Getting started with Git and GitHub

Mark Hanly

UNSW

2019/03/07 (updated: 2022-05-30)

The challenge

The challenge

As data analysts, we produce a lot of files

- code
- figures
- tables
- reports

Organising these files is an important, albeit often unglamorous, part of our job.

Managing versions What do you do with old code?

Ensuring reproducibility Can others reproduce your results? What about your future self?

Facilitating collaboration Can other team members easily contribute?

The old school solution

🚲 The old school solution

The old school solution combines many practical strategies:

- Using file naming conventions to track different file versions
    - analysis-16jun20.do
    - analysis-16jun20-mh.do
    - analysis-final.do
    - analysis-final-final.do
- Note changes directly in code comments or separate documentation
- Use shared folders, google drive etc
- Share code and code snippets over email, slack etc

This is ok(ish) for one person working on a small, brief project.

But if the project is **large**, involves **many people** or goes on for a **long time** this approach very quickly breaks down 😟

The new school solution

🚀 The new school solution

The new school solution is to use dedicated **version control software**.

There are a few options out there but by far the most popular choice is the combination of **Git** + **GitHub** .

Via git-scm.com

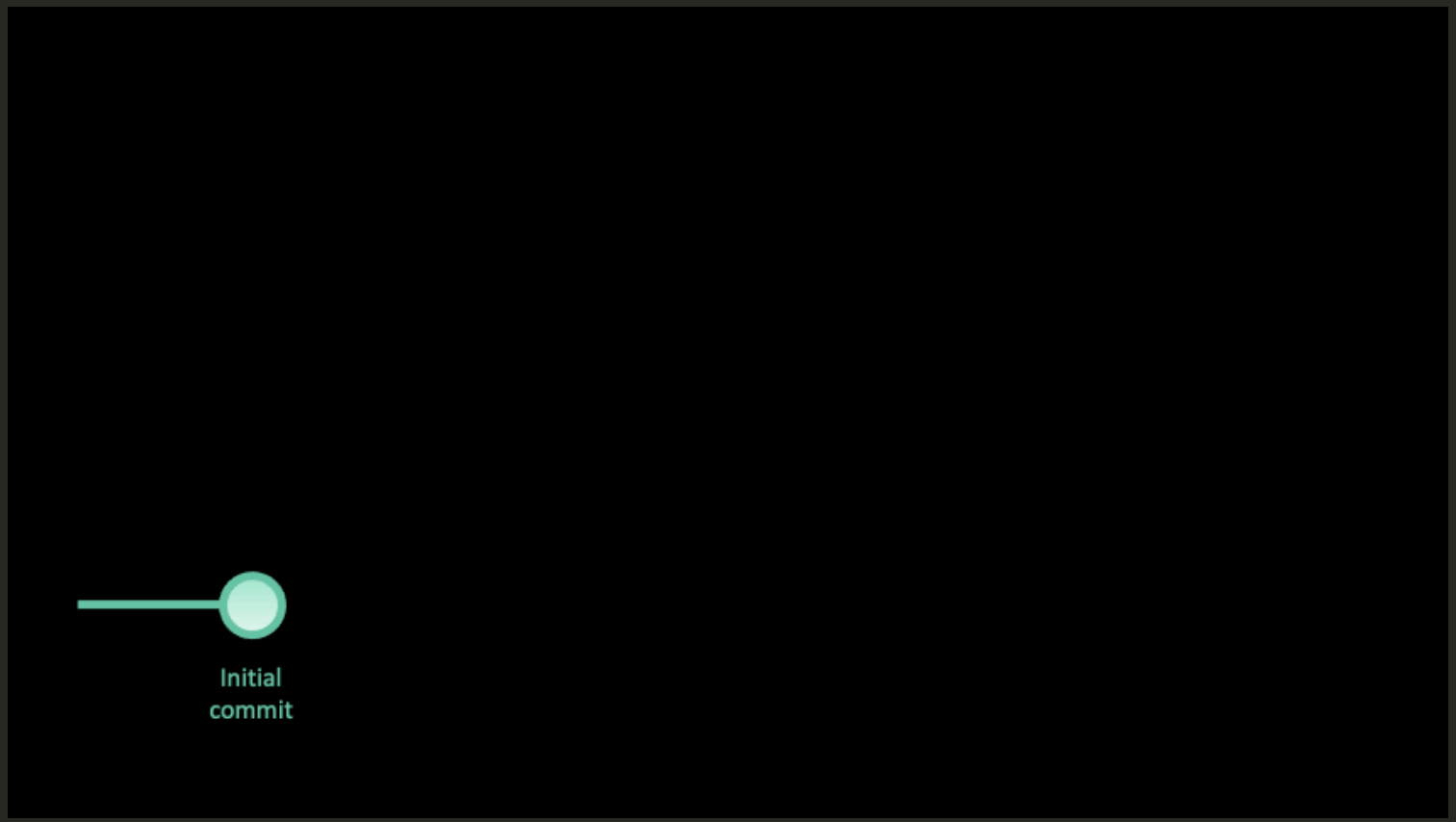
Git is software for tracking different versions of code files you are working on **locally**.

Via logos-world.net

Git Hub is a **cloud-based** platform for sharing and collaborating on projects managed with Git

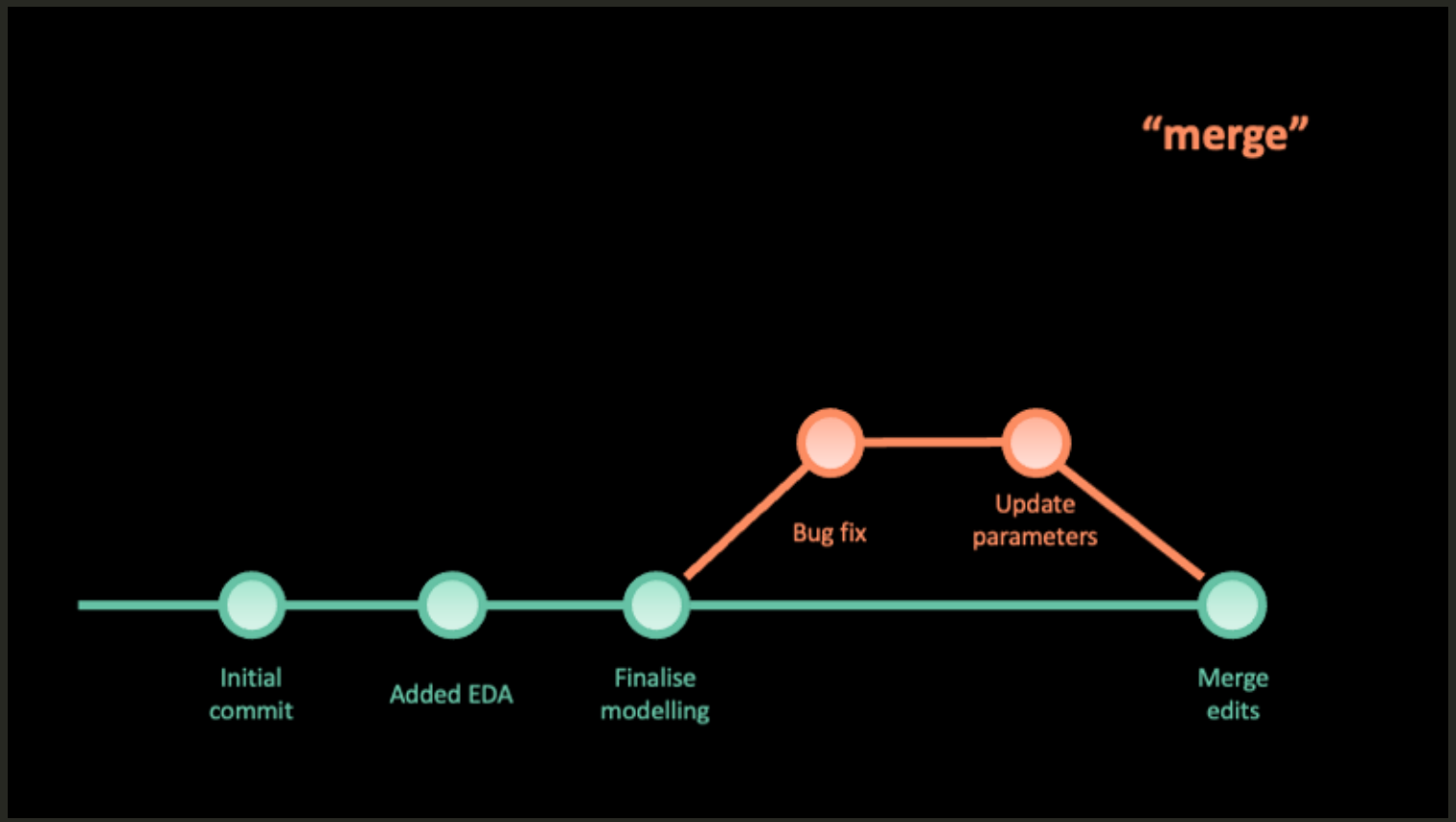
How does it work?

Committing, branching and merging with Git



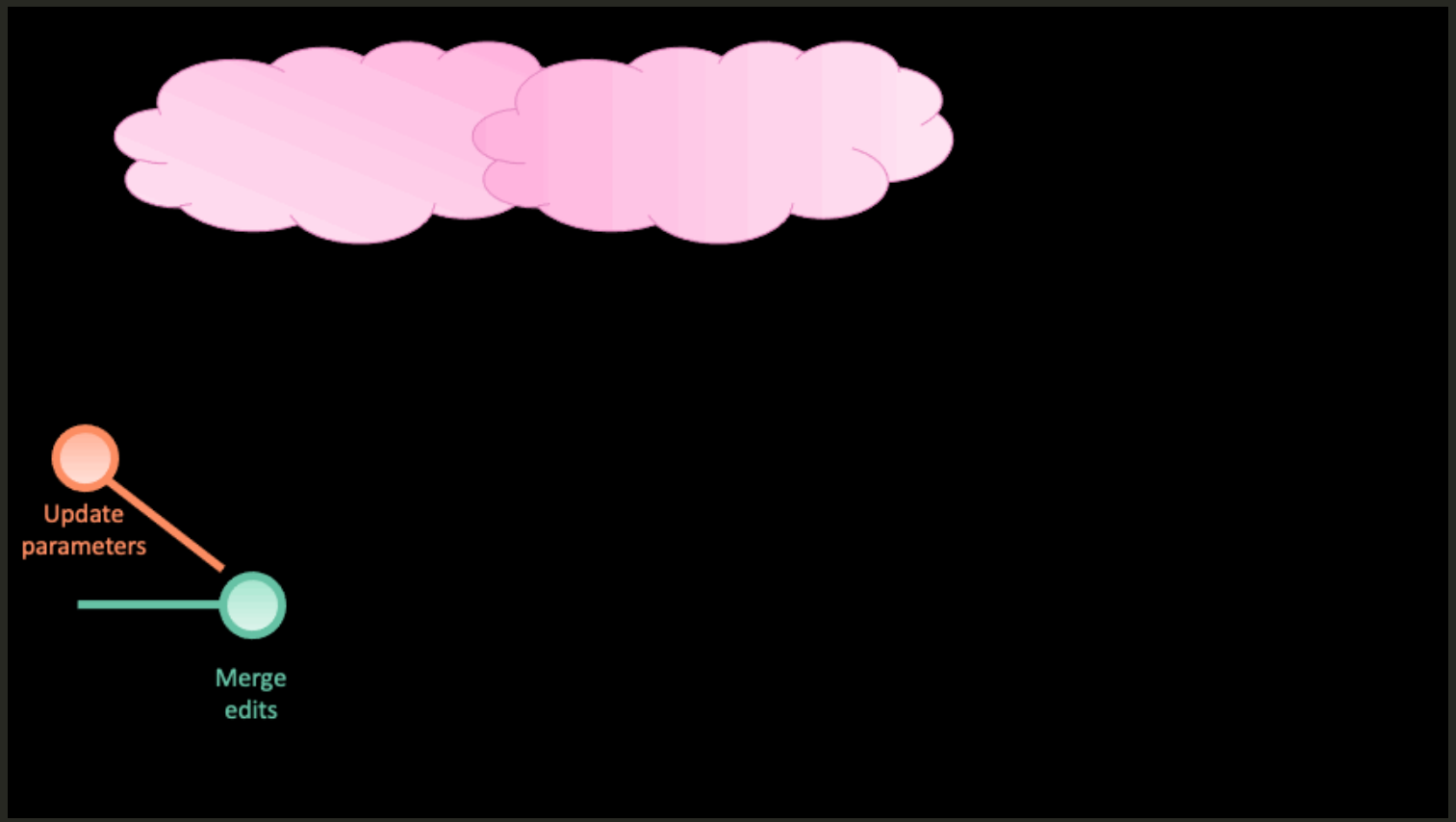
How does it work?

Committing, branching and merging with Git



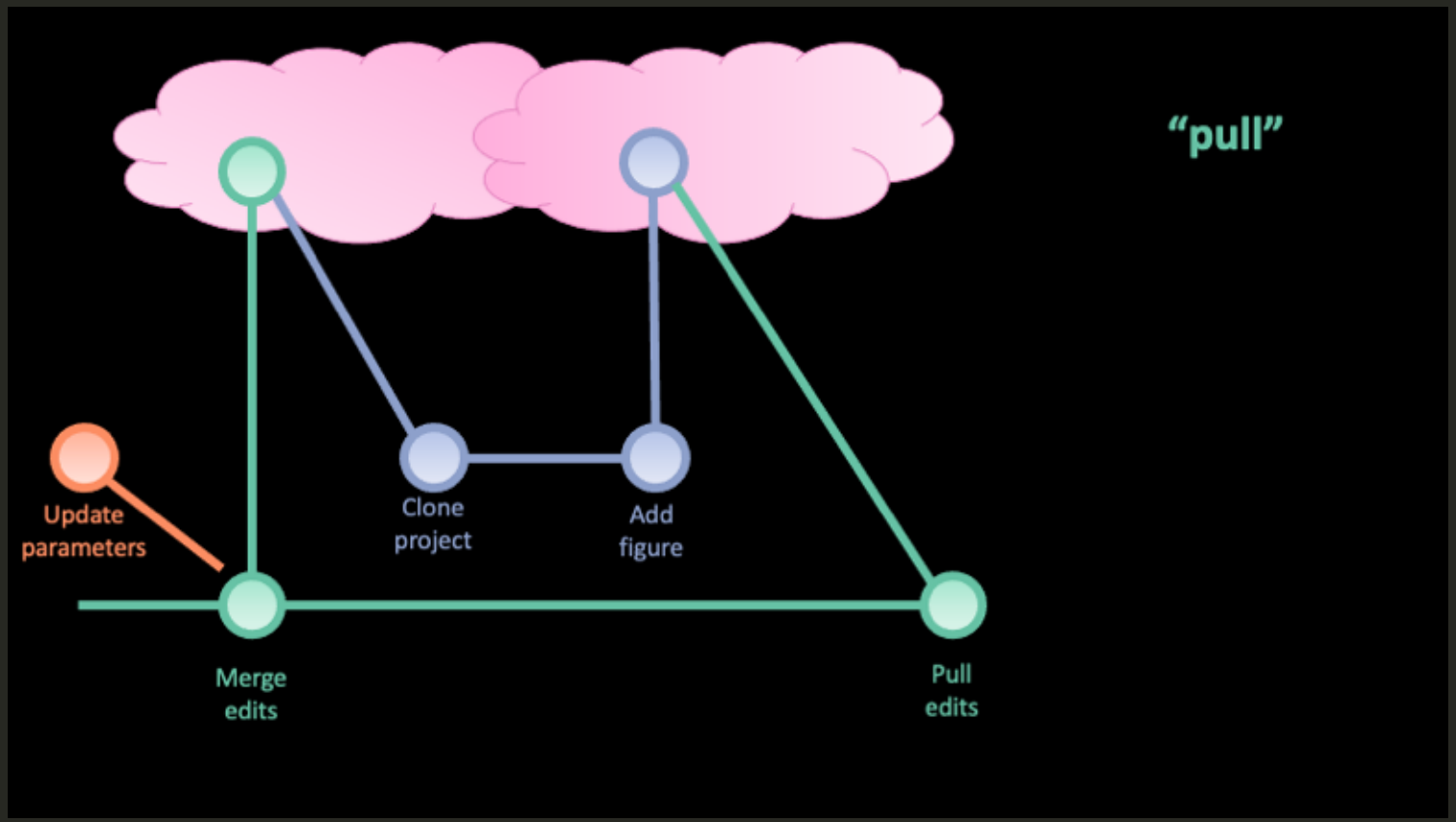
How does it work?

Sharing versions with GitHub

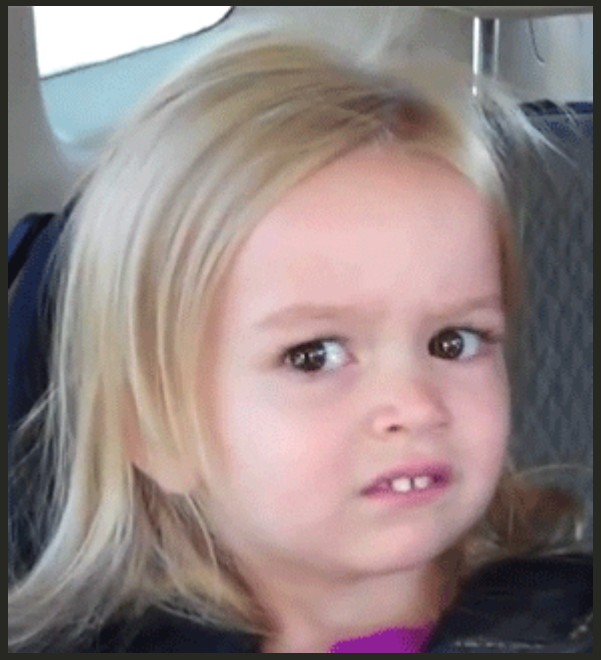


How does it work?

Sharing versions with GitHub



It seems a bit complicated



Via GIPHY

Is it worth it?

Yes!! It is definitely worth it!

1. Version control | Essential for reproducibility

2. Collaboration | OpenSAFELY, OHDSI, R package development

3. Organisation | e.g. Code, data & outputs all stored together

4. Easy web presence | e.g. any public GitHub repo; these slides!

5. Great for teaching

6. Powerful search

7. Issue tracking | Hyperlink the bug, discussion about the bug & the solution!

8. It is all *free** | *Well, mostly. The free products will cover most academic/research use cases.

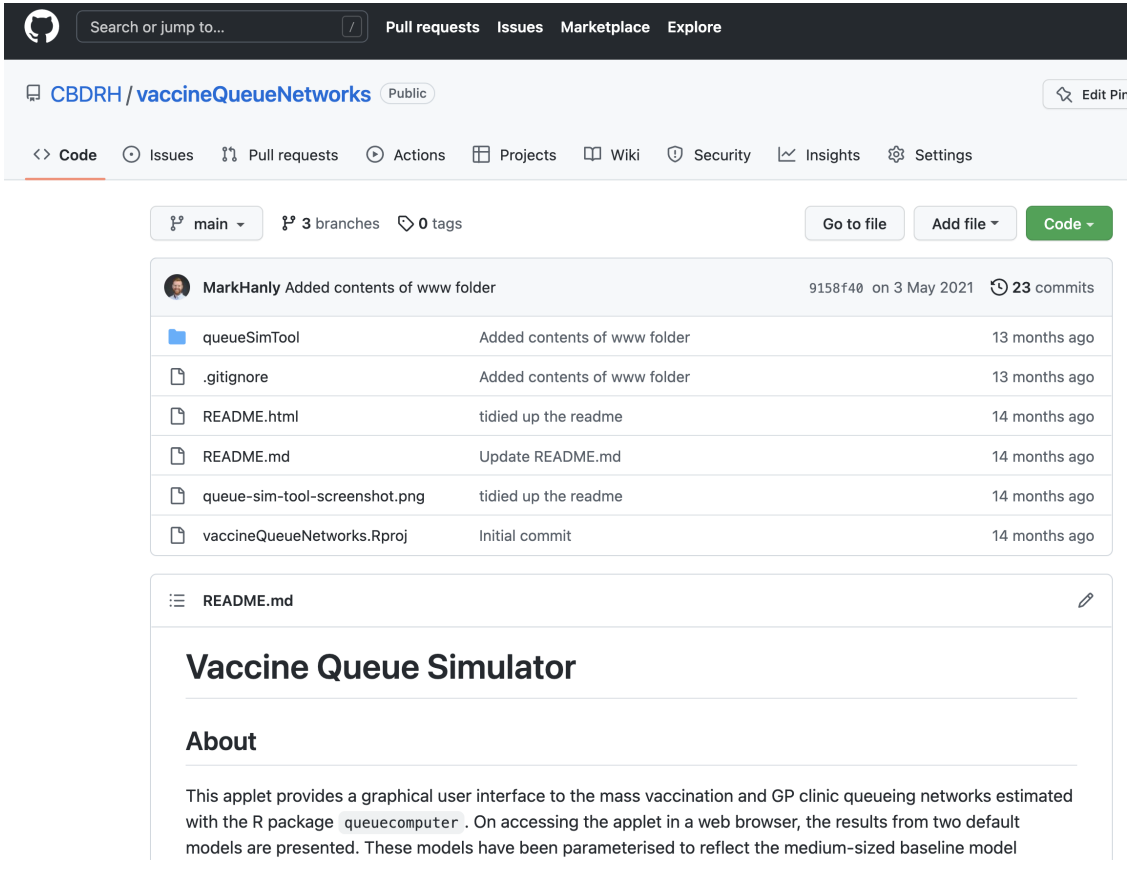
Let's check out an example

If reading this at home explore the repo for:

Commit history

Branches

Version comparison



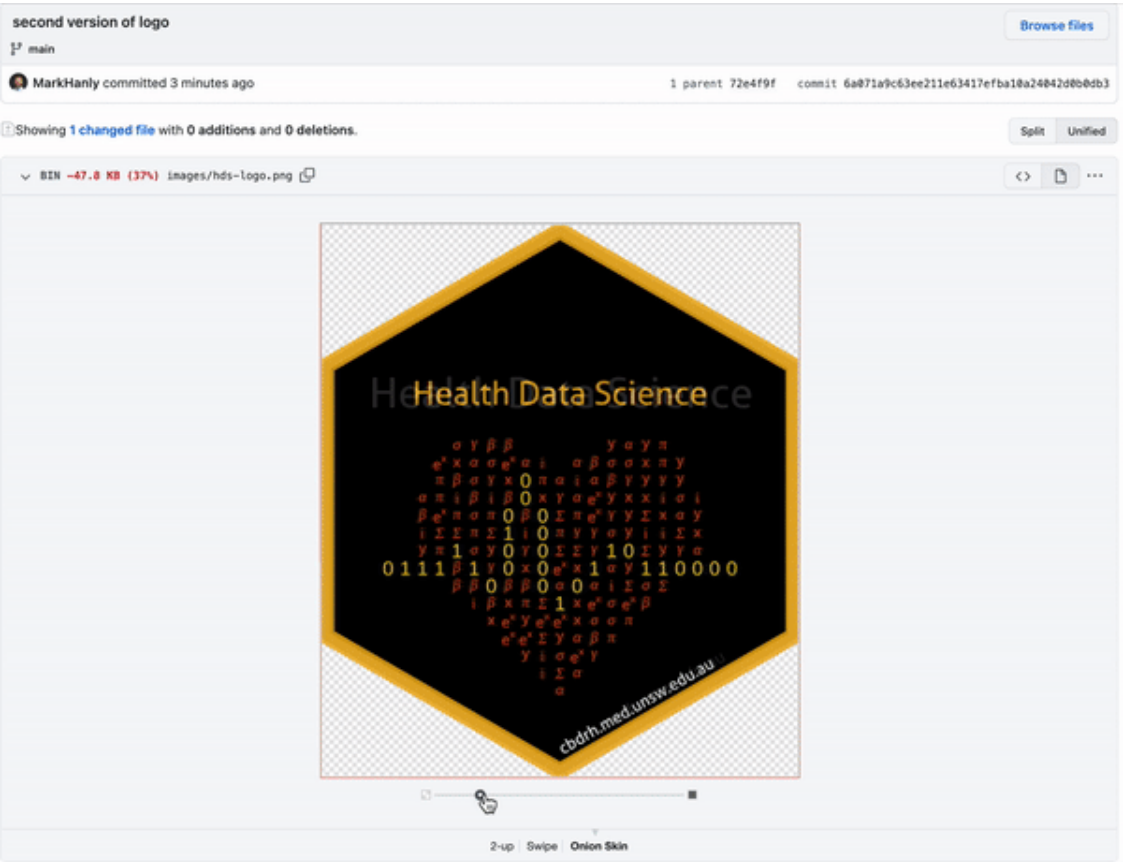
https://github.com/CBDRH/vaccineQueueNetworks

Let's check out an example

You can even compare difference versions of images!

This works for PNG, JPG, GIF, and PSD

You can try this example here



Are you in!?

Via GIPHY

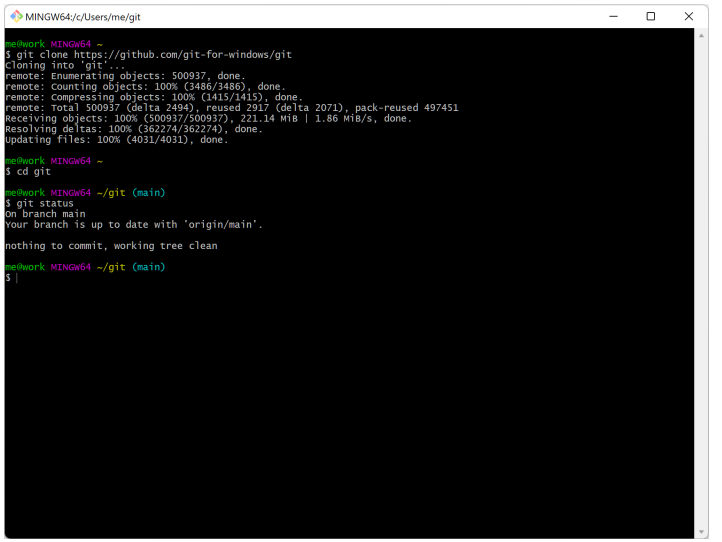
Interfacing with Git

Working locally: Git client

To use Git on your computer you need a *Git Client*. There are heaps of git clients available, with different pros and cons. Here are a few examples:

1. Command line

- Terminal (Mac)
- Git bash (Windows)
- PowerShell (Windows)



Via gitforwindows.org

Downloads these from:

git website

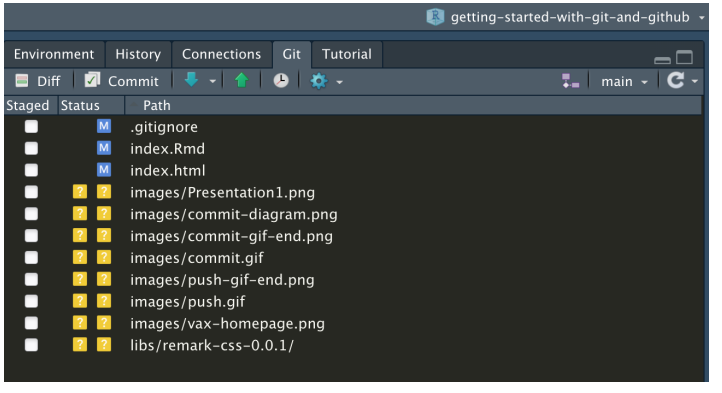
git for windows

Working locally: Git client

To use Git on your computer you need a *Git Client*. There are heaps of git clients available, with different pros and cons. Here are a few examples:

2. R Studio interface

- For R users!
- Check out happygitwithr.com

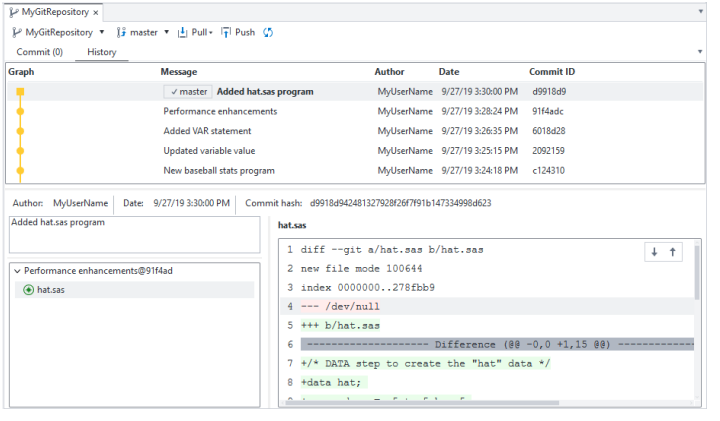


Working locally: Git client

To use Git on your computer you need a *Git Client*. There are heaps of git clients available, with different pros and cons. Here are a few examples:

3. SAS Enterprise Guide 8 interface

- For SAS users!
- Understanding Git Integration in SAS Enterprise Guide
- Similar integration in SAS Studio



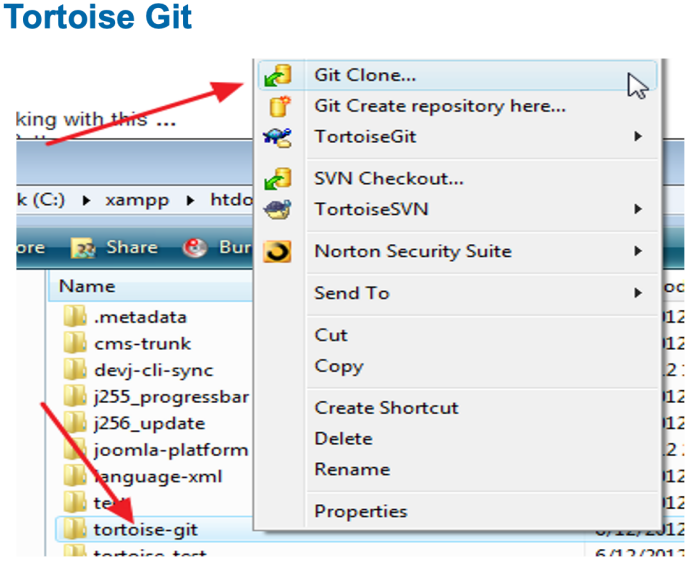
Via documentation.sas.com

Working locally: Git client

To use Git on your computer you need a *Git Client*. There are heaps of git clients available, with different pros and cons. Here are a few examples:

4. Tortoise Git

- Windows integration to file explorer
- If you want to point and click
- https://tortoisegit.org/



Collaboration: Git hosting service

To enable collaboration you need a *Git Hosting service*. There are heaps of options with different pros and cons, but the core functionality is usually the same.

GitHub

Probably the most popular hosting service so there are lots of tutorials and resources available online.

Via logos-world.net

Collaboration: Git hosting service

To enable collaboration you need a *Git Hosting service*. There are heaps of options with different pros and cons, but the core functionality is usually the same.

Gitea

A light-weight easy to install option.

Implemented in ERICA.

Via vectorlogo.zone

Collaboration: Git hosting service

To enable collaboration you need a *Git Hosting service*. There are heaps of options with different pros and cons, but the core functionality is usually the same.

GitLab

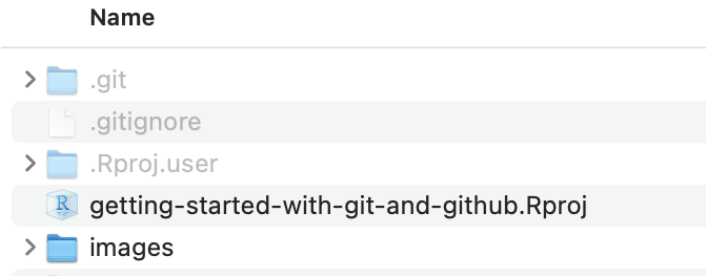
Can be hosted on private server so ideal for sensitive projects.

Implemented in SURE.

Via wikimedia.org

Key concepts

Key concept: Repository



A repository is where the magic happens 🧙

- Referred to as a repo
- Database of changes to your code (aka *diffs*)
- Hidden folder on your system named **.git**

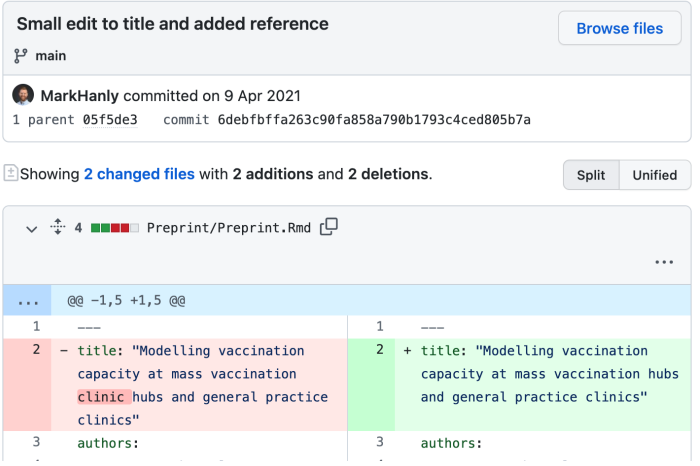
Once you initialise a git repo in a folder, git is ready to start tracking changes to all the files and subfolders therein.

Key concept: Commit

Commit

- A Commit is like clicking save in a MS Word document
- It saves a snapshot of the code at that point in time
- Why you commit you must write a message explaining what changed
- Committing is a two stage process - more on this later

Key concept: Diff

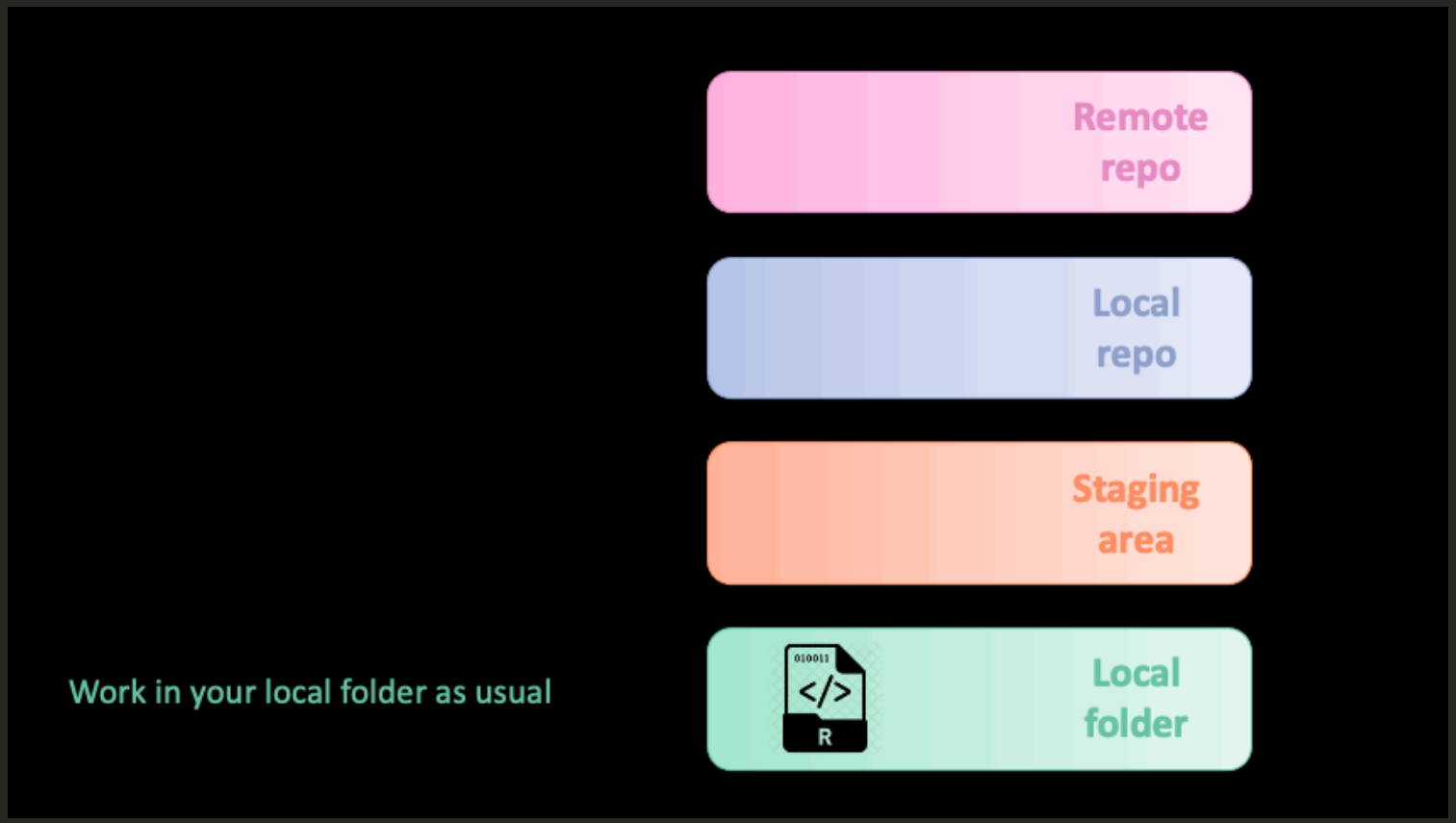


Diff

- A diff is the set of changes between two versions of a file
- When you commit, it is the *diff* that gets recorded, not the whole version
- The screenshot highlights a single diff: deletion of the word "clinic"

Workflow

Workflow


Remote repo
Local repo
Staging area
Local folder
Work in your local folder as usual

Workflow

Push files to remote repo
`git push`

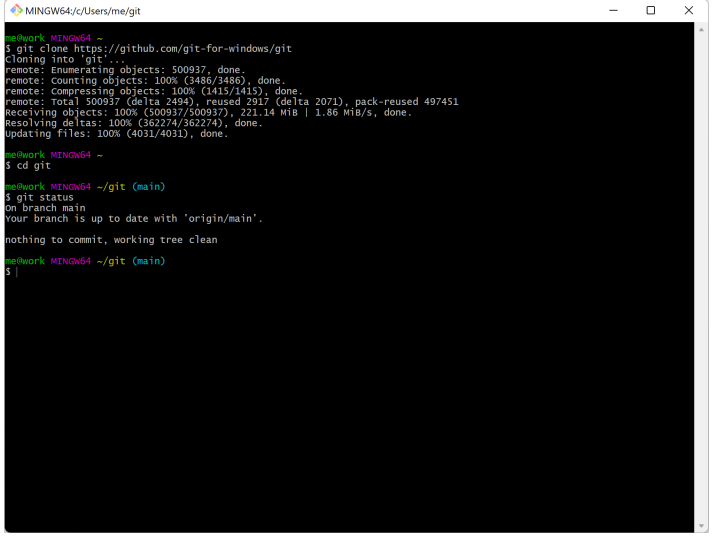
Commit files to local repo
`git commit –m "message"`

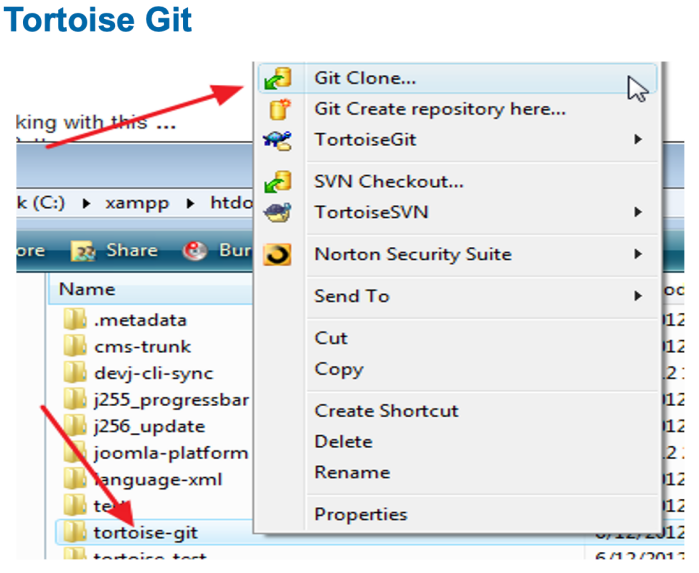
Add files to the staging area
`git add -A`

Work in your local folder as usual

Workflow

How you actual perform these commands depends on the git client you are using...





For today's exercises we will be using *the scary black box*.

Frequently asked questions

Frequently asked questions

What files can I track?

This all works best for non-binary or text files e.g. your typical code files (.sas, .do, .R, .Rmd, .py) but also .png, .csv etc

You can include non-binary files (.pdf, .docx, .xlsx etc) in a git repo but the diff features won't work. It can still be useful if using the repo as a central orgnaisation point (e.g. if you save your published manuscript with the analysis code).

Frequently asked questions

How often should I commit?

Well, it depends.

Just like when you start writing a word document you might not click save very often as you spew out lots of ideas, but at the end of your thesis you will carefully save evey change!

It is best practice to keep commits fairly small and related, especially for mature projects.

Frequently asked questions

What's with the staging area?

There are advantages of breaking up the commit into two steps

- You can split a big edit into bite-sized pieces
- Easier to review
- If you make a mistake you can *unstage*. You can't un-commit!

Frequently asked questions

What about my sensitive data/code?

If your data are sensitive be very careful not to track with Git (more tips on this later).

Private repos are available if you don't want the public to see your code (you can change to public later if desired.)

Frequently asked questions

How does my repo align with my project folder?

Just like when you create project folders and sub-folders you need to put some thought into how to organise your repos.

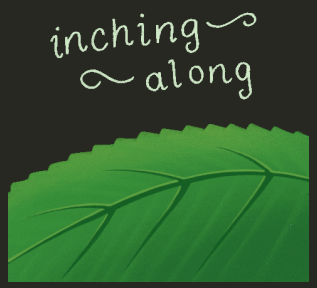
One option is to initiate the repo in the root folder of the project, meaning that all files and subfolders are tracked by default.

An alternative would be to initiate the repo in a specific folder, e.g. your `Code` folder, and keep your data in a separate `Data` folder.

In either approach you can specify any files or folders that Git should ignore in a special file named `.gitignore`.

Any other questions?

Exercises



Via GIPHY

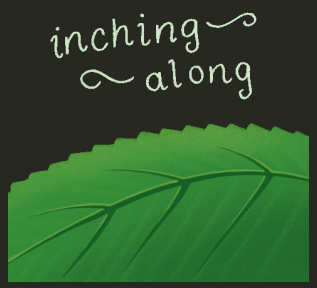
Good luck! 🍀

Overview of exercises

- Exercise 1 - 3: Working locally with Git 🚗
- Exercise 4: Configuring GitHub + Git ✈️
- Exercise 5 - 7: Working with Git and GitHub 🚀

Notes on exercises

- These are not hard but can be... *fiddly*. Don't panic if you get lost. You can always watch the demo and come back to it afterwards.
- At this stage, if something goes wrong it can be easiest to close the shell and start again.



Via GIPHY

Good luck! 🍀

Tips for Success

- **Enter the commands carefully!** **A** versus **a** or **-** versus **--** will make a difference!
- **Pay attention to any messages** git returns. They're often incomprehensible but should flag if something has gone wrong
- You can launch Git Bash/Terminal from your file browser by clicking on a folder and selecting **Git Bash Here** (in windows) or **New Terminal at Folder** (in macOS)
- Use `git status` often to confirm the current state of your repo
- ↹ Use **tab** in Git Bash/Terminal to auto-complete file and folder names
- ↑ Use the up arrow in Git Bash/Terminal to retrieve the last command you entered
- ← → Use the left and right arrows to position your cursor in Git Bash/Terminal (e.g.if you want to edit a typo in a command)
- `ctrl+V` doesn't work to paste into Git Bash, use **insert** or **Right Click -> Edit -> Paste**

Useful shell commands

Command	Action
cd	Change directory (e.g. cd ‘H:/My Documents’)
pwd	Print working directory
ls	List files
mkdir	Create a new directory (e.g. ‘mkdir NewProject’)
touch	Create a new file (e.g. ‘touch mynewlist.txt’)
..	Refers to the folder up one level (e.g. ‘cd ..’)
git init	Initialise a new git repository
git status	Check the status of a Git repository
git add	Add a specific file to the staging area (e.g. ‘git add code.sas’)
git add -A	Add all files with changes to the staging area
git commit -m “msg”	Commit files in the staging area
git pull	Download changes from a remote repository
git push	Upload local commits to a remote repository
git clone	Copy an existing repo from GitHub etc

Exercise 1 of 8 🚗

Task: Practice using shell commands to create a new folder and empty text file with GitBash (Windows) or Terminal (Mac)

Tips

Ctrl+V doesn't work in GitBash, right-click and choose paste instead.

`cd ~` is the home directory

`cd ..` moves you "up" one folder

1. Open GitBash/Terminal. Type `pwd` to confirm the current folder
2. Use the `cd` command (change directory) to navigate to the location you want to create your new folder. e.g. `cd ~/Documents`
3. Use the `mkdir` command to create a new folder named GitDemo. Enter `mkdir GitDemo`
4. Use the `cd` command again to change directory to the new folder. Enter `cd GitDemo`
5. Use the `touch` command to create a new text file named method.txt. Enter `touch method.txt`

Can you see the new folder and file on your computer? 🎈

Exercise 2 of 8 🚗

Task: Configure Git with your username and email

Tips

In Git Bash and Terminal, use the up key ↑ to bring up the last command you entered

Using GitBash or Terminal, configure Git with your username and email by entering the commands below:

1. `git config --global user.name "John Doe”`
2. `git config --global user.email johndoe@example.com`

To check it has worked you can enter `git config --list` .

Do you see your user name and email listed among your configuration settings? 🎉

Exercise 3 of 8 🚗

Task: Initialise a git repo then add and commit changes to your text file

Tips

Enter `git status` after every step to check the current state of your repo!

We want to emulating writing real code so make sure to do the edit → save → add → commit cycle a few times

1. Make sure you are in your new folder: `cd GitDemo`
2. Initialise a git repo: `git init`
3. Check the status of your repo: `git status`
4. Open method.txt with your preferred text editor & write a short list or recipe. Save after adding each item click save and then:
    1. Stage the file: `git add -A`
    2. Commit the file: `git commit -m “your message here”`
5. Enter `git log` to view a log of your changes

Do you see a list of the commit messages you entered? 🎂

Exercise 4 of 8 🚀

Create a personal access token (PAT) on GitHub to allow you to authenticate your account.

Tips

Remember to save your PAT token somewhere safe.

For the PAT scopes, you can just check repo for now.

In step 5 you will push your local repo to GitHub. In order to communicate between git and GitHub you will need to create a special password called a Personal Access Token or PAT. You can create the PAT on GitHub then use it if you are prompted to enter a password on git.

Follow steps 1-9 on **Creating a personal access token** here: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Exercise 5 of 8 🚀

Task: Create an empty repo on GitHub and push your local repo to this empty GitHub repo

Tips

Remember, ctrl+c and ctrl+v wont work in the shell. You can use `insert` or right-click and choose paste.

1. Log in at github.com and create a new public repo (use + button in top right-hand corner)
2. Give the GitHub repo the same name as your project folder
3. Ignore the options to initialize a README, add a .gitignore or license
4. When you are ready select **Create Repository**
5. When you create the blank repo, three options will appear. Follow the instructions labelled **to push an existing repository from the command line**. Copy and run the three lines of code one-by-one into Git Bash/Terminal.
6. If prompted, enter your GitHUb username and the PAT you created in the previous step.

Exercise 6 of 8 🚀

Task: Clone an existing GitHub repo to your local computer

Tips

If your current working directory is GitDemo then `cd ..` will change your directory to the parent folder.

If you clone the new repo here then `GitDemo` and `getting-started-practice` will be side-by-side

1. Go to https://github.com/MarkHanly/getting-started-practice
2. Click on the Code button and copy the HTTPS url to your clipboard.
3. Using GitBash/Terminal, use `cd` to navigate to a location you want to clone this practice repo. **Don't clone it into the repo you created earlier!**
4. Enter `git clone https://github.com/MarkHanly/getting-started-practice.git` to clone the practice repo locally (this is the url you've just copied so you can paste it in)

Can you see the `getting-started-practice` folder copied locally? 🎈

Exercise 7 of 8 🚀

Task: Create a new branch, add a new file and push your branch to GitHub

Tips

Replace `yourmame` with your actual name e.g. `mhanly`.

For the `push` to work you need to be added as a collaborator on the GitHub repo. If you haven't already, send me your GitHub username!

1. Navigate to the newly cloned repo: `cd getting-started-practice`
2. Create a new branch (name the branch with your name so that it is unique): `git checkout -b yourname-branch`
3. Add a new .txt file (name the file with your name so that it is unique): `touch yourname-edits.txt`
4. Open the new .txt file in the usual way, write a line or two and click save.
5. Confirm you have edited the file: `git status`
6. Use `git add -A` and `git commit -m "message"` as before to add and commit changes
7. Push your local changes to the GitHub repo `git push --set-upstream origin yourname-branch`

Can you see your branch listed under https://github.com/MarkHanly/getting-started-practice/branches ? 🎉

Exercise 8 of 8 🌕

Task: Submit a pull request to merge your branch into the main branch on GitHub

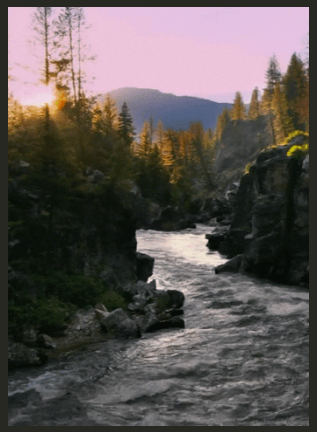
Tips

To merge a branch on GitHub we submit a "pull request". This is different to `git pull` which pulls a remote repo to a local repo.

1. Go to https://github.com/MarkHanly/getting-started-practice/branches
2. You should see you branch listed under active branches. Click New pull request.
3. Click Create pull request
4. You should see a note to say "This branch has no conflicts with the base branch". Click Merge pull request and then Confirm merge.

Can you see your file listed under the main branch at https://github.com/MarkHanly/getting-started-practice? 🎉

Where to go next?



Via GIPHY

See the References and Resources slide for suggestions on where to go next.

Git and Github are powerful tools with numerous features. We have only scraped the surface, focussing on the simplest elements of an individual workflow but the real benefits come with collaboration.

Collaborating effectively requires additional steps that haven't been covered in these intro slides:

branching | spawn a new branch to develop new features

merging | combine separate branches back together

pull requests | request your feature branch to be integrated into the code base

References and resources

High level (convince yourself and your team)

- Jennifer Bryan (2018) Excuse Me, Do You Have a Moment to Talk About Version Control?, The American Statistician, 72:1, 20-27
- Kieran Healy (2019) The Plain Person's Guide to Plain Text Social Science

Software focus

- Happy Git and GitHub for the useR by Jennifer Bryan
- Understanding Git Integration in SAS Enterprise Guide SAS documentation
- Stata and GitHub Integration A medium.com blogpost by Asjad Naqvi

Comprehensive references

- Atlassian tutorials (These are excellent but focus on git+Bitbucket rather than git+GitHub)
- Chacon, S., & Straub, B. (2014). Pro git Springer Nature. (available online)
- GitHub docs

Thank you for taking part! 🙏

If this was useful for you, or if you have any questions or suggestions, please let me know at m.hanly@unsw.edu.au 🙂

Slides created via the R package **xaringan** and hosted on **netlify**. Thank you to **Dr Malcolm Gillies** for input on these slides 🚀

Feel free to share, the url is getting-started-with-git-and-github.netlify.app.

The source code for these slides is available at github.com/MarkHanly/getting-started-with-git-and-github. Issues can be added here.