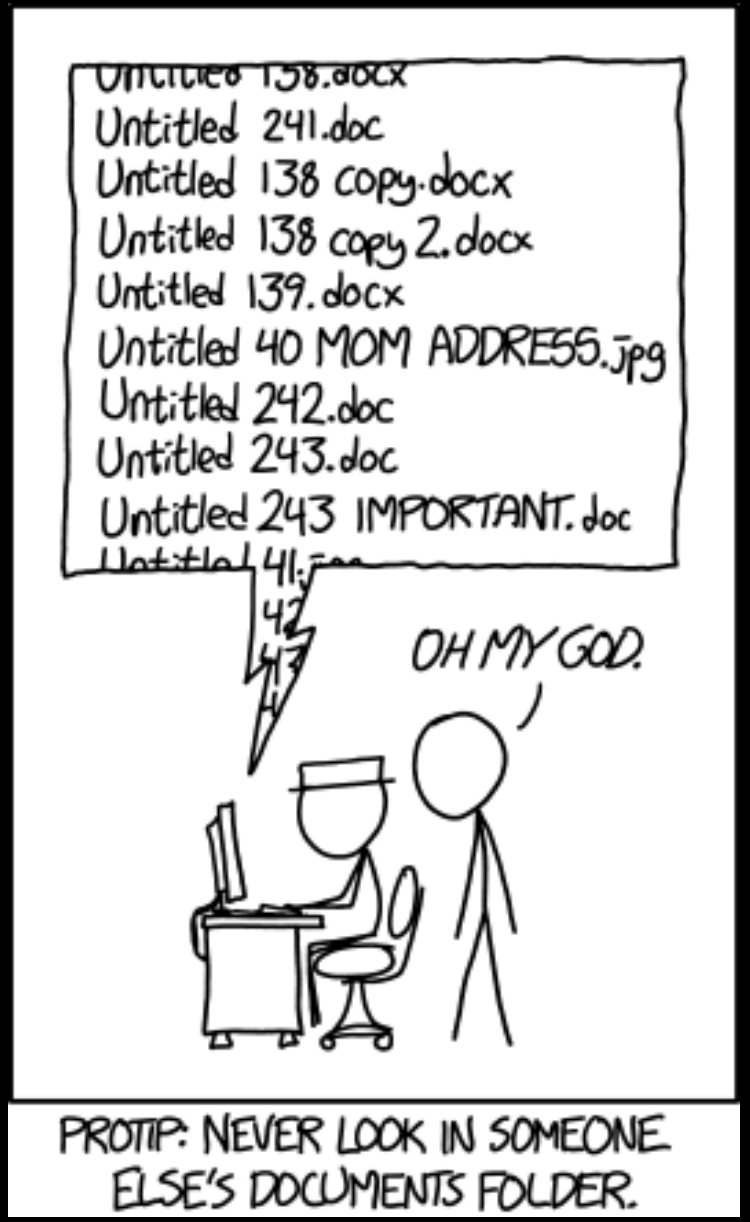
~~Best~~ "Better" Coding Practices

Managing your coding projects for homework and research so you don't hate yourself in six months time

Tom Logan

tomlogan@umich.edu


Untitled 241.doc
Untitled 138 copy.docx
Untitled 138 copy 2.docx
Untitled 139.docx
Untitled 40 MOM ADDRESS.jpg
Untitled 242.doc
Untitled 243.doc
Untitled 243 IMPORTANT.doc
OH MY GOD.
PROTIP: NEVER LOOK IN SOMEONE ELSE'S DOCUMENTS FOLDER.

Shout out to Jeff Leek - inspiration

- *How to be a modern scientist* by Jeff Leek – very short and sweet guide to doing research and coding properly (highly recommend)

Jenny Bryan

- “Your closest collaborator is you six months ago, but you don’t reply to emails.”

- “If the thought of re-running your analysis makes you ill, you’re not doing it right.”

Let's update academia!

- While the world has changed rapidly, academia has **not**.
- Limitations of "the paper"
  - a brief summary of a research project
  - no access to all the data, code, other learning etc.

- Alternative is "Github of Science"
  - Open-source
  - Continuous, collaborative development
  - Incentive changes are necessary, but let's equip ourselves

Outline

- Beforehand
- Repository/directory structure
- Github -> version control
- Sharelatex (if time permits)
- Logging
- Atom
- Code Styles

Before the workshop 1/2

- Have a code project folder that you'd like to clean up or start building or at least have one in mind
- Install Atom or other text editor
- Create yourself a profile on github.com AND get the student developers pack

  https://education.github.com/pack
- Create an account on ShareLaTeX (optional)

  https://www.sharelatex.com

Before the workshop 2/2

WINDOWS

- Show hidden files, folders and drives
- Install git: https://www.develves.net/blogs/asd/articles/using-git-with-powershell-on-windows-10/#installing-git

MAC

- In the terminal run
  xcode-select –install
  brew install git-lfs

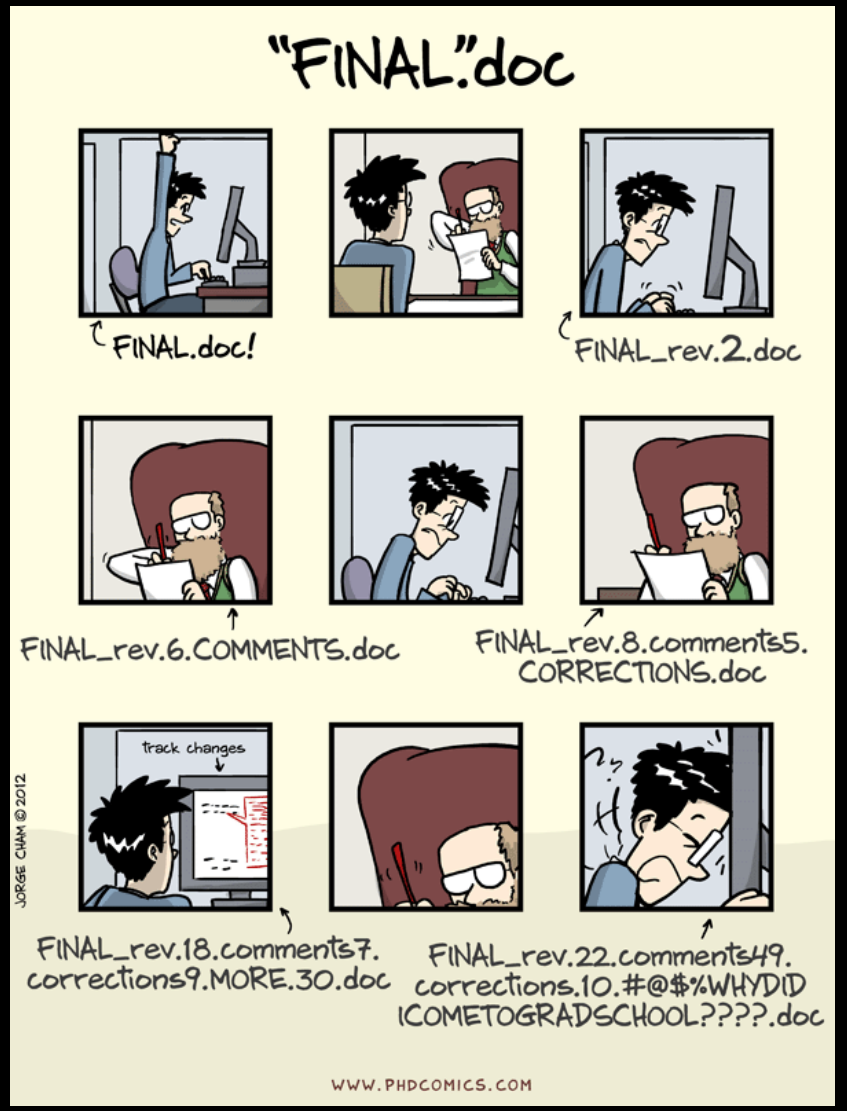
LINUX

- In the terminal run
  apt-get install git
  apt-get install git-lfs

Why

GitHub

- Version control – it tracks the entire history of a doc so you can
  - Go back
  - Compare changes
  - See how you did something
- Version control is one of those things that you don't need, until you *really really really* need it.
- Applies to code *and* reports/documents (but not data)



Go

1. Go to github.com
2. Create a PRIVATE repo (unless you already have one for your project)
3. Name (no spaces!)
4. No need for readme or .gitignore yet

5. In the (Ubuntu) terminal – cd into the directory where you want your project

   git clone <copy from github>
6. Move your .git folder into the directory you want if it's not already there

Basic

- Repo – repository/ project directory
- Git – the version control software that Github runs
- Commit – a timestamp on a set of changes
- Push – send to online repo
- Pull – bring onto your computer from online repo
- Merge – reconciling differences in two versions

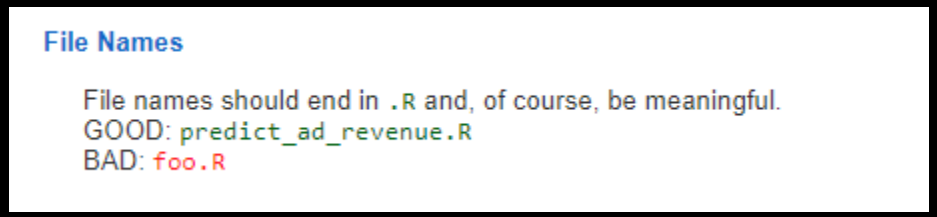
Directory

- Go to your directory in your file explorer
- We're going to create the following structure

- <your project name>/
  - .git
  - data/ *(or src)*
    - raw/
    - processed/
  - fig/
    - exploratory/
    - final/
  - report/
  - code/ *(the sub folders are project specific)*
    - processing/ *or data_processing*
    - analysis/
    - plotting/
    - logging-code
  - _scratch/

File and folder naming

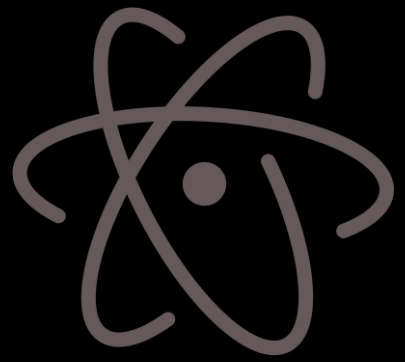
- Never put spaces in folders or file names (use underscores or dashes, use lowercase)
- R style guide:



File Names

File names should end in `.R` and, of course, be meaningful.
GOOD: `predict_ad_revenue.R`
BAD: `foo.R`

Time for Atom



- Open Atom
- File -> Open Folder -> select your project
- This should open the folder and directory on the LHS
- Toggle the git tab (ctrl+shift+9 OR view->Toggle)

Create your .gitignore

- Right click on the primary level in the directory
- New File
- Name it .gitignore
- The .gitignore is for things you DON'T want pushed to git. This includes private info and data or files that are created by your code.
- Add the following lines *(there will be more)*

  **/_scratch

  data/ *(more on this later)*

  fig/exploratory

  .log
- Add other directories or file extensions you don't want to push online (e.g. .csv, .rds, .RData, …)

Markdown

- Markdown is a plain text and quick way of writing
- You can write in code blocks, math, normal, tables, lists etc.
- It is how you can thoroughly document your project, what the steps are, and why

- Guides
- Markdown in a minute: https://www.markdowntutorial.com/
- Markdown cheatsheet: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Markdown

- You can preview your markdown by clicking ctrl+shift+m
- Code

  ```python
  >>> print("Hello world!")
  Hello world!
  ```

- Table

```
| Tables        | Are           | Cool  |
| ------------- |:-------------:| -----:|
| col 3 is      | right-aligned | $1600 |
| col 2 is      | centered      |   $12 |
| zebra stripes | are neat      |    $1 |
```

Write your README.md

```
# Land Surface Temperature
Tom M Logan
www.tomlogan.co.nz

## Description:
Understanding the factors influencing urban land surface temperature during the night and day.

## Steps:
1. Process the LandSat images to land surface temperatures
2. Statistical analysis

## 1. LandSat images to ...
### 1.1 ...
...

## Code Book
<table of variable names and descriptors>
```

Commit messages

Writing good commit messages

Bad

- some css
- styling
- ooops
- misc fixes and cleanups

Good

- add subtle background pattern to body
- make subheadings larger on archive pages
- fix typo in site footer
- cleanup code with htmltidy

Make your init commit

- Now it’s time to push your first changes to github
- Stage your changes – on the git panel – “Enter” toggles the stage
- Stage is any you want to be included in your next commit.
- Write your commit message “init commit”
- Commit – Ctrl Enter
- Push

- Now go to your github.com repo and see the new changes!

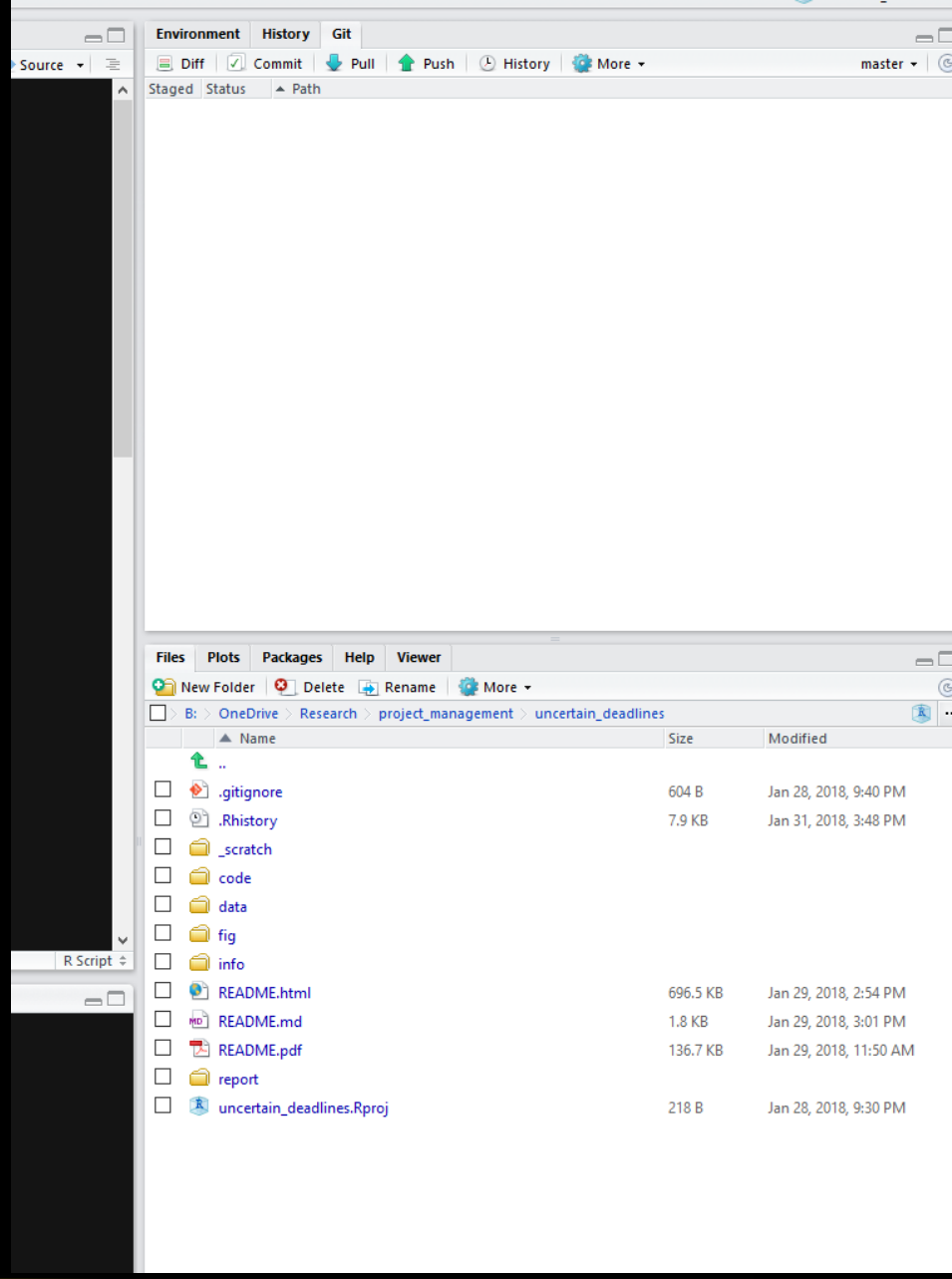
RStudio

- Rstudio provides a nice interface similar to Atom for use with R projects
- Open Rstudio
- File -> new project
- New Directory
- This will create a .Rproj file in the directory

- Setup a project – use markdown – push/pull/commit etc.

Rstudio

- This should put a git option on the RHS bar and the file directory
- Open .gitignore
  - Add .Rhistory and .html and .Rproject and .Rdata
  - You should see the changes appear in the git tab
- Open README.md – knit it and check it out
- Stage, commit, push!



Reproducibility and code clarity

- Code book *(depends on size of data, but if you're doing statistics, there must be somewhere that records the var_name and description)*
  - Human readable is not the same as computer readable
  - In your README.md have a table with

    var_name | description | unit
- Your README.md should include an explicit and exact recipe for how you did the analysis – include assumptions you made. This is basically what you'll put into "methods" so writing this now will make your job easier.

Code

- You may find having different scripts for the following useful:
  - Processes raw data into processed data
  - Analyse processed data
  - Plotting functions (which may be called from the analysing function)
  - Logging (so you can call this from other scripts)

Code style

- Python

  https://www.python.org/dev/peps/pep-0008/

- R

  https://google.github.io/styleguide/Rguide.xml

Every language has a style guide. Know yours.

Code function and variable names

R

- `variable.name` is preferred, `variableName` is accepted
  GOOD: `avg.clicks`
  OK: `avgClicks`
  BAD: `avg_Clicks`
- `FunctionName`
  GOOD: `CalculateAvgClicks`
  BAD: `calculate_avg_clicks` , `calculateAvgClicks`
  Make function names verbs.
  Exception: When creating a classed object, the function name (constructor) and class should match (e.g., lm).
- `kConstantName`

Python

"Function names should be lowercase, with words separated by underscores as necessary to improve readability."

"Variable names follow the same convention as function names."

Data

- raw data is anything that you download or collect
  - It has NO processing
  - All processing should be in a script file – e.g. you should be able to run a script and it'll create the processed data
- processed data (tidy data set) is what you analyse

Data uploading (case dependent!)

- Data can be uploaded to git… but not through the normal way. Use git lfs
- Ideally just upload a zipped version of your raw data (which can get converted to the processed/analysed data) and the data on which you run your analysis.

- Alternatively you can get a DOI for data that you upload to Figshare – it's free if it's public – so post your raw and analysis data there once you're accepted!

Fig

- Any analysis with data starts with exploring the data (or should! See Leek's book *The elements of data analytic style*)
- Create figures
- Add to fig/exploratory
- These don't have to be pretty! – just make sure you record how you created them (or name the axis appropriately)

- fig/final is for the pretty figures that'll go into report or presentations

Report

- Regardless of word or latex – version control will be useful
- ShareLaTeX has seamless integration with Git
- Example

  Show ShareLaTeX integration with Github

- This works great with .bib files that you output from Paperpile or Mendeley

ShareLaTeX

- Check out my blog discussing this, OverLeaf, Google Docs, and Word:

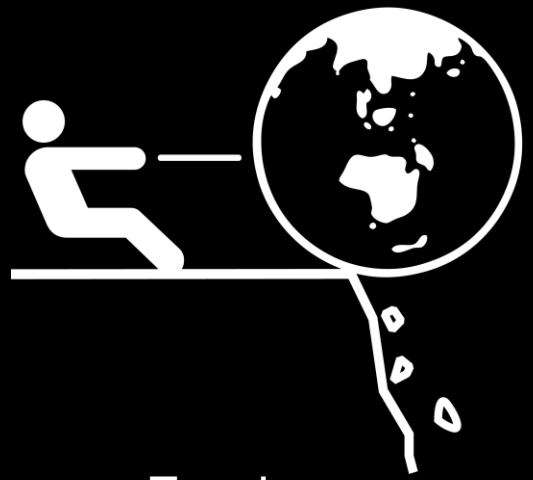
  http://reckoningrisk.com/research-practice/2017/comparing-editors-for-reports/

Paperpile – citation manager

- Check out our blog discussing Paperpile:
  http://reckoningrisk.com/research-practice/2017/literature-reviews/

Other random thoughts while I put this together

- Learn to touch type: https://www.dancemattypingguide.com/dance-mat-typing-level-1/
- If you want to use git without atom, that's fantastic and you should know how to do it. A simple guide http://rogerdudler.github.io/git-guide/



Tom Logan
reckoningrisk.com
tomlogan@umich.edu
@tomMLogan