R studio and then save this as `quarto-publish.yml`
3. Add the following content to the `quarto-publish.yml`

```yaml
on:
  workflow_dispatch:
  push:
    branches: main

name: Quarto Publish

jobs:
  build-deploy:
    runs-on: ubuntu-latest
    permissions:
      contents: write
    steps:
      - name: Check out repository
        uses: actions/checkout@v4

      - name: Set up Quarto
        uses: quarto-dev/quarto-actions/setup@v2

      - name: Install R
        uses: r-lib/actions/setup-r@v2
        with:
          r-version: '4.2.0'

      - name: Install R Dependencies
        uses: r-lib/actions/setup-renv@v2
        with:
          cache-version: 1

      - name: Install TinyTeX
        run: |
          Rscript -e 'tinytex::install_tinytex(force = TRUE)'
          echo "PATH=$HOME/.TinyTeX/bin/x86_64-linux:$PATH" >> $GITHUB_ENV

      - name: Ensure TinyTeX Path
```

```
        run: echo "$HOME/.TinyTeX/bin/x86_64-linux" >> $GITHUB_PATH

      - name: Render and Publish
        uses: quarto-dev/quarto-actions/publish@v2
        with:
          target: gh-pages
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

4. call `renv::snapshot()`, `snapshot()` updates the lockfile with metadata about the currently-used packages in the project library.

This `quarto-publish.yml` I have worked on standardising across the projects I've developed for the Data Fellows so far - should you have any issues please reach out to Hayley to help troubleshoot!

Other options for publishing content can be found here: https://quarto.org/docs/publishing/

1.6 Step 6 Push updates to GitHub

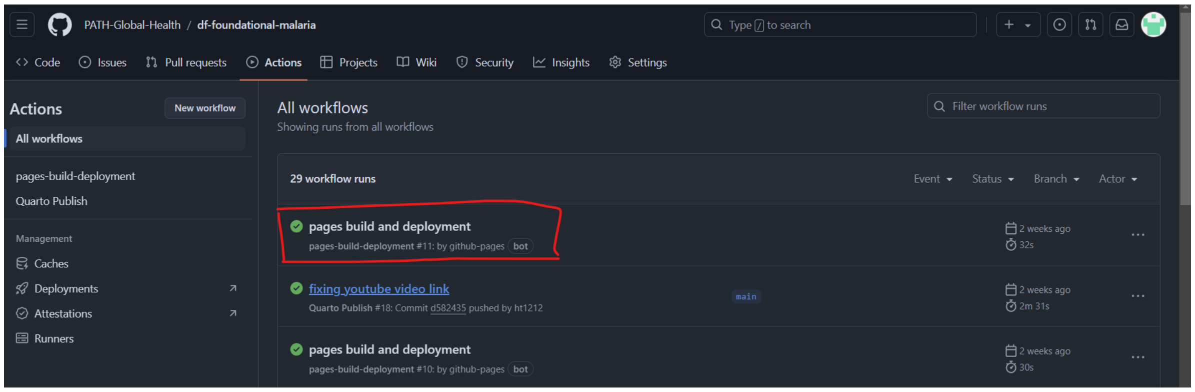
We can now push the all of the following changes to GitHub and test if the publishing action has worked - don't worry that we haven't changed any content yet we will get there!

Switch to the `Terminal` pane and run the following:

```
git add .
git commit -m "deploying and testing github actions and publishing"
git push
```

1.7 Step 7 Check build and status

Head to the repo and go to the Actions Tab - we're hoping to see something like this:



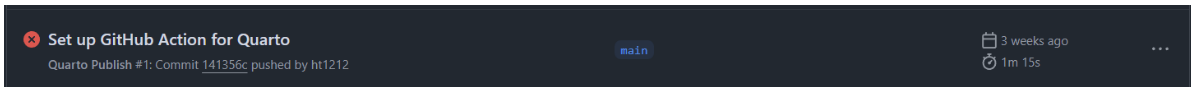
29 workflow runs

pages build and deployment pages-build-deployment #11: by github-pages bot		2 weeks ago 32s
fixing youtube video link Quarto Publish #18: Commit d582435 pushed by ht1212	main	2 weeks ago 2m 31s
pages build and deployment pages-build-deployment #10: by github-pages bot		2 weeks ago 30s

Which shows us out Build action was completed, worked correctly and the website page should be rendered!

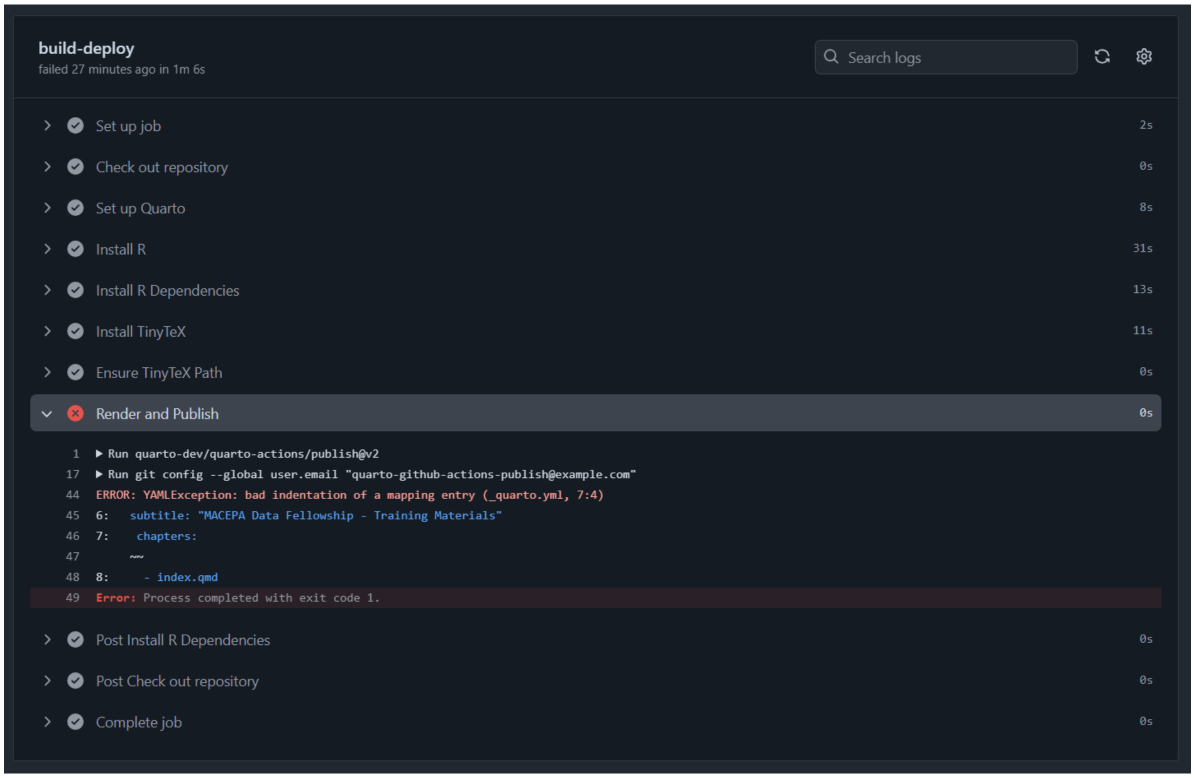
If you head back to the `<>code` tab and click the link we activated in the `About` section this should open up our github pages site and will hopefully currently be populated with Quartos book template!

If the build action failed we will see somthing like this:



Set up GitHub Action for Quarto Quarto Publish #1: Commit 141356c pushed by ht1212	main	3 weeks ago 1m 15s

If we click into this GitHub will give us a good indication of what exactly failed:



```
build-deploy
failed 27 minutes ago in 1m 6s

Set up job                      2s
Check out repository            0s
Set up Quarto                   8s
Install R                       31s
Install R Dependencies          13s
Install TinyTeX                 11s
Ensure TinyTeX Path             0s
Render and Publish              0s
▶ Run quarto-dev/quarto-actions/publish@v2
▶ Run git config --global user.email "quarto-github-actions-publish@example.com"
ERROR: YAMLException: bad indentation of a mapping entry (_quarto.yml, 7:4)
6:   subtitle: "MACEPA Data Fellowship - Training Materials"
7:    chapters:
     ~~
8:      - index.qmd
Error: Process completed with exit code 1.
Post Install R Dependencies     0s
Post Check out repository       0s
Complete job                    0s
```

The error here is caused by incorrect indentation in the YAML file. YAML is strict about indentation, and in this instance the curhttps://quarto.org/docs/guide/rent `_quarto.yml` file, the `chapters:` key was incorrectly indented. We can correct this, commit and push the changes and our site was deployed correctly!

Often build errors are informative and can be fixed easily enough - if you're struggling reach out to Hayley and we can always trouble shoot together!

1.8 Step 8 Adding content

Now we have all the set up out the way we can start adding our content and updating the themeing!

This step will be very context depenent on the type of materials we want to add to our modules so I have outlined some common steps across materials that have been built so far and we can always add more as we go.

The Quarto Guide is going to be super useful https://quarto.org/docs/guide/ especially if you're new to working in Markdown - I won't replicate the resources here and recommend using the Quarto guide as a reference point when writing in Quarto.

There is always the option in Quarto to use the `visual` editor - this simplifies the process of writing and formatting content in markdown formats and allows users to focus on writing without needing to remember markdown syntax, making it easier for beginners and those unfamiliar with code-based formatting. The Visual Editor provides an interface similar to popular word processors like Microsoft Word or Google Docs.

The Visual Editor includes toolbars with buttons for common formatting options such as:

- Bold and italic text
- Creating headings and subheadings
- Adding bullet points or numbered lists
- Inserting links, images, and tables
- Adding callouts for notes, warnings, or tips
- You can still insert and edit code chunks directly in the Visual Editor. These code chunks are executed as usual, and their outputs (tables, figures, etc.) are displayed inline, making it easier to integrate code into your documents.

To use the Visual Editor in RStudio or other supported IDEs:

- Open any Quarto (`.qmd`) or markdown file in your project.
- Click the **"Visual"** button in the top right corner of the editor pane to toggle between the standard markdown editor and the Visual Editor.

1.8.1 Understanding the `_quarto.yml` file

The `_quarto.yml` file is a configuration file that controls the structure and appearance of your Quarto project, such as the website's title, menu, and theme.

For a **Quarto book**, the `_quarto.yml` file specifies the structure, appearance, and behavior of the book. Below is an example of what the structure might look like and what you will see when you open this initially in your project.

```yaml
project:
  type: book

book:
  title: "My Quarto Book"
  author: "Author Name"
  chapters:
    - index.qmd
    - introduction.qmd
    - chapter1.qmd
    - chapter2.qmd

format:
  html:
    theme: cosmo
    toc: true
    number-sections: true
    css: styles.css
```

We want to start by modifying the `title` and `author` fields under `book` to what is relevant for the module (we can just delete the author line as it's not needed here)

We can also add a `subtitle` that's nice to include in all our topics like this:

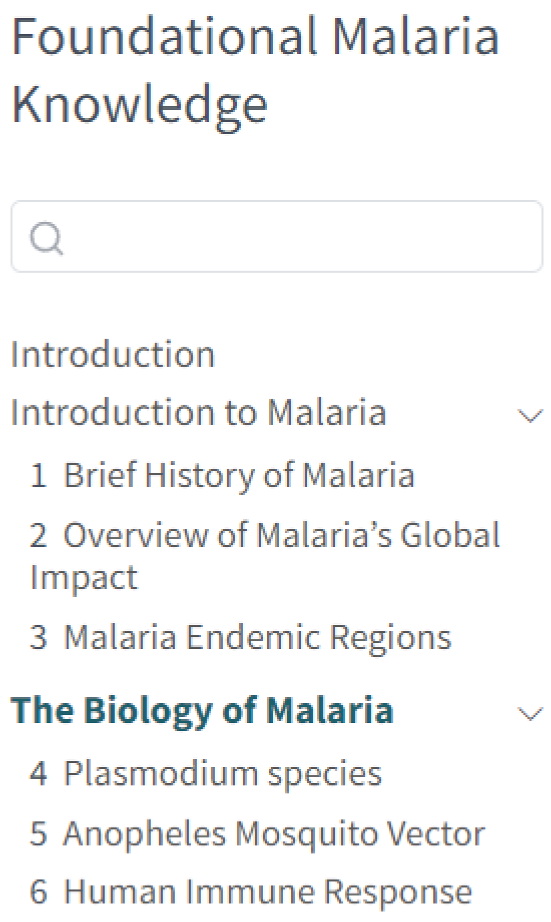
```
book:
  title: Foundational Malaria Knowledge
  subtitle: MACEPA Data Fellowship - Training Materials
```

The chapters section defines the order of the chapters in your book. It links to the `.qmd` files that will become the different sections of the book:

```
book:
    chapters:
        - index.qmd           # Main page or introduction
        - introduction.qmd    # First chapter
        - chapter1.qmd        # Subsequent chapters
        - chapter2.qmd
```

You can add or remove chapter files as needed. Just make sure each file you list here exists in the project.

For some of our modules we have several topics within a module and sub topics within them so we can use the `part` as well as `chapters` to achieve something like this:


Foundational Malaria Knowledge
Introduction
Introduction to Malaria
1 Brief History of Malaria
2 Overview of Malaria's Global Impact
3 Malaria Endemic Regions
The Biology of Malaria
4 Plasmodium species
5 Anopheles Mosquito Vector
6 Human Immune Response

To do so you can set the `_quarto.yml` to something this:

```yaml
chapters:
- index.qmd
- part: topic1-intro.qmd
      chapters:
      - topic1a-history-malaria.qmd
      - topic1b-global-impact.qmd
      - topic1c-endemicity.qmd
- part: topic2-intro.qmd
      chapters:
      - topic2a-plasmodium-spp.qmd
```

```
        - topic2b-lifecycle.qmd
        - topic2c-immunity.qmd
   - part: topic3-intro.qmd
        chapters:
        - topic3a-anopheles-trans.qmd
        - topic3b-environment.qmd
        - topic3c-human-behaviour.qmd
```

In each of these instances the title that is in the `.qmd` file becomes the heading that is shown on the final website as shown in the image above.

Note

Once you start adding `.qmd` files with module content make sure to replace the defualt files and include these in the `_quarto.yml` to ensure the modules are displayed on the site. Often I start adding content to the templates and renaming the files before creating new `.qmd` files

1.8.2 Adding custom styling

We've already created custom PATH theming and I've saved all the necessary files in the `data-fellows box folder` under 'quarto-theming' Data fellowship program planning → Technical Content Organization → quarto-theming

Copy all of these into your root folder and add the following to your `_quarto.yml` file

```
format:
  html:
    theme:
    - cosmo
    - custom.scss
    template-partials: title-block.html
    css: include/webex.css
    include-after-body: include/webex.js
    embed-resources: true
  pdf:
    documentclass: scrreprt
editor: visual
```

1.8.3 Embedding PDFs

If you have a slide deck to host on the website we first need to save this as a pdf and ensure it it then saved in the same root directory of our Quarto project.

You can use the following syntax to embed the pdfs inside a `.qmd` file replacing the file name with the associated file name of your file.

```
<iframe src="file-name.pdf" width="800" height="600">

</iframe>
```

1.8.4 Embedding YouTube Videos

If you want to include videos hosted on YouTube an easy was to achieve this is with the following syntax, replacing the link with that to the video of interest:

```
https://www.youtube.com/embed/bJ6nS-I-HiM
```

1.8.5 Interative Quizzes

You can use the `webexercises` package to produce quizes and is what I used in the Foundational Epi module

The documentation is a good place to start here and the source code in the Foundational Epi Module repository if you want to develop something similar!

1.8.6 Publishing content

Once you have created the necessary module content in `.qmd` files, have updated the `_quarto.yml` file with the correct references for each chapter we can again push out changes to github and check out our newly published site!

Remeber:

```
git add .
git commit -m "commit message"
git push
```

And it's as simple as that!

1.9 Wrapping It All Together: Maintaining Your Quarto Project

By following these steps, you've built a robust workflow for creating, managing, and publishing your Quarto Book from scratch. Here are a few final tips to help you maintain and expand your project effectively:

1. **Iterating on Content Content updates:**
   - As your project evolves, you'll likely need to update chapters or add new ones. Remember, every time you modify a .qmd file or create a new one, ensure that the `_quarto.yml` file is updated accordingly. Push these changes to GitHub to automatically deploy updates.
   - Version control: Keep track of your content changes by writing clear commit messages. This makes it easy to refer back to previous versions if needed.
2. **Troubleshooting Common Issues:**
   - Build failures: Sometimes, your GitHub Actions may fail due to misconfigurations or syntax errors (e.g., in the `_quarto.yml` file). GitHub provides detailed logs for failed builds in the Actions tab, which can help pinpoint the issue.
   - Rendering issues: If the published site doesn't display content as expected, double-check that your .qmd files are properly linked in the `_quarto.yml` file and that all references (images, links, etc.) are correctly specified.
3. **Collaboration and Feedback Collaborative development:**
   - If you're working with a team on the resources, consider using GitHub's collaborative features like Pull Requests to review changes before they're merged into the main branch. This can help catch errors early and maintain consistency across the project.
4. Ensure that your project dependencies (tracked by `renv.lock`) are up to date by regularly running `renv::snapshot()` after installing new R packages. This ensures that your environment remains consistent across different systems.

2 Building from an existing GitHub Repository

These resources are intended for users who are contributing to an existing project hosted on GitHub and want to collaborate on building or updating Quarto-based materials. You will learn how to clone the project, make changes locally, and collaborate with others via GitHub.

Hopefully you have previously run the steps in the Starting Point and are set up with GitHub - if not make sure you follow the steps set out there.

2.1 Step 1: Clone the Existing GitHub Repository

The first step to collaborating is to clone the GitHub repository to your local machine. This creates a copy of the project on your computer, allowing you to work with the files locally.

1. Locate the repository: Visit the GitHub page for the repository you wish to work on. You should see a green Code button on the repository's main page.
2. Copy the repository URL: Click the Code button and copy the HTTPS URL. For example if you wanted to clone and work on this module:

```
https://github.com/PATH-Global-Health/quarto-module-dev.git
```

3. Clone the repository in RStudio
    - Open **RStudio**.
    - Go to **File** → **New Project** → **Version Control** → **Git**.
    - Paste the repository URL you copied earlier into the dialog box.
    - Choose where to store the project locally on your computer.
    - Click **Create Project**.

This will create a local copy of the repository that you can work on. You'll see the project files in the RStudio Files pane.

2.2 Step 2: Familiarize YourSelf with the Project

Before making any changes, it's important to understand the structure of the project and its current status.

- **Explore the `_quarto.yml` file**: This file defines the structure of the Quarto book (or website) and the order of chapters. Take a look to understand how the chapters are organized.
- **Review existing content**: Open and read through some of the `.qmd` files (Quarto markdown) to get a sense of the writing style, structure, and the materials that have already been added.

2.3 Step 3: Set up Your Local Environment

To work on this project, you may need to install the same R packages that the project uses. This is often managed through **renv**.

1. **Activate `renv`**: If the project is using `renv`, you will see an `renv.lock` file in the repository. To install the necessary packages, run the following command in the RStudio console `renv::restore()` This will install all the packages listed in the `renv.lock` file, ensuring your environment matches the one used by others working on the project.
2. If additional software is needed (like TinyTeX for PDF output) install these.

2.4 Step 4 Make Changes

Now that your environment is set up, you can start working on the project. Whether you're fixing a bug, adding content, or improving the formatting, here's how you can proceed:

1. If you are going to be the sole user of the repository now you don't need to create a new branch - you can skip the next two steps.
2. **Create a new branch**: It's good practice to create a new branch for your changes so that you don't modify the main branch directly. To create a branch, run the following command in the **Terminal** tab in RStudio:

```
git checkout -b new-feature-branch
```

Replace `new-feature-branch` with a name that describes the changes you are making or perhaps your name for example.

3. **Edit or add files**: Make your changes to the `.qmd` files or other assets in the repository. For example, you might want to add a new chapter or modify existing content. As you edit files, save your changes in RStudio.

This step will be very context depenent on the type of materials we want to add to our modules so I have outlined some common steps across materials that have been built so far.

The Quarto Guide is going to be super useful https://quarto.org/docs/guide/ especially if you're new to working in Markdown - I won't replicate the resources here and recommend using the Quarto guide as a reference point when writing in Quarto.

There is always the option in Quarto to use the `visual` editor - this simplifies the process of writing and formatting content in markdown formats and allows users to focus on writing without needing to remember markdown syntax, making it easier for beginners and those unfamiliar with code-based formatting. The Visual Editor provides an interface similar to popular word processors like Microsoft Word or Google Docs.

The Visual Editor includes toolbars with buttons for common formatting options such as:

- Bold and italic text
- Creating headings and subheadings
- Adding bullet points or numbered lists
- Inserting links, images, and tables
- Adding callouts for notes, warnings, or tips
- You can still insert and edit code chunks directly in the Visual Editor. These code chunks are executed as usual, and their outputs (tables, figures, etc.) are displayed inline, making it easier to integrate code into your documents.

To use the Visual Editor in RStudio or other supported IDEs:

- Open any Quarto (`.qmd`) or markdown file in your project.
- Click the **"Visual"** button in the top right corner of the editor pane to toggle between the standard markdown editor and the Visual Editor.

2.4.1 Understanding the `_quarto.yml` file

The `_quarto.yml` file is a configuration file that controls the structure and appearance of your Quarto project, such as the website's title, menu, and theme.

For a **Quarto book**, the `_quarto.yml` file specifies the structure, appearance, and behavior of the book. Below is an example of what the structure might look like and what you will see when you open this initially in your project.

```yaml
project:
  type: book

book:
  title: "My Quarto Book"
  author: "Author Name"
  chapters:
    - index.qmd
    - introduction.qmd
    - chapter1.qmd
    - chapter2.qmd

format:
  html:
    theme: cosmo
    toc: true
    number-sections: true
    css: styles.css
```

We want to start by modifying the `title` and `author` fields under `book` to what is relevant for the module (we can just delete the author line as it's not needed here)

We can also add a `subtitle` that's nice to include in all our topics like this:

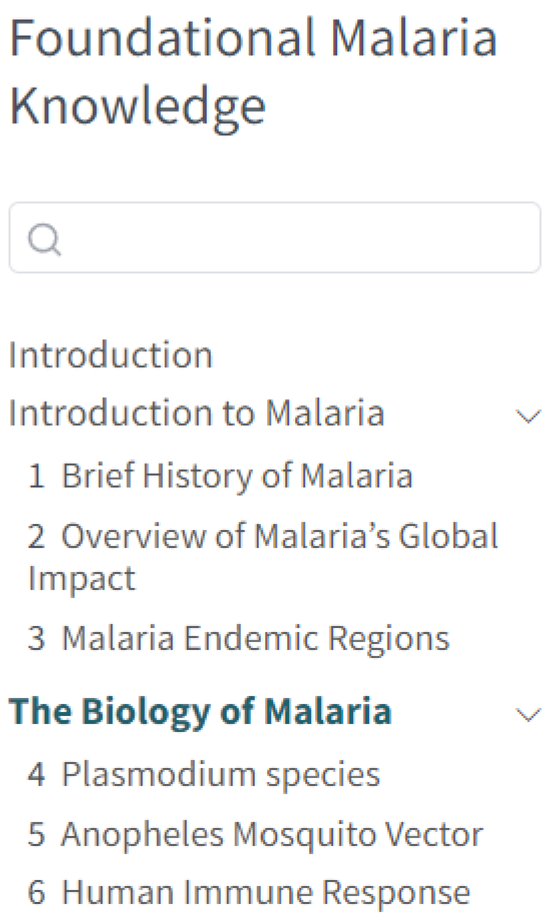
```yaml
book:
  title: Foundational Malaria Knowledge
  subtitle: MACEPA Data Fellowship - Training Materials
```

The chapters section defines the order of the chapters in your book. It links to the `.qmd` files that will become the different sections of the book:

```yaml
book:
    chapters:
        - index.qmd           # Main page or introduction
        - introduction.qmd    # First chapter
        - chapter1.qmd        # Subsequent chapters
        - chapter2.qmd
```

You can add or remove chapter files as needed. Just make sure each file you list here exists in the project.

For some of our modules we have several topics within a module and sub topics within them so we can use the `part` as well as `chapters` to achieve something like this:


Foundational Malaria Knowledge

Introduction

Introduction to Malaria

1 Brief History of Malaria

2 Overview of Malaria's Global Impact

3 Malaria Endemic Regions

The Biology of Malaria

4 Plasmodium species

5 Anopheles Mosquito Vector

6 Human Immune Response

To do so you can set the `_quarto.yml` to something this:

```yaml
chapters:
- index.qmd
- part: topic1-intro.qmd
     chapters:
     - topic1a-history-malaria.qmd
     - topic1b-global-impact.qmd
     - topic1c-endemicity.qmd
```

```
  - part: topic2-intro.qmd
        chapters:
        - topic2a-plasmodium-spp.qmd
        - topic2b-lifecycle.qmd
        - topic2c-immunity.qmd
   - part: topic3-intro.qmd
        chapters:
        - topic3a-anopheles-trans.qmd
        - topic3b-environment.qmd
        - topic3c-human-behaviour.qmd
```

In each of these instances the title that is in the `.qmd` file becomes the heading that is shown on the final website as shown in the image above.

Note

Once you start adding `.qmd` files with module content make sure to replace the defualt files and include these in the `_quarto.yml` to ensure the modules are displayed on the site. Often I start adding content to the templates and renaming the files before creating new `.qmd` files

2.4.2 Adding custom styling

We've already created custom PATH theming and I've saved all the necessary files in the `data-fellows box folder` under 'quarto-theming' Data fellowship program planning → Technical Content Organization → quarto-theming

Copy all of these into your root folder and add the following to your `_quarto.yml` file

```
format:
  html:
    theme:
    - cosmo
    - custom.scss
    template-partials: title-block.html
    css: include/webex.css
    include-after-body: include/webex.js
    embed-resources: true
  pdf:
    documentclass: scrreprt
editor: visual
```

2.4.3 Embedding PDFs

If you have a slide deck to host on the website we first need to save this as a pdf and ensure it it then saved in the same root directory of our Quarto project.

You can use the following syntax to embed the pdfs inside a `.qmd` file replacing the file name with the associated file name of your file.

```
<iframe src="file-name.pdf" width="800" height="600">

</iframe>
```

2.4.4 Embedding YouTube Videos

If you want to include videos hosted on YouTube an easy was to achieve this is with the following syntax, replacing the link with that to the video of interest:

```
https://www.youtube.com/embed/bJ6nS-I-HiM
```

2.4.5 Interative Quizzes

You can use the `webexercises` package to produce quizes and is what I used in the Foundational Epi module

The documentation is a good place to start here and the source code in the Foundational Epi Module repository if you want to develop something similar!

2.5 Step 5: Commit and Push your Changes - main branch

If your workflow involves making changes directly to the **main** branch, follow these steps to commit and push your changes.

1. **Stage your changes**: First, you need to add the modified files to the staging area. To see which files have been modified, run:

```
    git status
```

This command will list the files that have been added, changed, or deleted. Once you're ready to stage them, run:

```
    git add .
```

The . adds all the changed files to the staging area. You can also stage individual files by replacing . with the filename.

2. **Commit your changes**: After staging your files, you'll need to commit them to the repository with a descriptive message that explains the changes you made. Run the following command:

```
git commit -m "Your descriptive commit message here"
```

Make sure the commit message clearly states what changes you've made

3. **Push your changes**: Once you've committed your changes, push them to the **main** branch on GitHub by running:

```
    git push origin main
```

This command will send your local changes to the main branch of the remote GitHub repository. After pushing, your changes will immediately be reflected in the repository and if a [GitHub Action]Section 1.5 to publish has been set up this will automatically be reflected in the online site.

Important Notes:

- **Working on the main branch**: When working directly on the main branch, it's important to ensure that your changes won't disrupt the main project. This workflow is typically used for smaller updates or when a project does not follow a branch-based development model.
- **Keep the main branch up to date**: Before making any changes, ensure your local copy of the main branch is up to date by running: `git pull origin main`
- **Collaboration caution**: If other team members are working on the same repository, communicate to avoid conflicts. Making significant changes directly on the main branch could cause merge conflicts if others are working simultaneously.

2.6 Step 5: Commit and Push Your Changes - new branch

Once you're happy with your changes, it's time to commit and push them back to GitHub.

1. **Stage your changes**: In the `Terminal`, check which files have been modified: `git status`
2. **Add the changes to staging:** If the output looks correct, add your changes to the staging area `git add .`
3. **Commit your changes:** Write a meaningful commit message describing what you've changed: `git commit -m "Added a new chapter on…"`
4. **Push your changes:** Push your branch to GitHub `git push origin new-feature-branch` (change the name to the branch name you are working on)

2.7 Step 6: Create a Pull Request

Now that your changes are on GitHub, you need to create a **Pull Request (PR)** so that the repository maintainers can review and merge your work into the main branch.

1. **Navigate to the repository on GitHub**: Once there, GitHub will automatically suggest that you create a pull request for your new branch.
2. **Open the Pull Request**: Click the **Compare & pull request** button. In the PR description, explain what changes you've made and why.
3. **Submit the Pull Request**: Click **Create pull request**.
4. Once the PR is approved, it can be merged into the main branch by a maintainer.

2.8 Step 7: Keeping Your Fork or Local Repository Up-to-Date

If you are working on a **forked repository** or need to sync your local repository with the main branch, follow these steps:

1. **Pull changes from the main repository**: Run the following commands to fetch and merge changes from the upstream (original) repository:

```
git fetch origin
git pull origin main
```

2. **Resolve any conflicts**: If there are any merge conflicts, RStudio will show you the conflicting files, and you can resolve them before pushing the changes.