Aims are big picture stuff broad questions something that i will contribute to but never truly answer.
Research questions tickable that lead up to the aims? Specific things:
Structure conclusion by running through these, outline? Asking the right questions-> link to questions! Text is answering the questions in a clear way.

What is Git?

- Version control system.
- It manages evolution of a set of files called a repo. Like “Track Changes” feature in Word but more powerful and scaled up to multiple files.
    - Take for example the analysis of whale data captured in a single R resource file. WIth informal version control, contributors can create derivative copies of whale.R decorating the file name with other descriptors.-> This leads to multiple versions of whale.R of intermittent relatedness. The original file whale.R quickly becomes part of a complicated phylogeny (#i.e evolutionary history of species. Phylogenetic analysis is the means of estimating the evolutionary relationships.) that no amount of “Track Changes” can resolve. (#bottom line, Git very good)
    - Git way of tracking the evolution of whale.R through a series of **commits,** equipped with an explanatory message.
    - Shows same history for a common collaborative GIt workflow. Contributors work independently but sync regularly to a common version.
- Repurposed for data science- manage collection of files that make up data analytical projects consisting of data, figures, source code etc.

Why use Git?

- Pros only outweigh cons when you consider overhead of working with other people including my future self.
- Model of file management might be rigid but it enables distbn of files across different people, computers and time (#particularly impt now?)
- Will gain the most if you pick a project that involves rapidly evolving files with others.

What is GitHub?

- GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.
- Slick user interface and distribution mech for Git repos. (GitHub is like a GDrive but more powerful and powerful)
- Git is the software you will locally use to record changes to a set of files vs GitHub is a hosting service that provides a home for such projects on the internet.

- Easy to create a hyperlink to a specific file or location in a file, at a specific version, which can make meta-conversations about project code or reports much more productive. (#why does Data Science in EES encourage doing things through R and not through the webpage then?)

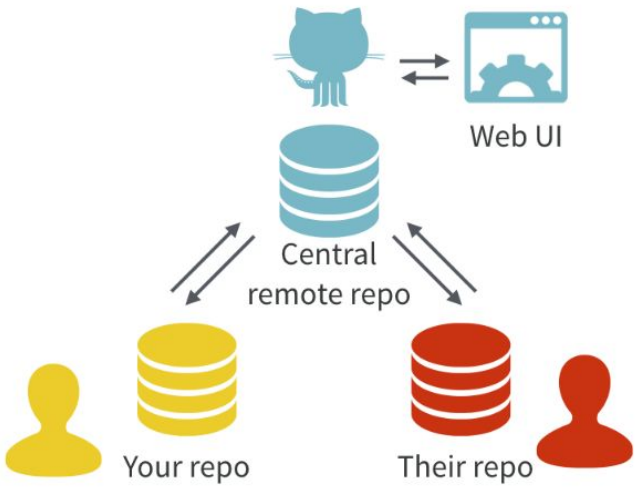


Figure 2: With Git, all contributors have a copy of the repo, with all files and the full history. It is typical to stay in sync through the use of a central remote repo, such as GitHub. Hosted remotes like GitHub also provide access to the repo through a web browser.

Tips:

- Recommend using a Git client (e.g GitKrakee, SourceTree vs restricting yourself to command line interface
- Make a change, commit it -> pull -> push?
- GitHub CANNOT display content if file is binary such as Word doc or Excel spreadsheet.
- The adoption of Git/GitHub suggests a pivot AWAY from .docx, .xlsx, and .pdf as primary file formats and towards .Rmd, .md, .csv

Terms:

Commit= takes a snapshot of all the files in the entire project. Version that is significant to you and that you might want to inspect or revert to later. Commit message ideally conveys the motivation for the change you made because diff will show the content (#of that change?)
Push= commits to github periodically. By pushing to Github, you make your work and all your accumulated progress accessible to others. Is like sharing doc with peers by sending it out as email attachment.
diff= set of differences between A and B is called a “diff”. With A being version A of one git commit and B being part of the next commit.
Pull request= Way to propose changes to a repo that that overlays Git’s regular branch and merge workflow with facilities for structured review. Pull request can be made between two branches in the same repo, an original and a so-called “fork”. Mechanism for making and accepting contributions to open source software projects on GitHub.
Diff inspection= how you re-explain to yourself how version A differs from version B. Not limited to adjacent commits, inspect the diffs between any two commits.
Branch= independent line of development line of development within a repo, where the intent is usually to merge it into the master branch when ready.

R Markdown:

- Just Markdown that includes chunks of R code.
- Provides unified authoring framework for data science, combing code, its results and my prose commentary.
- R Markdown docs fully reproducible and support different output formats e.g PDFs, Word files, slideshows etc
- Rmd files designed to be used in 3 ways
    1) For communicating to decision makers who want the simplified conclusions, not necessarily the code behind the analysis
    2) For collaborating with other data scientists including future self, who are interested in BOTH conclusions and how you reached them (i.e the code)
    3) As an **environment** in which to do data science. Captures not only what I did, but what I was thinking.
- rmarkdown package converts R Markdown files (.Rmd) to Markdown files (.md)
- Once .Rmd document has been rendered to .md (i.e R markdown doc converted to markdown file via the above) committed and pushed to GitHub, anyone viewing it on GitHub can read the prose, study the R code and view the results of running that code.
    - Reader doesn't need to download the code/install stuff/run it, reader can just access/gaze upon the product in a web browser. (#easier for others?)
    - Point is that a director that is a GitHub synced Git repo can simultaneously be the code-heavy back end of the proj and an outward-facing front end (#latter is a clean and tidy product for others to view and understand).
    - Therefore provides lightweight system for exposing work-in progress to collaborators without slowing down to create separate reports.

*Help > Cheatsheets > R Markdown Cheat Sheet*
*Help > Cheatsheets > R Markdown Reference Guide*.

General Use:

- What if two people have made changes to the repo? Imagine that friend 1 makes a change to a file, commits it locally and pushes to GitHUb. Meanwhile I also make a diff change to the same file and also commit locally. If try to push my commit to GitHub, I will FAIL because there are commits on GitHub that I do not have (#friend has changed something!). So… -> I MUST PULL. After successful merge -> can PUSH
- Merge conflict? I.e happens when Git doesn't know how to jointly apply the diffs from 2 different commit to their common parent.
    - Best way to deal is to prevent them. So therefore must commit-> pull -> push often.
    - Or else at each location of conflict, must pick one version or the other or create a hybrid and mark it as resolved.
- GitHub also offers ways to host a proper website directly from a repo collectively known as GitHub Pages.
    - One extreme: As long as I have one .md file, GitHub Pages can create a simple website automatically.
    - Other extreme: Can take advantage of Jekyll static site generator (#what is this?)

Workflow

- R statements (assignment statements) where you create objects have the same form:

`object_name <- value`
`When reading this, read as “object name gets value)`

- Can create new objects with `<-`
  `E.g x <- 2 + 2`
  `To do this use RStudio’s keyboard shortcut: Alt + - (latter is the minus sign). ALways make sure that arrow is surrounded by spaces.`
- Can inspect an object (i.e x in this case) by typing its name
  `x`
  `#> [1] 4`

- Another example:
  `this_is_another_example <- 33.2`
  `To inspect this object, try this shortcut (#tho I think this is more complicated than it’s worth?): Type “this”, then press TAB, add characters until you have a unique prefix, then press return`
  `If you’ve made a mistake (e.g this_is_another example should have a value of 33.3 instead of 33.2, try this shortcut: Type “this” then press Cmd/Ctrl + ↑. This will list all the commands you’ve typed that start with those letters (i.e “this”). Use the arrow keys to navigate then press enter to retype the command. Change 33.2 to 33.3 and rerun.`

https://github.com/jrnold/r4ds-exercise-solutions/blob/master/workflow-basics.Rmd

- You can run each code chunk by clicking Run icon (play button at the top of the chunk) OR by pressing Ctrl+Shift+Enter
- To produce complete report containing all text, code and results, click “Knit” OR Ctrl+Shift+K OR `rmarkdown::render("1-example.Rmd")`
- After you knit, R markdown sends .Rmd file to knitr, which executes all code chunks and creates a new .md doc which includes the code and its output. .md file generated is then processed by pandoc which is responsible for creating the finished file. Pros of this 2-step workflow is that you can create wide range of output formats
- To run code inside an R Markdown document, you need to insert a chunk. There are 3 ways to do so :
  1) They keyboard shortcut Ctrl+Alt+I
  2) The “Insert” button icon in the editor toolbar
  3) Manually type chunk limiters ```` ```{r} ```` and ```` ``` ````
     https://yihui.org/knitr/options/?version=1.2.5019&mode=desktop (#covers chunk options etc)
- can continue to run the code using the keyboard shortcut Ctrl + Enter.
- However, chunks get a new keyboard shortcut: Ctrl + Shift + Enter, which runs all the code in the chunk. Think of a chunk like a function. A chunk should be relatively self-contained, and focussed around a single task.
- Chunk header consists of ```` ```{r ````, followed by an optional chunk name, followed by comma separated options, followed by `}`. Next comes your R code and the chunk end is indicated by a final ```` ``` ````
- `Chunk name by` ```` ```{r by-name}. ```` Good for navigating to specific chunks using the drop down code navigator in the bottom-left of the script editor. Good also because you can set up networks of cached chunks to avoid re-performing expensive computations on every run. one chunk

name that imbues special behaviour: `setup`. When you're in a notebook mode, the chunk named setup will be run automatically once, before any other code is run.

- Chunk options:

  The most important set of options controls if your code block is executed and what results are inserted in the finished report:

  - `eval = FALSE` prevents code from being evaluated. (And obviously if the code is not run, no results will be generated). This is useful for displaying example code, or for disabling a large block of code without commenting each line.
  - `include = FALSE` runs the code, but doesn't show the code or results in the final document. Use this for setup code that you don't want cluttering your report.
  - `echo = FALSE` prevents code, but not the results from appearing in the finished file. Use this when writing reports aimed at people who don't want to see the underlying R code.
  - `message = FALSE` or `warning = FALSE` prevents messages or warnings from appearing in the finished file.
  - `results = 'hide'` hides printed output; `fig.show = 'hide'` hides plots.
  - `error = TRUE` causes the render to continue even if code returns an error. This is rarely something you'll want to include in the final version of your report, but can be very useful if you need to debug exactly what is going on inside your `.Rmd`. It's also useful if you're teaching R and want to deliberately include an error. The default, `error = FALSE` causes knitting to fail if there is a single error in the document.

The following table summarises which types of output each option supressess:

| Option | Run code | Show code | Output | Plots | Messages | Warnings |
|---|---|---|---|---|---|---|
| `eval = FALSE` | - | | - | - | - | - |
| `include = FALSE` | | - | - | - | - | - |
| `echo = FALSE` | | - | | | | |
| `results = "hide"` | | | - | | | |
| `fig.show = "hide"` | | | | - | | |
| `message = FALSE` | | | | | - | |
| `warning = FALSE` | | | | | | - |

- Table with additional formatting use the `knitr::kable` function.

```
knitr::kable(
  mtcars[1:5, ],
  caption = "A knitr kable."
)
```

The code below generates this table:

Table 27.1: A knitr kable.

| | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.62 | 16.5 | 0 | 1 | 4 | 4 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.88 | 17.0 | 0 | 1 | 4 | 4 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.32 | 18.6 | 1 | 1 | 4 | 1 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.21 | 19.4 | 1 | 0 | 3 | 1 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.44 | 17.0 | 0 | 0 | 3 | 2 |

  - Read the documentation for `?knitr::kable for more customisation`

- `Caching.` Can be painful if you have some computations that take a long time. The solution is `cache = TRUE.` When set, this will save the output of the chunk to a specially named file on disk. On subsequent runs, knitr will check to see if the code has changed, and if it hasn't, it will reuse the cached results.
  - good idea to regularly clear out all your caches with `knitr::clean_cache()`