# Methods in Open Science

## Version Control

Image from NASA: https://commons.wikimedia.org/wiki/File:Earth_Western_Hemisphere_transparent_background.png#filelinks

In this module, we will explore the concept and application of version control as it applies to research and code development projects. We will also discuss the creation of README files using Markdown and documents using Quarto.

# Version Control


Version control software allows you to keep track of changes made to a directory (such as a folder on your computer) or code/files associated with a project, a website, documentation, and/or a research project. A history of changes is tracked so that it is possible to revert back to older versions of the file(s) or data. It is also possible to create copies, experiment with the content or code, then merge the results back into the main project.

Further, version control can allow for multiple editors of files and can keep track of who made changes and allow all copies of the files/data associated with the project to be synchronized across different devices and cloud repositories.

In short, version control software allows for better management of directories, files, and data. For example, when writing a document, it is common to save changes by creating multiple versions of the file as it is edited (e.g., document1, document2, document_revision, document_final, document_final2, etc.). Version control provides a much cleaner means to track changes and, for large projects with or without multiple contributors, can be a cleaner way to work.

## Version Control Uses

- Keep track of changes
- Allow collaboration
- Revert to older version
- Save to a remote location
- Experiment with changes outside of the main branch
- Integrate or reject changes made by others

4

This slide provides examples of some of the uses of version control software. Version control allows for tracking changes, collaboration, reverting to older versions of files/data, saving and backing up files to a remote location, experimentation on a copy of the data that can then be merged back to the main copy if desired, and integration or rejection of changes made by others.

## Git

- Version control software
- Free and open-source
- Scalable, small footprint, and fast
- Available on all operating systems
- Command line interface and GUI
- For Windows: Git + Bash emulator

https://git-scm.com/

5

There are different software tools that allow for version control. Here, we will discuss Git. Git is currently free and open-source. It is generally scalable from small projects to large projects with many files and/or editors. It generally has a small footprint in regard to the storage used to keep track of changes and versions. It is available for use on all operating systems and can be accessed within the Bash UNIX shell. On Windows, you can install Git along with a Bash emulator. The link on this page can be used to download and install Git. There is also a GUI for Git; however, we will not explore that option here.

## Terminology

Commit = stores the current contents of the index, a set of changes, in a new commit along with a message from the user describing the changes
Index = cache where changes are stored before they are committed
Stash = another cache where changes can be stored until they are ready to be added to the index and an eventual commit
Working tree = the current branch in your workspace
Branch = a set of linked version histories or commits
Master = the default name for the first branch or main branch in the version history
Merge = joining two or more commit histories
HEAD = a pointer to the most recent commit on the current branch (i.e., your current location within the version history)
Tracked = files either in the index cache or not yet added to it
Workspace = local copy of a Git repository
Local repository = another term for where you keep your copy of a Git repository on your local machine
Remote repository = a secondary copy of a Git repository where you push changes for collaboration or backup
Origin = the default name for a remote repository

https://opensource.com/article/19/2/git-terminology

6

This slide summarizes some key terminology related to version control and Git in particular. The text was derived from the link provided on the slide. We will now talk through some of the key terminology used.

A commit represents a save point in which the state of the files are recorded. Changes made to the files are cached to the index when running the git add command. When a commit is made, the changes stored in the index are recorded as part of the version history. Again, think of this as a snapshot of the state of the files. Note that it is also possible to save changes without committing by using the stash cache. These changes can later be added to the index then applied with a commit. The working tree represents the current status of your workspace. This allows you to see what changes have been made and what updates will be applied at the next commit based on what has been added to the index cache.

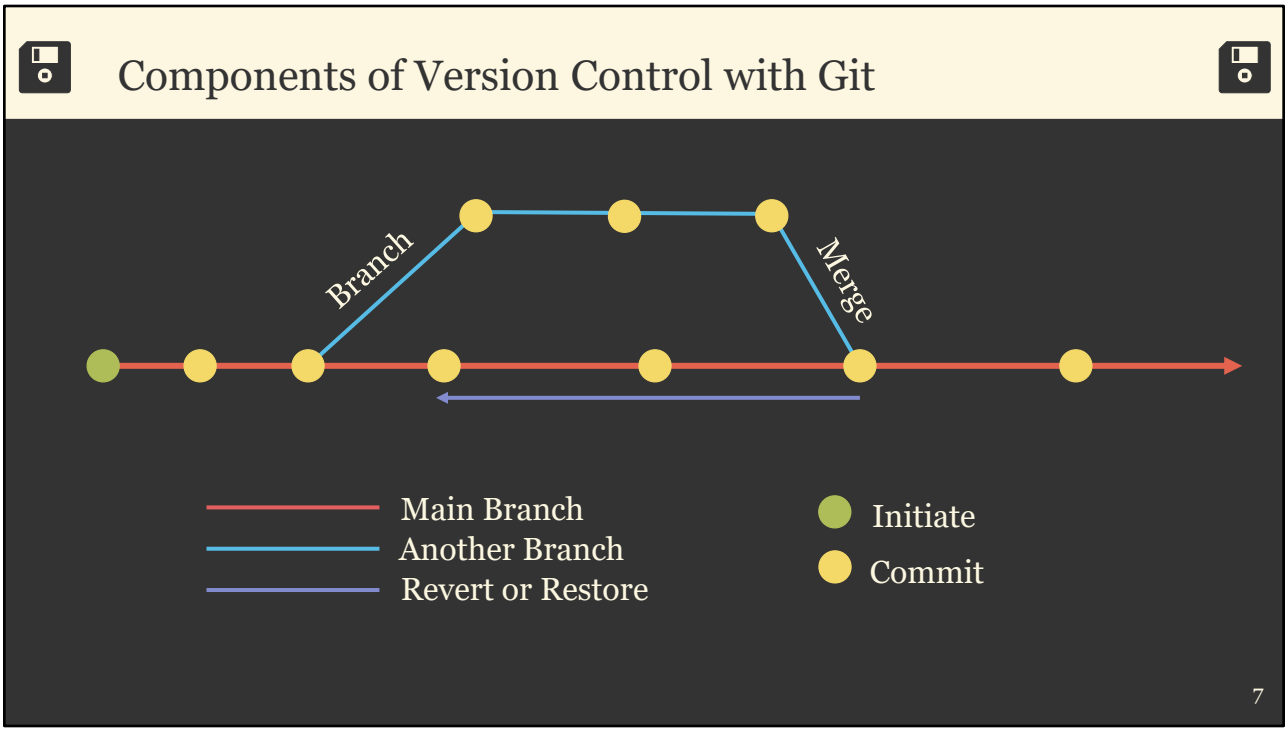
Note that in a project or directory it is not necessary to track all files. For example, you may not track changes made to data files that are stored in the directory but are just being used to test code or perform experiments. When changes to a file are monitored, these files are said to be tracked. If changes are not monitored, then these files are untracked.

In order to allow for changes to be made to the files or project outside of the

main version, branches can be generated. The branch will make a copy of the state of the files relative to a specific commit. Note that the main branch is generally termed the master. Once a branch is made outside of the master branch, it is possible to make changes to the files and save commits that do not impact the master branch. For example, this might be done to add new functionality to a software tool. Since major changes are being made, the developer can perform experiments and make commits outside of the main branch. If it is later decided that these changes should be integrated into the main branch, then the branch can be merged with the main branch. There are options for how the two version histories of merged branches are consolidated.

The HEAD is the term use for the most recent commit on the branch that is currently active.

It is also possible to link your local project to a remote version. A local copy of the project, files, or directory is called a workspace or local repository. The copy saved remotely is known as the remote repository. The default name applied to a remote repository to which a local repository is associated is the origin.




This diagram provides a description of the concepts of commits, the main branch, a branch, and reverting back or restoring to a prior commit.

The green dot represents the state of the files when the version control process is first initiated, or this represents the state of the files, or the initial reference point that all later changes are referenced to. You can also think of this as a baseline. All yellow dots represent commits where changes are documented for all files that are being tracked. Each commit represents a set of changes and generally will have a message associated with it that is used to note the changes made for later interpretability.

The red line represents the main branch. In contrast, the blue line represents another branch. This branch was initiated at a specific commit from the main branch. All subsequent commits on this branch do not impact the main branch until it is merged back in with the main branch. At this point, changes made in the main branch and additional branch must be consolidated or reconciled in the form of a new commit.

It is also possible to revert to a prior commit to remove or undo existing changes. For example, you could revert the main branch to a prior commit in order to undo the impact of merging the changes from the other branch. Since even reverts are tracked, you could also later restore these changes if desired.

## Getting and Creating Projects

- $ git init = initialize a repository
- $ git clone = clone repository to a new directory


https://git-scm.com/docs

8

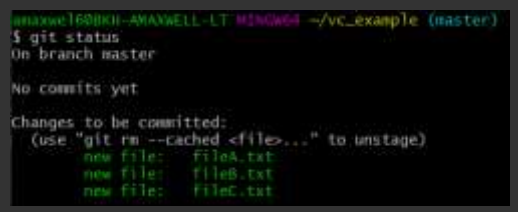
We will now explore some common commands used to track changes and make commits. The syntax shown here represents Git commands implemented using Bash. Note that desktop software tools are available to use Git in a GUI environment. However, we will only explore the CLI here.

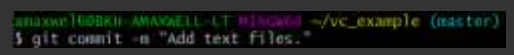
First, to initiate a folder or directory to track, you will need to navigate to this folder then use the git init command to initiate the first saved state. This will create a hidden .git folder within the directory. You may have to turn on hidden files in your file explorer to see this folder. The active directory can be changed from the command line or using Bash via the cd (change directory) command.

It is also possible to clone an existing directory or folder in which changes are being tracked to a new folder.

## Taking a Snapshot and Keep Track of Changes

- $ git add = add files and changes to the index
- $ git status = show status of working tree
- $ git diff = compare two commits or commit and current working tree (changes that have been added to the index but not yet committed)
- $ git commit = record change to the repository (snapshot of version history)
- $ git restore = restore or revert to state at a specified commit
- $ git rm = remove files from working tree and from the index
- $ git reset = reset current HEAD to the specified state (a specific commit on a specific branch)


https://git-scm.com/docs

9

To add changes made since the last commit to the index, you can use the git add command. You can check the status of the working tree using git status. This will list out all changes that will be tracked at the next commit, or all changes that have been added to the index using add. Once a commit is made, all changes will be tracked and calling git status should indicate that no changes are in the index.

Again, to actually create a save point, you use git commit. This command will also require that a message be provided (e.g., -m “Fixed the for loop error”), which can aid in helping you document what changes were made. I find that git add, git status, and git commit are the commands I use most often.

The git diff command is used to compare the difference between two commits or a commit and the current working tree (i.e., changes that have not been committed yet). The git restore command is used to restore or revert all tracked files or specific tracked files to a prior state, as defined by a prior commit, while git rm will remove files from the working tree and index so that changes will not be or no longer be tracked. Lastly, git reset will reset the current HEAD, or most recent commit on the current branch, to a defined state or commit in the version history.

## Branching and Merging

- $ git branch = list, create, or delete branches
- $ git checkout = switch branches or restore working tree files (clear index)
- $ git switch = switch between branches
- $ git merge = merge two or more development histories or branches together


$ git branch 'Name of new branch'
$ git branch 'Name of branch to switch to'

https://git-scm.com/docs

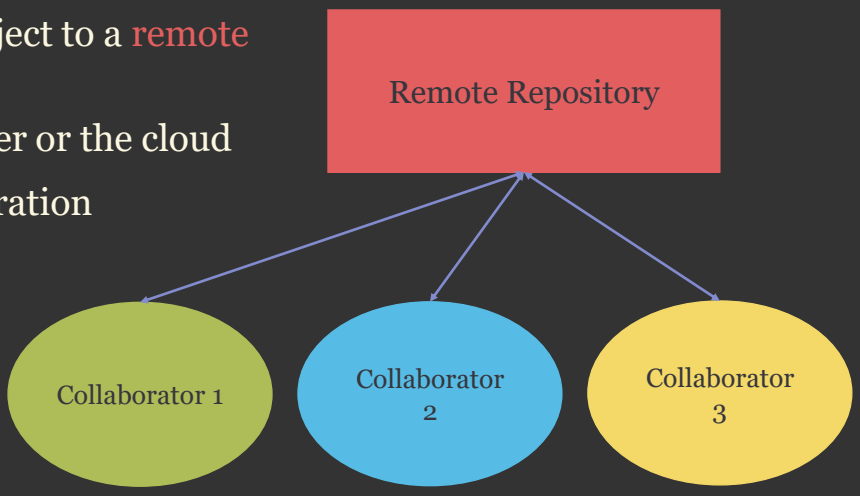
10

You can save a branch to work outside of the main branch using git branch. You can switch between branches or restore the working tree files to a prior state using git checkout while git switch is used to specifically switch between branches.

Lastly, git merge is used to merge two or more development histories together, such as merging changes made within a branch to the main branch. When a merge is made, changes between the branches must be reconciled to record a single state to the new commit.

## Remotes

- Link local git project to a remote copy
- Back up to a server or the cloud
- Allow for collaboration

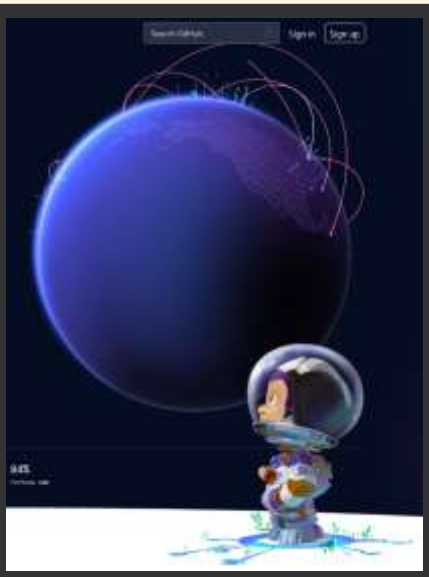



11

As already mentioned, it is possible to link a local repository to a remote repository to back up the project or allow collaboration. We will now explore how this process works using GitHub specifically.

## Using GitHub

- A git repository (repo) hosting system
- Manage repositories
- Cloud-based
- Free and paid account options
- Add collaborators



https://github.com/

12

First, Git and GitHub are not the same thing. Git is a version control software tool, as discussed above, while GitHub is a hosting system/service that can be used to establish a remote repository. You do not need to use Git to use GitHub, and you do not need to use GitHub to use Git locally. Note that there are other repository hosting services. We will discuss GitHub here because it is commonly used and has free options.

GitHub is used to manage repositories, can be linked to local repositories to serve as a remote repository, supports collaboration, is cloud-based, and offers both free and paid accounts. Although purchasing a paid account does add functionality, I have found that the free version is adequate for my work.

GitHub is cloud-based, so your data and project are backed up on a remote, secure server.

## Using Remotes

- fetch = retrieve latest commits from the remote; changes can then be compared using diff then merged with local copy using merge
- pull = retrieve latest commits from the remote and merge with local copy (i.e., update local copy); this combines fetch and merge
- push = send local commits and changes to the remote repository

$ git remote add origin 'URL to remote'

$ git pull 'remote name' 'branch name'

$ git push 'remote name' 'branch name'

$ git push  origin master


13

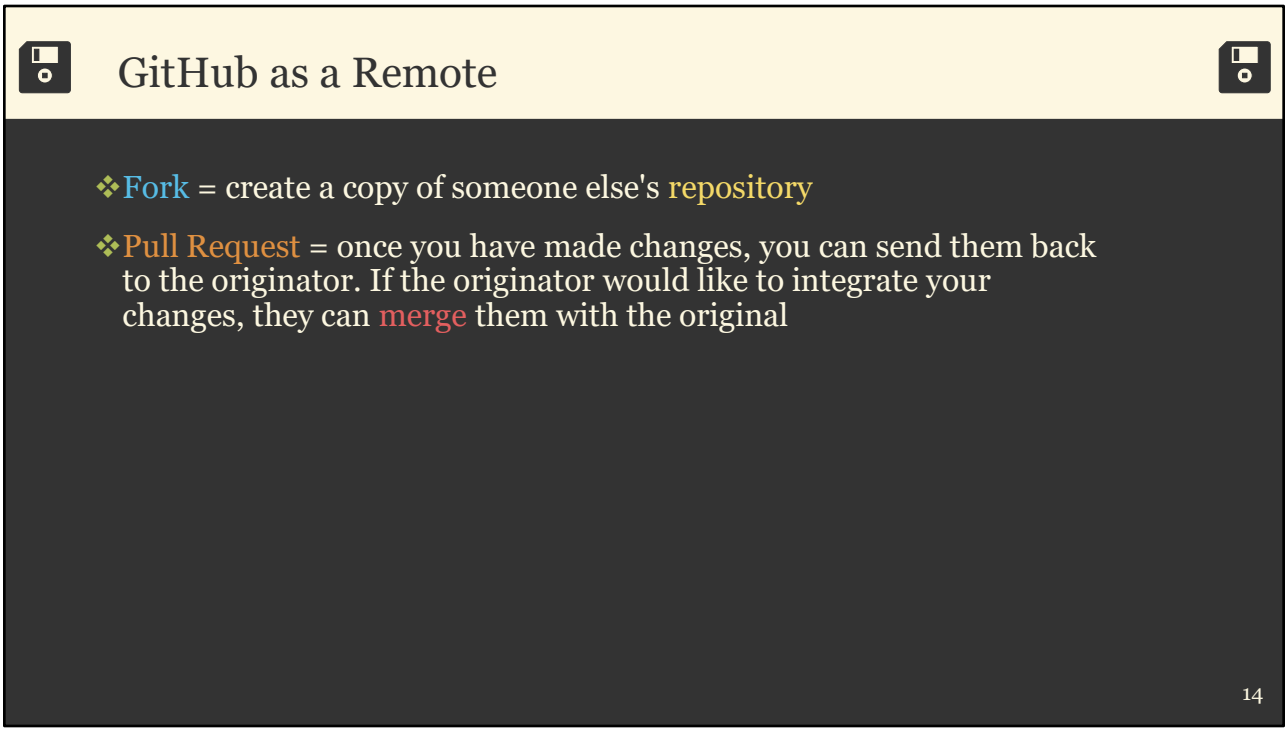
In order to associate your local repository with a remote repository hosted on GitHub, you must first add a remote. This can be accomplished using the git remote add command. The default name for a remote repository is origin, and you can connect to it using a unique URL, which GitHub will generate for you.

Once a connection is made, you can pull content from the remote repository to the local repository using the git pull command. This will require you to provide the name of the branch on the remote repository to pull from and the name of the branch on the local repository that you wish to add a commit to. Again, it is common to use the name origin as the name of the remote repository. So, git pull origin master would commit the content of the remote origin branch to the local main branch.

Note that git pull combines the functions of git fetch and git merge: it will both pull changes from the remote and then merge them with the specified branch of the local repository.

In order to push changes up to the remote repository, you can use the git push command. Again, you will need to provide the name of the remote repository branch that you want to save to and the local repository branch that you want to copy to the remote. Generally, you will want to add all changes to the index (git add) then commit the changes (git commit –m "Your message here")

before pushing to the remote so that the most recent changes or commit made for the local repository is pushed to the remote.



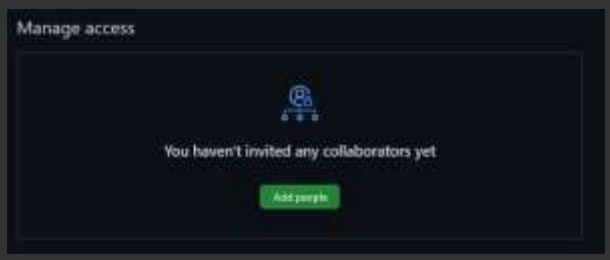

If you would like to make a copy of another developer's or scientist's GitHub repo, you can create a fork. A fork generates a copy of that repo under your account. You can then work with the files, make changes, and save commits. You can even make a local copy of the repository to work with.

If you would like the originator of the repo to consider integrating in your changes or some of your changes, you can then make a pull request. The originator can then review your changes and merge some of them or all of them into the original copy.

Forking and pull requests are a key component of how GitHub supports collaboration. Since changes made in your forked copy do not impact the original repo, the original is protected from edits, and the owner of the repo maintains control of the project. However, the owner can then implement changes made by others by reviewing pull requests and merging changes.

## GitHub as a Remote

- Merge pull requests
- Invite collaborators
- Add or remove collaborators
- Make repo public or private
- Delete repo




15

As the owner of a remote repo on GitHub, you can merge changes associated with pull request submitted by others, invite collaborators so that they can work with and make commits to the original repo (note that this is different from forking a repo since you are allowing others to work with and change your original copy), add or remove collaborators, make a repo publicly viewable or private, and delete your repos and associated content.

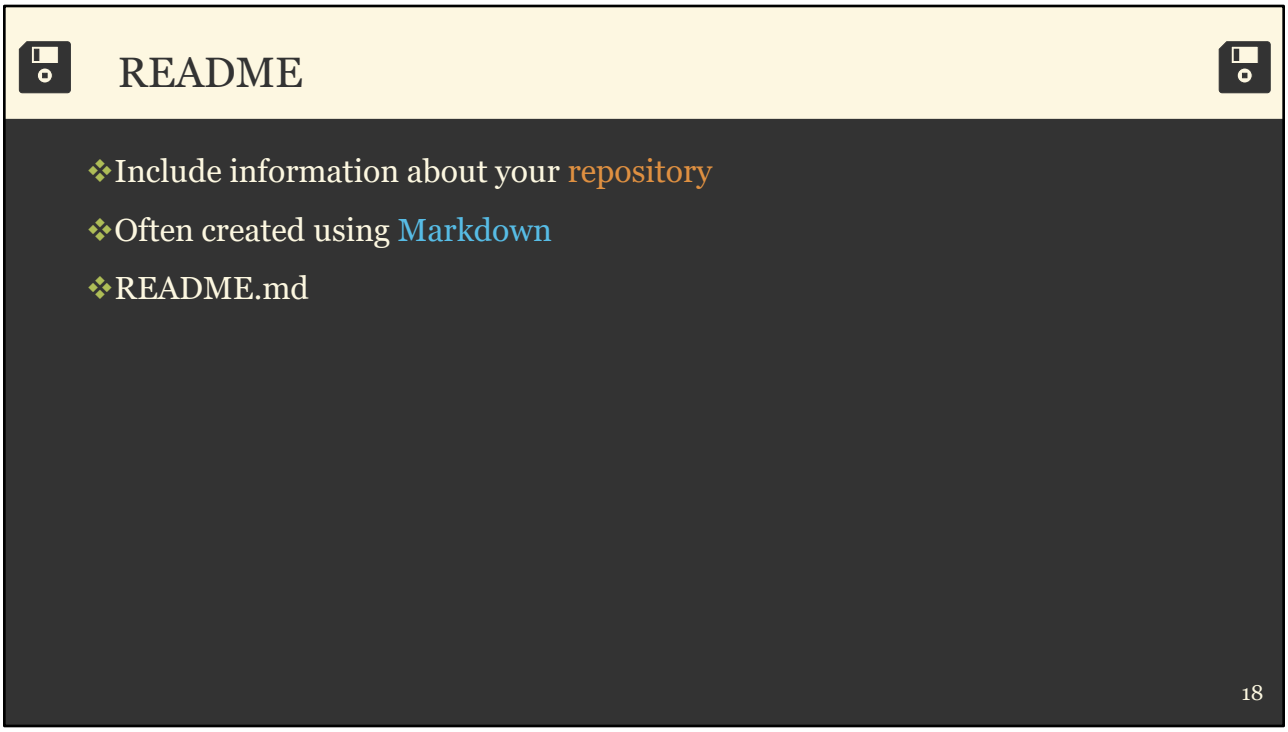
## Summary of Key Points

- Version control is used to maintain an edit history.
- These methods are valuable for experimenting with changes that you may or may not want to merge back into the main project.
- These methods are valuable for collaborating with others.
- Local repositories can be linked to remote repositories for backup and collaboration.
- Git is one technology used for version control.
- GitHub is an example of a remote repository service.

# Documentation and Markdown

17

We will now explore methods used to document code with a focus on GitHub repos.



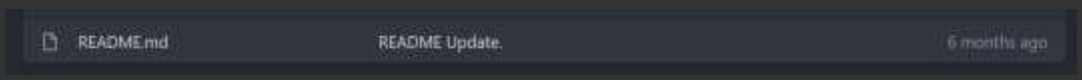

README files are generally used to explain data, code, and/or deliverables. For example, if you create a dataset for a client, you may ship the dataset in a compressed folder and include a README file to explain the data and the naming conventions. This README file is commonly a plain text file (TXT). However, you could also include documentation as a text document (e.g., Word doc), PDF, or HTML webpage. TXT files are commonly used because they can be viewed on most systems.

On GitHub, a README file should be included in the main directory of the repository. The content of this file will define what is displayed on the landing page for the remote repo. In order to generate well formatted landing pages, as opposed to just loading plain text, it is common to use a markup language that can be rendered with defined formatting. This is commonly accomplished using Markdown. Markdown is a mark up language (similar to HTML used on websites) that allows text typed in a plain text document to be rendered with formatting. Markdown is generally very simple and easy to use.

Markdown is commonly stored in a .md file, which is just a plain text file. So, GitHub repos will commonly have a README.md file in the main directory.

## Markdown

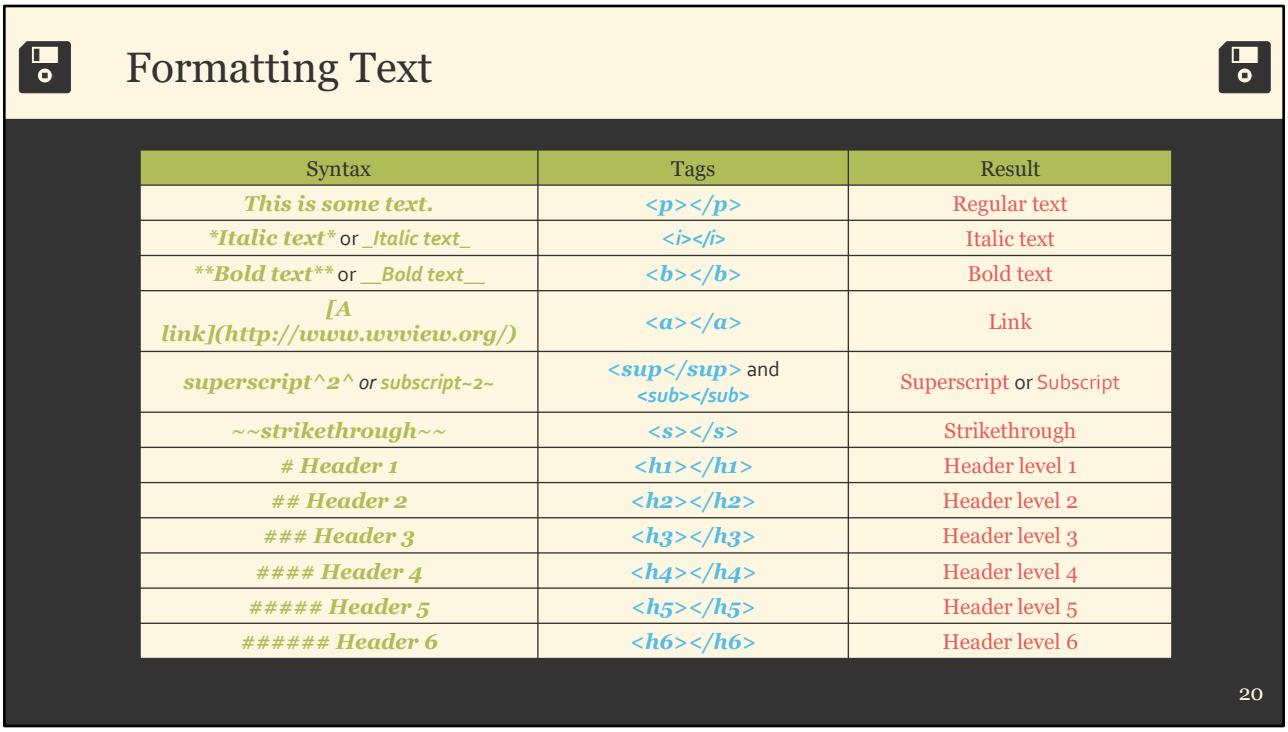
- Create README files
- Create documents, reports, and presentations
- Share your results for reproduceable science
- Create demos
- Create webpages


19

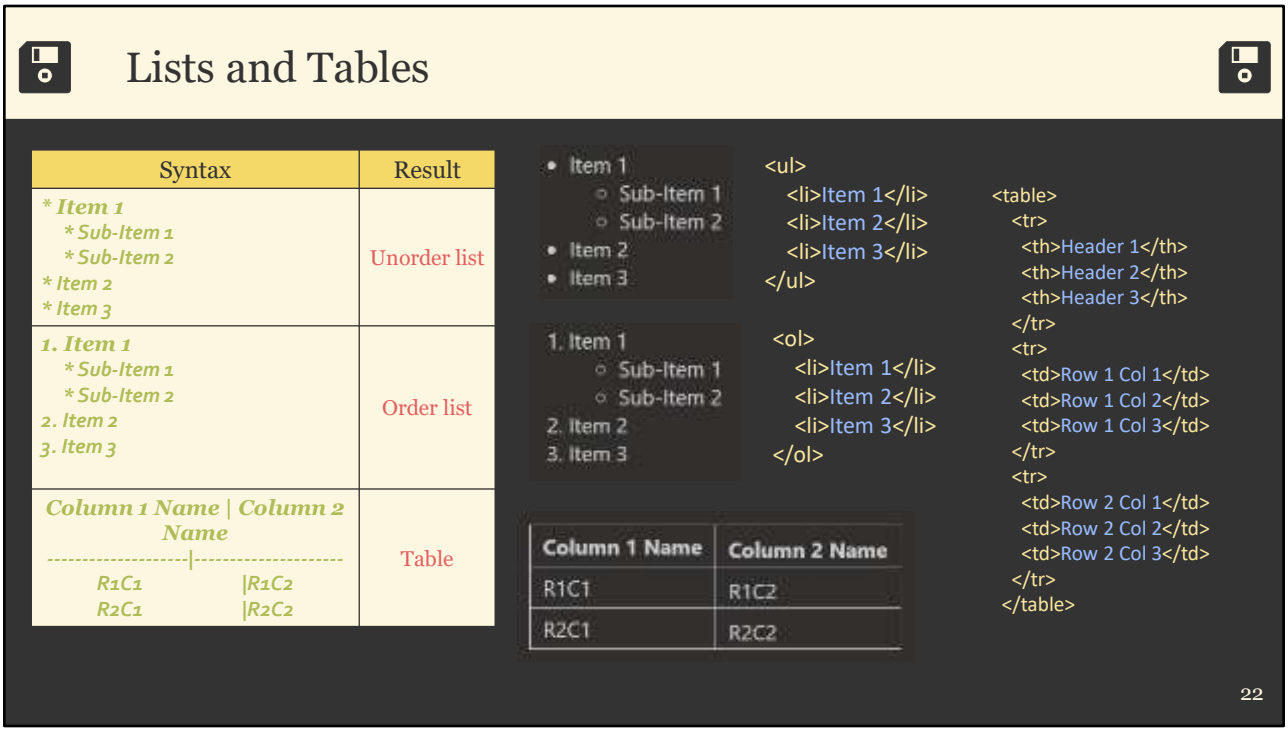
Markdown has many uses. Again, it is used to create README file in GitHub. See the example GitHub repo file structure in the provided image. However, Markdown can also be used to render documents, reports, presentations, demos, and webpages.

In this course, we will use Markdown to create the README files associated with GitHub repos. We will also use Markdown to format text within Jupyter Notebooks and Quarto files.




Text formatting is defined using special characters. Please see the examples in the table for plain text, italicized text, bolded text, links, superscripts, subscripts, strikethroughs, and different levels of headers. It is also possible to use formatting tags, which are generally the same as the associated HTML tags. Generally, the Markdown syntax is more concise than using formatting tags.


This slide provides examples of Markdown syntax, associated formatting tags, and the resulting output. These examples were created within a cell within a Jupyter Notebook, which can accept, interpret, and render Markdown.




The table on this slide provides formatting examples for unordered lists, ordered lists, and tables. I have also provided examples of associated formatting tags. Note that the Markdown syntax is generally more concise. Also, note the space between the formatting symbols and the text. For example, there must be a space between the asterisk and the text in order to render the result as an item in an unordered list. Without a space, the result would be italicized text.

## Others

| Syntax | Tags | Result |
|---|---|---|
| *> This is a blockquote* | *<blockquote></blockquote>* | Blockquote |
| *`y = 2`* | *<code></code>* | Inline code |
| *---* | *<hr>* | Horizontal rule |
| *![Alt text](Link or URL)* | *<img>* | Image |

Note: it is not currently possible to add videos within GitHub README markdown. Can link to video using text or an image.

23

This slide provides examples for blockquotes, code, horizontal rules, and image links.

Code that is included within a Markdown cell will not be executed. Instead, the code will be rendered and formatted as code text as opposed to plain text. This is often used to render example code.

Links to images can be URLs, which point to where a photo or image is stored on a webpage or server, or links to local files using a folder path. If an image will be used on a GitHub repo landing page, it must be included in the repo as a file or hosted on a webpage or at an accessible URL and referenced using that URL. Links to files on your local machine will not work from GitHub.

Lastly, it is not currently possible to place a video on a GitHub landing page, such as a link to a YouTube video. Instead, an image or text could be linked to the video hosted on an external site. The link can then launch the video in a new tab.

## Rendering

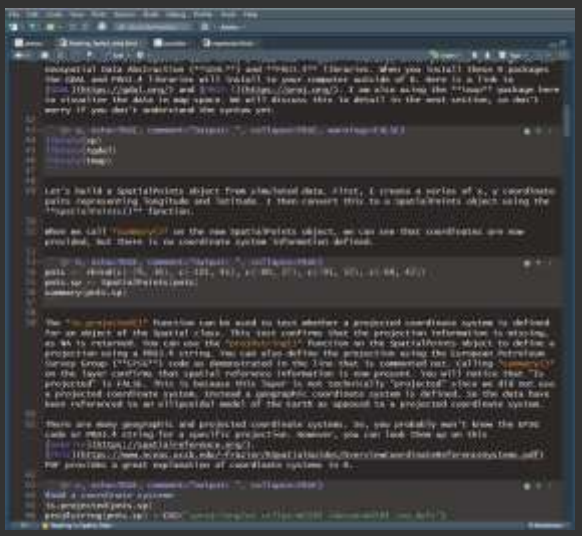
- Render to HTML
- Render to PDF


Outside of GitHub, Markdown can be used to create documents and presentations. It can be used within Jupyter Notebooks. These can be rendered into a variety of output including HTML webpages and PDF documents.

We will discuss using Markdown within Quarto documents in the next section.

## R Markdown

- Create documents, reports, and presentations
- Share your results for reproduceable science
- Create demos
- Create webpages



25

RStudio allows for experiments and code to be created and stored in R Markdown files (.Rmd). These files allow for Markdown to be used to create formatted text and the inclusion of code blocks that can be executed with the results displayed in the flow of the document. Markdown files can then be rendered to documents, PDFs, presentations, and/or websites.

Note that you may need to install some other software in order to render to some formats. For example, rendering to PDF may require LaTex.

## Summary of Key Points

- Markdown is a simple language used to generate formatted text and content.
- Notebooks can include Markdown cells along with code cells.
- Both Python and R make use of Markdown.
- Notebooks can be rendered to Markdown files.
- Markdown files can be rendered to PDF documents and HTML files.
- Markdown can be used to create README files for use in projects and GitHub repos.
- Markdown is a useful tool for generating results that are transparent and reproduceable.

# Quarto

## What is Quarto?

- Tool for scientific and technical publishing
- Works with multiple languages: Python, R, Julia, and Observable
- Works in multiple environments: Jupyter Notebooks, RStudio, VS Code
- Render a variety of outputs: articles, presentations, dashboards, websites, blogs, and books
- Render to a variety of formats: HTML, PDF, Word documents, and ePubs
- Well integrated with RStudio


https://quarto.org/

quarto

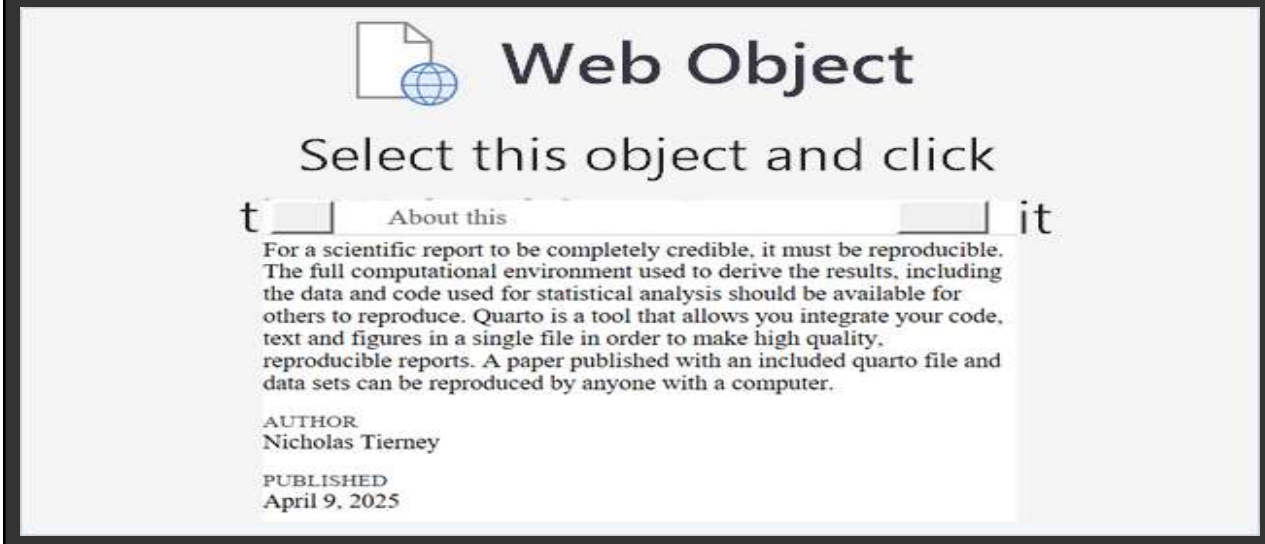
28

Quarto is a more recently introduced tool for generating scientific and technical publications. It is maintained and developed by the Posit company, formally RStudio. It can work with multiple languages (Python, R, Julia, and Observable) in multiple environments (Jupyter Notebooks, RStudio, VS Code, and Positron). It is well-integrated with RStudio.

You can generate a variety of output types including articles, presentations, interactive data dashboards, multi-page websites, blogs, and books. These products can be rendered to a variety of file formats including HTML, PDF, Word documents, and ePubs.

The link provided on this slides is for the main Quarto website.

## *Quarto for Scientists*

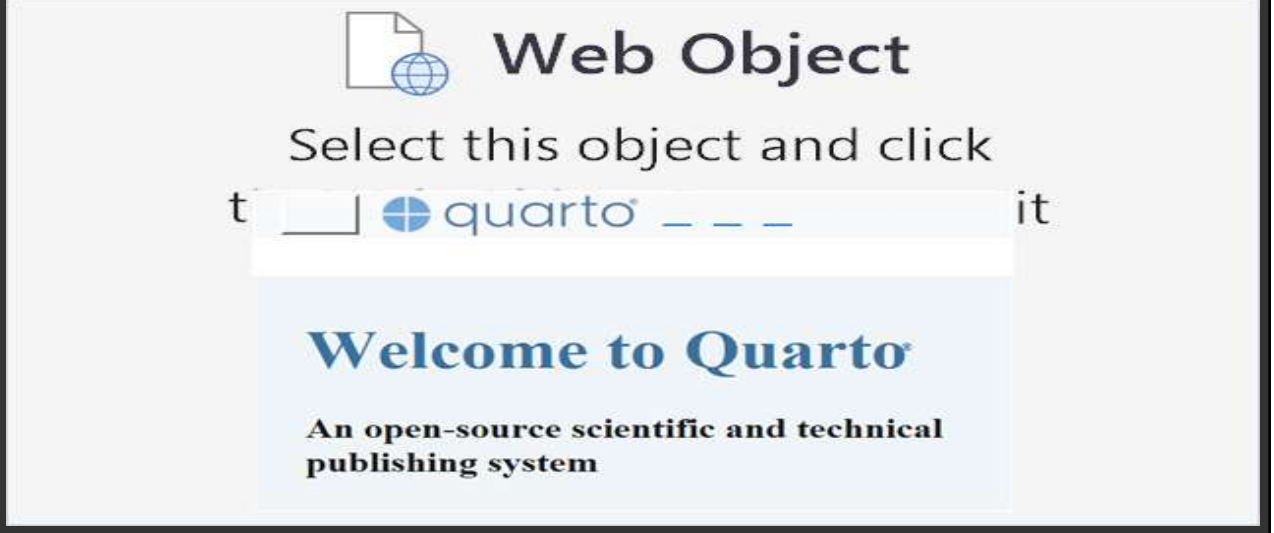



https://qmd4sci.njtierney.com/

29

The online text *Quarto for Scientists* is a great resource for researchers, analysts, and scientists that plan to adopt Quarto for documenting their work or creating training materials. I recommend exploring this text if you are interested in learning or adopting Quarto. It provides more detail that what is provided here.

## Quarto Documentation



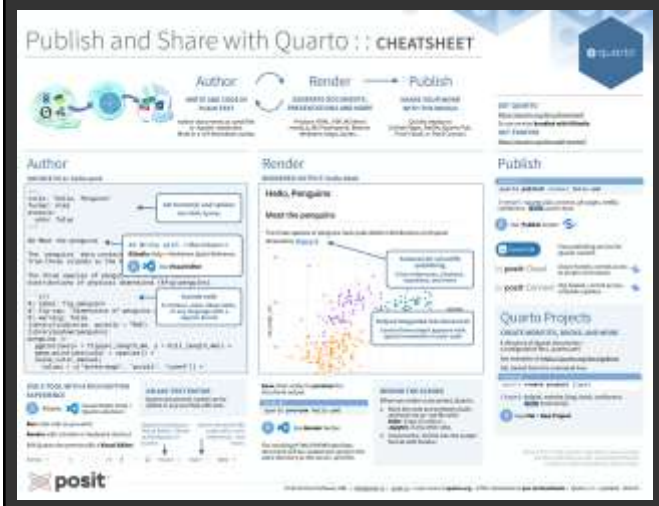

https://quarto.org/

30

Great documentation is also provided on the Quarto website.

## Cheat Sheet

https://rstudio.github.io/cheatsheets/html/quarto.html





31

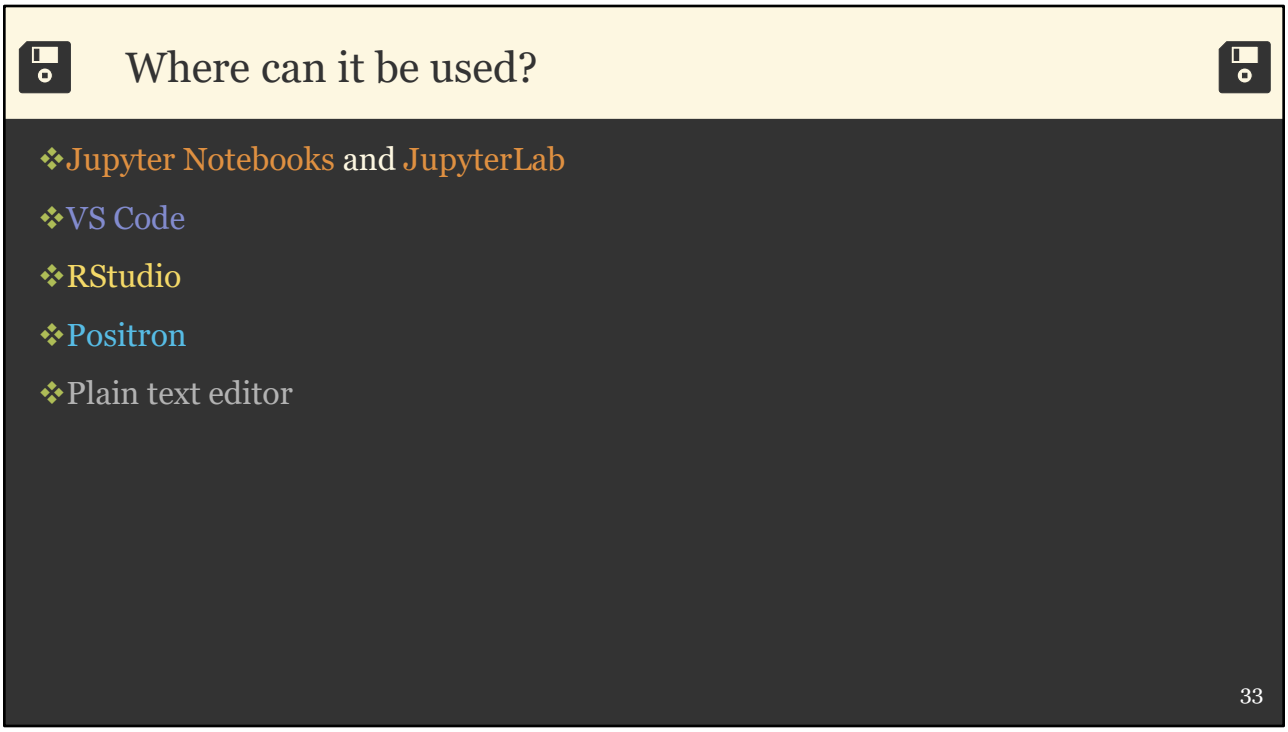
The page linked on this slide includes the Quarto cheat sheet, which is a great resource for getting started with Quarto.

## Why use Quarto?

- Easier to render to different formats
- Works with multiple languages
- Active development
- Very well documented

32

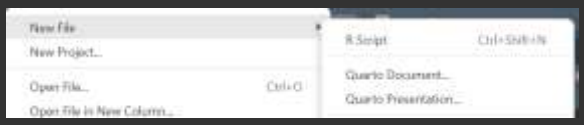
I have found Quarto to be very easy to use. It is very easy to render a variety of documents. All text is generated using Markdown, so it is easy to transition from R Markdown to Quarto. It is also well documented and still in active development, so new functionality is likely to be introduced.




It can be used within several tools or integrated development environments (IDEs) Including Jupyter Notebooks and JupyterLab, VS Code, RStudio, and Positron. You can also use plain text editors to create Quarto files.

## Setting up a document

- .qmd extension
- Define within YAML header
- Options and styles vary by output type




```
---
title: "Getting Started with geodl"
subtitle: "Geospatial deep learning with R, torch, and terra"
editor: visual
format:
  html:
    theme: sandstone
    fontsize: 1.1em
    linestretch: 1.7
    toc: true
---
```

https://quarto.org/docs/output-formats/html-basics.html

https://quarto.org/docs/output-formats/pdf-basics.html

https://quarto.org/docs/presentations/

https://quarto.org/docs/dashboards/

https://quarto.org/docs/websites/

https://quarto.org/docs/books/

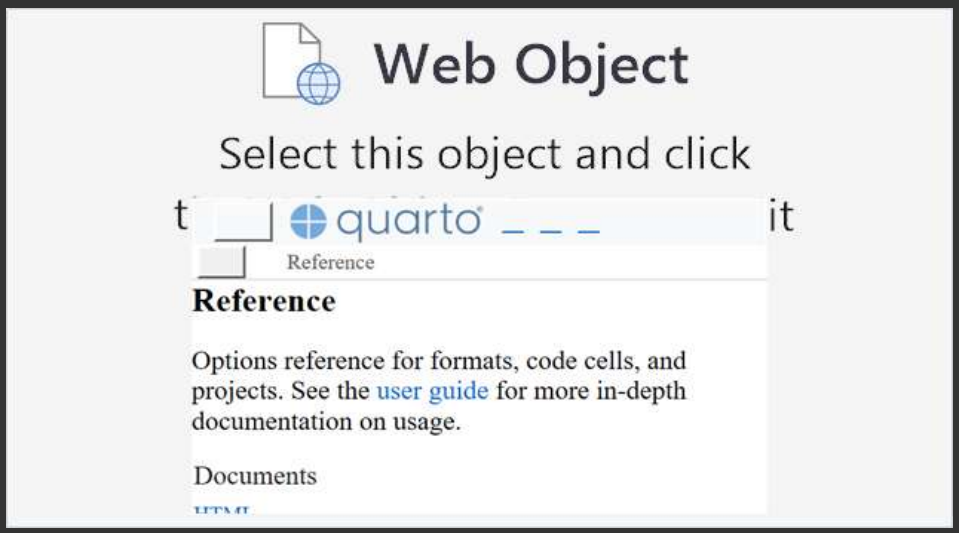
34

Quarto documents have a .qmd extension as opposed to a .md or .Rmd extension. The document and rendering options are specified in a YAML (YAML Ain't Markup Language) header. The options available depend on the type of product being rendered. The links on this page provide options for a variety of different output.

Note that Quarto files can be created in RStudio using File → New File → Quarto Document or Quarto Presentation.

## Types of Documents

- HTML
- PDF
- MS Word
- Open Office
- ePub
- Power Point
- Websites
- Books
- Manuscripts
- Etc.




https://quarto.org/docs/reference/

35

A variety of outputs can be generated. Take a look at the website referenced here, which provides separate pages for different types of documents that each provide guidance for how to build and configure a specific document type.

## HTML themes

https://quarto.org/docs/output-formats/html-themes.html

https://bootswatch.com/


```
---
title: "Getting started with geodl"
subtitle: "Geospatial deep learning with R, torch, and terra"
editor: visual
format:
  html:
    theme: sandstone
    fontsize: 1.1em
    linestretch: 1.7
    toc: true
---
```

36

The link on this page shows how to use different HTML themes when rendering HTML webpages. These themes are defined by Bootswatch. The screen capture included on the slide shows a webpage rendered from Quarto that uses the “Sandstone” Bootswatch theme.

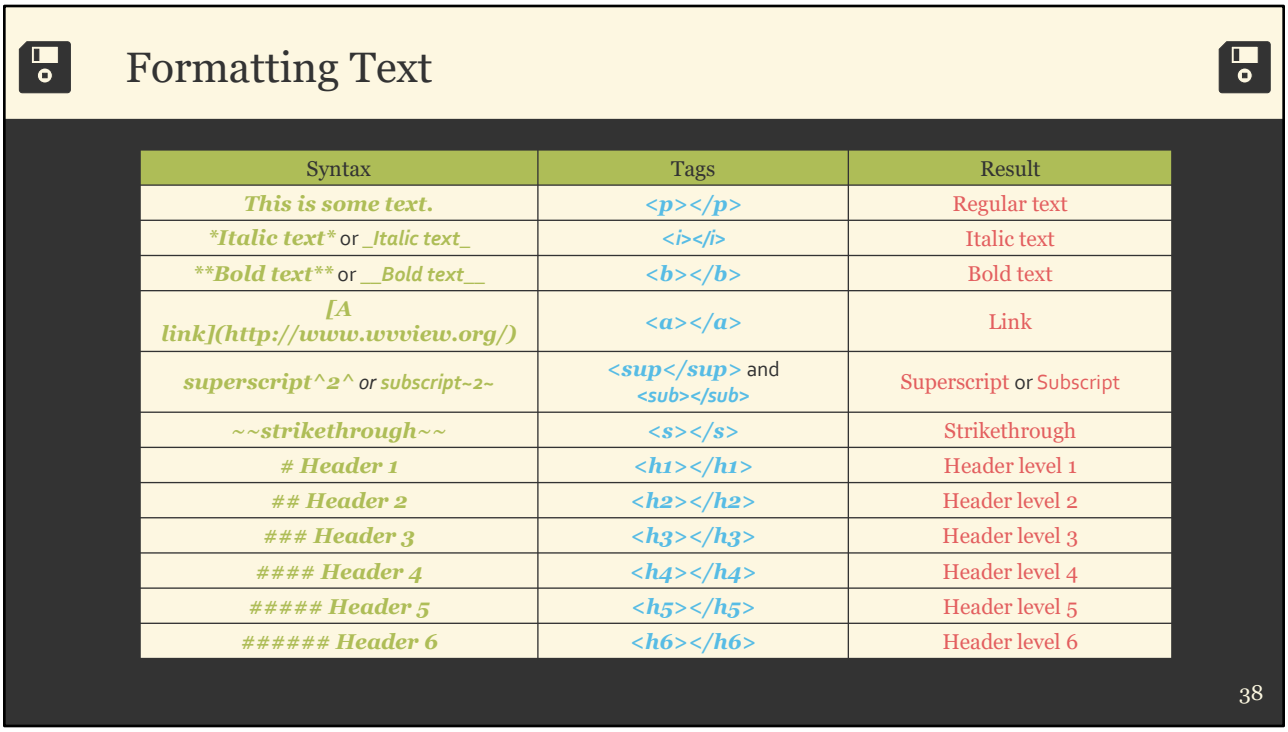
You can also build and customize themes using Cascading Style Sheets (CSS).

# Rendering Engines

| Engine | Main languages/workflows | Typical file types | Strengths | Limitations / cautions | Best use case |
|---|---|---|---|---|---|
| Markdown/no engine | Plain Markdown, no executable code | .qmd, .md | Fastest and simplest; no code execution dependencies; ideal for prose, lecture notes, websites, handouts, and static documentation | Cannot execute code chunks; .md files with executable chunks will error | Static teaching materials, documentation, syllabi, websites, conceptual notes |
| Knitr | Primarily R, also Python via reticulate, Bash, SQL, and other knitr-supported engines | .qmd, .Rmd | Best integration with R, RStudio, tidyverse, ggplot2, htmlwidgets, Shiny, and legacy R Markdown workflows; can mix R and Python through reticulate | Python execution is mediated through R/reticulate rather than native Jupyter; mixing many languages can become environment-sensitive | R-centered reproducible reports, statistics, GIS workflows in R, teaching with R |
| Jupyter | Python, Julia, R, Bash, and many other Jupyter kernels | .qmd, .ipynb | Best for Python-first workflows; integrates naturally with notebooks, JupyterLab, VS Code, scientific Python, and Jupyter widgets; supports many kernels | Kernel/environment management matters; R users may prefer knitr; output behavior can differ from knitr | Python-based data science, machine learning, geospatial Python, notebook-to-report workflows |

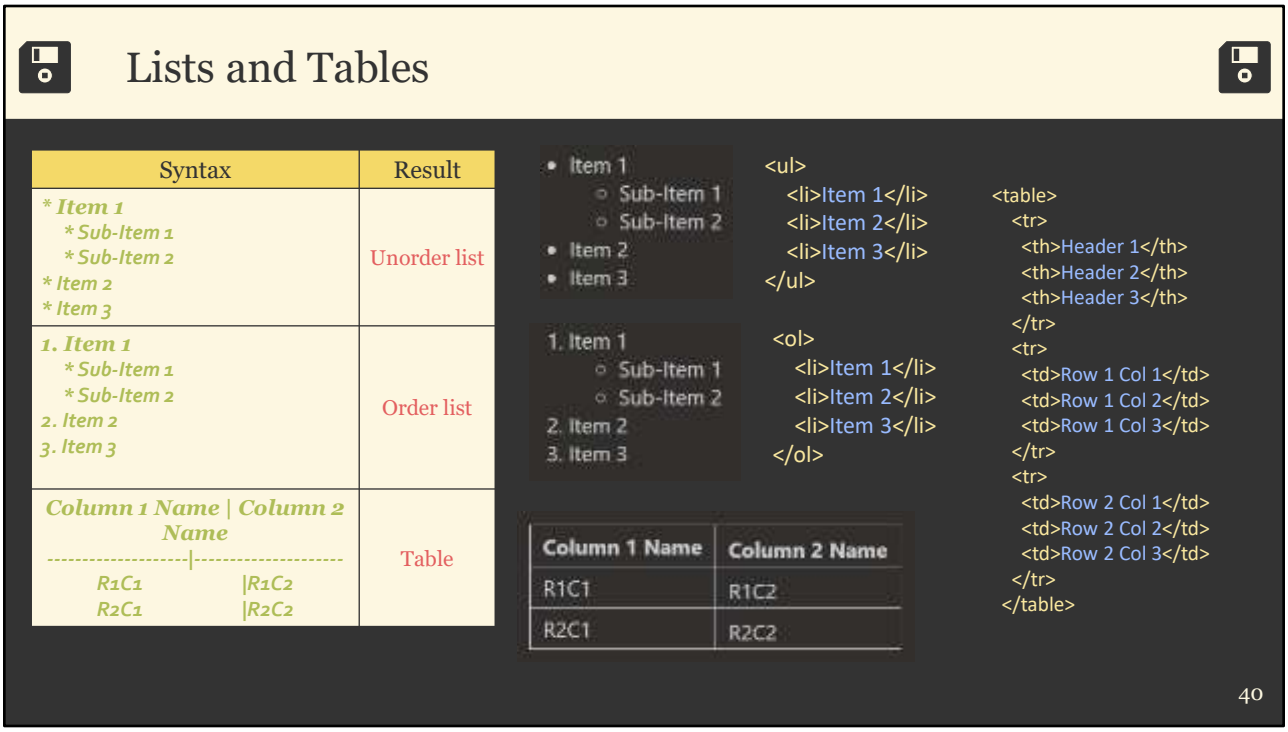
37

Different rendering engines can be used to create documents from Quarto files. Three commonly used rendering engines (Markdown, Knitr, and Jupyter) are described in the provided table. Note that I commonly use Knitr when working in RStudio.




Since Quarto builds on Markdown, text and text formatting are defined using the same methods/syntax as those used by Markdown.


This slide provides examples of Quarto/Markdown syntax, associated formatting tags, and the resulting output.




The table on this slide provides formatting examples for unordered lists, ordered lists, and tables. I have also provided examples of associated formatting tags. Again, Quarto and Markdown syntax are identical for these elements.

## Others

| Syntax | Tags | Result |
|---|---|---|
| *> This is a blockquote* | *<blockquote></blockquote>* | Blockquote |
| *`y = 2`* | *<code></code>* | Inline code |
| *---* | *<hr>* | Horizontal rule |
| *![Alt text](Link or URL)* | *<img>* | Image |

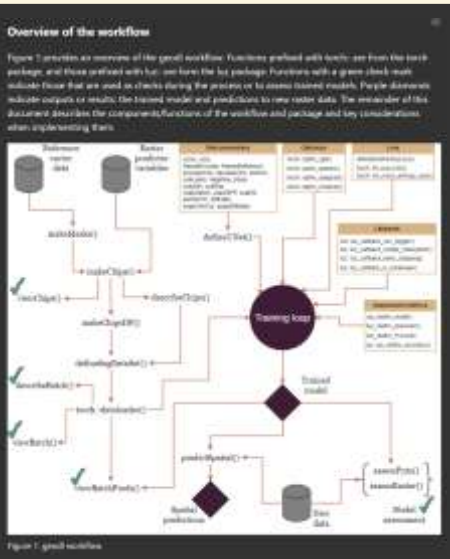
Note: it is not currently possible to add videos within GitHub README markdown. Can link to video using text or an image.

41

This slide provides examples for blockquotes, code, horizontal rules, and image links.

## Including images

```
![Figure 1. geodl workflow.](images/Fig1.png)
```




42

This slide demonstrates the syntax used to insert an image into a Quarto document. The text in brackets is the alt text or caption associated with the image.

## Code Examples

```{r}
#| label: r-chunk-name
```

```{python}
#| label: py-chunk-name
```

```
{r}
#| eval: true
#| echo: true
#| include: true
#| error: false
#| message: false
#| warning: false

myRecipe <- recipe(lcClass ~ ., data = trainD) |>
  step_range(all_numeric_predictors())
```

43

This slide demonstrates how code chunks are defined within a Quarto document. The user must specify the language and can provide a title if desired.

## Code Options

| Option | Description |
|---|---|
| eval | Evaluate the code chunk (if false, just echos the code into the output). |
| echo | Include the source code in output |
| output | Include the results of executing the code in the output (true, false, or asis to indicate that the output is raw markdown and should not have any of Quarto's standard enclosing markdown). |
| warning | Include warnings in the output. |
| error | Include errors in the output (note that this implies that errors executing code will not halt processing of the document). |
| include | Catch all for preventing any output (code or results) from being included (e.g. include: false suppresses all output from the code block). |
| renderings | Specify rendering names for the plot or table outputs of the cell, e.g. [light, dark] |

https://quarto.org/docs/computations/execution-options.html

44

As visualized on the prior slide, the syntax for specifying code rendering options is different from that used by R Markdown. This is one of the key differences. The table provided on this slide describes some of the most important formatting options. For example, eval: true indicates to execute the code while eval: false indicates to not execute the code. echo: true indicates to show the source code while eco: false indicates to not show it. output: true indicates to show the output while output: false indicates to not show it. The warning and error options can be used to suppress warning or error messages, respectively.

## Coding rendering/execution options

- Defined within YAML header or YAML syntax within code cells
- Options vary by engine (Jupyter, Knitr, or Observable JS)
- Code output options from Knitr: eval, echo, code-fold, code-summary, code-overflow, code-line-numbers, lst-label, lst-cap, tidy, tidy-opts, collapse, prompt, class-source, attr-source
- Cell output options from Knitr: output, warning, error, include, panel, output-location, message, results, comment, class-output, attr-output, class-warning, attr-warning, class-message, attr-message, class-error, attr-error
- Can also define caching options


https://quarto.org/docs/reference/cells/cells-knitr.html

45

As noted in the prior slide, specifying code/cell execution and output options are a bit different for Quarto in comparison to R Markdown. You can include options in the YAML header. If specified within the code chunk, instead of placing the options within the opening curly brackets, YAML syntax within the code block is used. See the example provided on this slide.

The options available depend on the rendering engine used. On this slide, we have provided option examples for Knitr. The link on the page provides descriptions for each of these options.

There are also options for caching code output, which can make rendering the document or product faster. Caching options are also discussed at the provided link.

## Inline Code

```
There are `r nrow(airquality)` observations in the airquality dataset,
and `r ncol(airquality)` variables.
```

https://qmd4sci.njtierney.com/using-qmd.html

46

This slide provides an example of inline code syntax.

## Other Content

- Equations https://qmd4sci.njtierney.com/math.html
- Citations and Bibliographies https://qmd4sci.njtierney.com/citations-and-styles.html
- Figure Customization https://qmd4sci.njtierney.com/changing-figures.html

47

The links on this slide are for chapters from the *Quarto for Scientists* text that discuss inserting equations and citations/bibliographies. The chapter associated with the last link discusses figure customization.

## Summary of Key Points

- Quarto was created by Posit and builds on Markdown.
- A variety of different document types can be generated.
- Quarto interfaces with different rendering engines.
- Text formatting is identical to that used for Markdown.
- Defining and setting options for code chunks are different in comparison to R Markdown.
- Quarto is a great tool for building a variety of different output documents.

This is the end of this lecture module.

Please return to the West Virginia View Webpage for additional content.

Thanks! Hope you found this useful.