STAT1378: Coding and Communication in Statistics

Topic 6: Version Control with Git and Github

Contents

- Version Control with Git and Github
  - `praiseme`

Acknowledgement

- The content in this week's notes are heavily inspired by the following workshops and publication:
  - Happy Git and GitHub for the useR by Jenny Bryan, Jim Hester & others. `https://happygitwithr.com`.
  - rOpenSci Ozunconf 2019 day 0 pre-ozunconf training, by Nicholas Tierney & Saras Windecker
  - R Packages (2 ed.) by Hadley Wickham & Jenny Bryan. `https://r-pkgs.org/index.html`

Before we begin

- We assume you have followed Topic 05 notes and created your own copy of the `praiseme` package.
- If you haven't done so, please go back and finish that task first.

What is Git?

- Git is a version control system.
- It tracks the evolution of a set of files – called a *repository*.
- Consider it as the *Track Changes* features from Microsoft Word but on steroids.

What is Github?

- Github is a hosting service.
  - There are others like Bitbucket, and GitLab.
- They provide a home for your Git-based projects on the net.
- It allows other people to see your stuff, sync up with you, and perhaps even make changes.
- Think of it as *DropBox* but again on steroids.

Why Git + GitHub?

- Have your ever tried to collaboratively write code or work on a project with someone by sending files back and forth via email or a Dropbox folder?
  - With Git, both of you can work on the same file at the same time.
  - Git will either combine your work/changes automatically, or it will show you all the ambiguities/conflicts.
  - Think of it as Google Doc but again on steroids.
- It makes sharing your package easy. Any `R` user can install your package from Github with just one line of code:

```
# install.packages("devtools")
devtools::install_github("username/packagename")
```

Why Git + GitHub? (cont.)

- GitHub is a great way to make a basic website for your package.
  - Users can easily browse code, and read documentation (via Markdown).
  - It provides a platform to for users to report bugs, suggest new features (with GitHub issues), and propose improvements (with pull requests).
- Have you ever accidentally pressed s instead of Cmd + S to save your file and introduced a bug?
  - It's very easy to accidentally introduce a bug that takes a few minutes to track down
- For the same reason, `Git` highlights all changes and that allows you to see and review exactly what's changed (and undo any mistakes).

Why Git + GitHub? (cont.)

- Is it worth it if I am working on a `private` solo project?
  - It's always a good idea to save your work at a remote location.
  - It also encourages a work flow of ongoing development of your package with `issues` and `pull request`.
- Is it easy to learn? No, not really.
  - There are plenty of strange terminology and unhelpful error messages.
  - But we are just giving you an introduction here and the discussion will be limited to what RStudio has to offer.

MACQUARIE University
SYDNEY·AUSTRALIA

Setup `Git` and `Github`

Setup: Part 1 Install `Git`

1) Install `Git`:
   - Windows: `https://git-scm.com/download/win`.
   - MacOS: `https://git-scm.com/download/mac`.
     - You will need the Xcode command line tools. (See Topic 05 for details)
2) Tell `Git` your name and email address. You can either:
   - use `usethis` within RStudio (recommended)

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

   - or run the following code **in the shell**,

```
git config --global user.name "YOUR FULL NAME"
git config --global user.email "YOUR EMAIL ADDRESS"
```

Setup: Part 1 Install `Git` (cont.)

- When you think you are all done, you can check your setup **in the shell**:

```
git config --global --list
which git
git --version
```

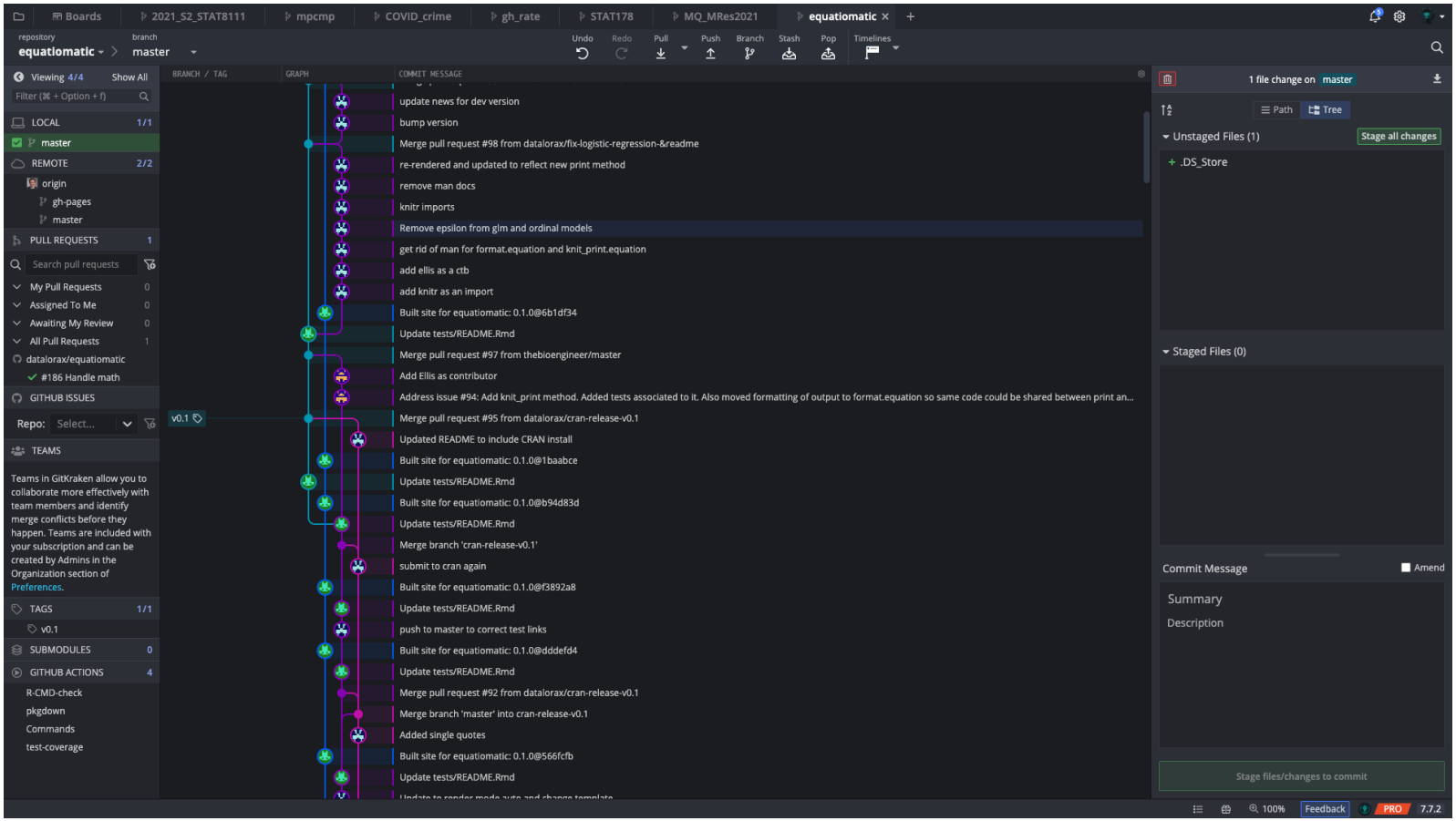
- If you are struggling on Windows, you should check this out: `https://happygitwithr.com/shell.html#windows-shell-hell`.

Setup: Part 2 Create an account on `Github`

- Create an account on GitHub: `https://github.com`.
- Use the same email address as above.
- You might want to give that some thought about your username.
- The free plan is fine but you should definitely upgrade it to an education account: `https://education.github.com`.

Setup: Part 3 (optional) Download a `Git` client

- Download a `Git` client.
  - I recommend `GitKarken`.
  - You can get the **pro** version for free with the Github Eduction account.



Setup: Part 4 Link RStudio and Github

- We then need RStudio and Github to talk to each other.
- The **current** recommendation is to setup a **personal access token** (PAT).
- GitHub offers instructions for this: here
- But because we are using RStudio, there are some helper functions/packages to manage these tasks.
- First run this to take you to the web to create a PAT:

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

- Then store your credentials with R:

```
# install.packages("gitcreds")
gitcreds::gitcreds_set()
# ? Enter password or token:
```

Setup: Part 4 Link RStudio and Github (cont.)

- Follow the prompt with your PAT token.
- When you are done, here are two great ways to check that all is well:

```
gh::gh_whoami()
```

```
# {
#   "name": "Thomas Fung",
#   "login": "thomas-fung",
#   "html_url": "https://github.com/thomas-fung",
#   "scopes": "admin:enterprise, admin:gpg_key, admin:org, admin
#   "token": "ghp_...jeZR"
# }
```

Setup: Part 4 Link RStudio and Github (cont.)

```
usethis::git_sitrep()
```

```
# Git config (global)

# * Name: 'Thomas Fung'

# * Email: 'thomas.fung.dr@gmail.com'

# * Vaccinated: TRUE

# i Defaulting to 'https' Git protocol

# * Default Git protocol: 'https'

# GitHub

# * Default GitHub host: 'https://github.com'

# * Personal access token for 'https://github.com': '<discovered>'

# * GitHub user: 'thomas-fung'

# * Token scopes: 'admin:enterprise, admin:gpg_key, admin:org, admin:org_hook,

# * Email(s): 'thomas.fung@mq.edu.au', 'thomas.fung.dr@gmail.com (primary)'
```

Setup: Part 4 Link RStudio and Github (cont.)

- Alternatively, you can setup a SSH key.
- You can read more about that method here:
  - `https://happygitwithr.com/ssh-keys.html` or
  - `https://r-pkgs.org/git.html#git-setup`

MACQUARIE University
SYDNEY·AUSTRALIA

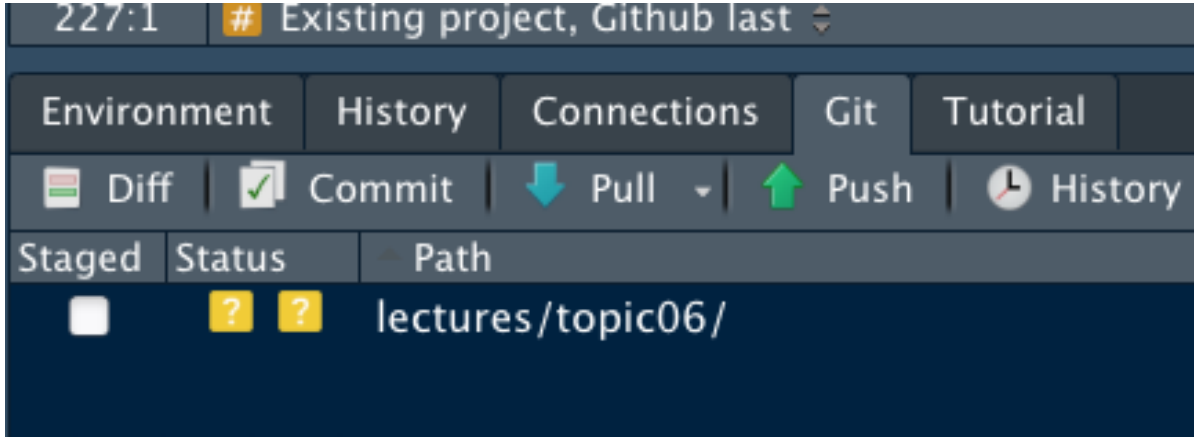
Getting started with `Github`

Getting started with `Github`

- Existing project, Github last
- New project, Github first.
- For other combination, you can read it here: `https://happygitwithr.com/usage-intro.html`.

Existing project, Github last

- In this workflow, we assumed you have an existing R project on your computer.
- The aim is to get an additional `Git` pane:



- If you do, you are already there!

Existing project, Github last (cont.)

- Otherwise, you have a few options:

1) Use the `usethis` package

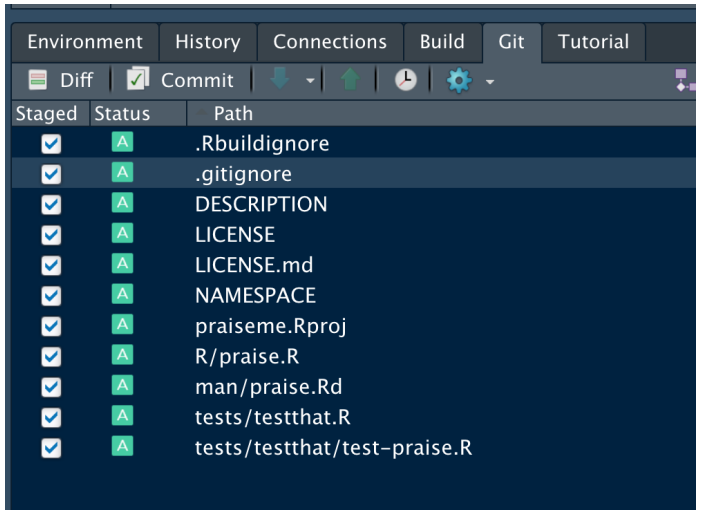
```
usethis::use_git()
```

2) In `RStduio`, to to Tools -> Project Options -> Git/SVN. Under `Version control system`, select `Git`.

- Select `Yes` if confirmation is needed for creating a new `Git` repo.
- At this stage, `RStudio` may need to restart and it should now have a `Git` pane.

Existing project, Github last (cont.)

- The next step is to **stage** and **commit**.
- But before then, let's review what's changed, which is the first benefit that `Git` brings.

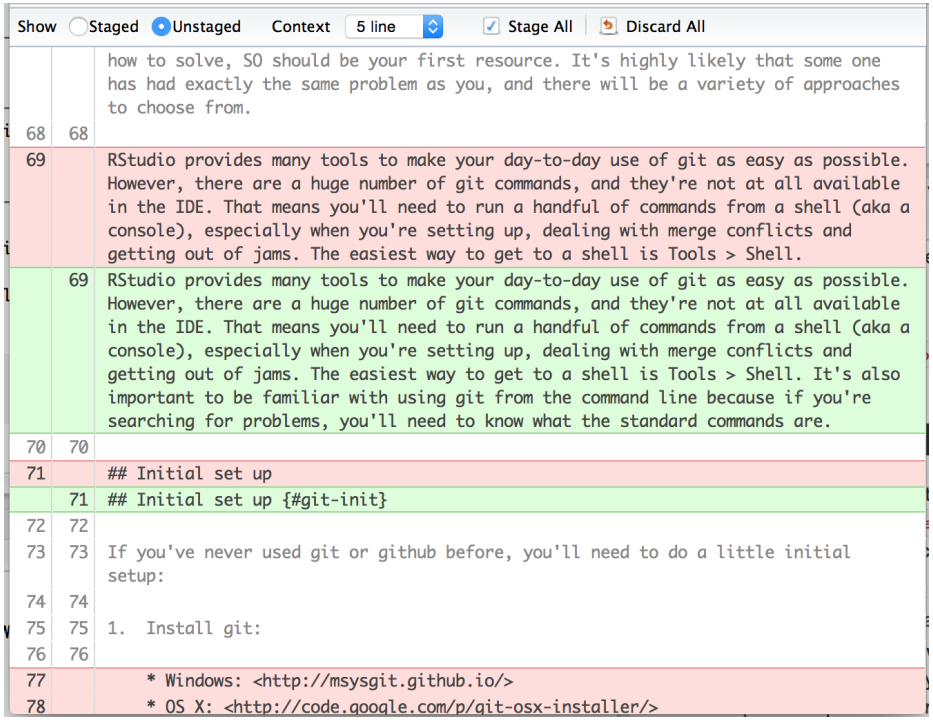


Existing project, Github last (cont.)

- Within `RStduio` Git pane, the icon represents:
  - **A**: Added. You added a new file
  - **M**: Modified. You've change the contents of the file
  - **D**: Deleted. You've deleted that file.
  - **?**: Untracked. You've added a new file that `Git` hasn't seen before.
  - **R**: Renamed. You've rename a file
- You can get more details about the changes with a `diff`.

Existing project, Github last (cont.)

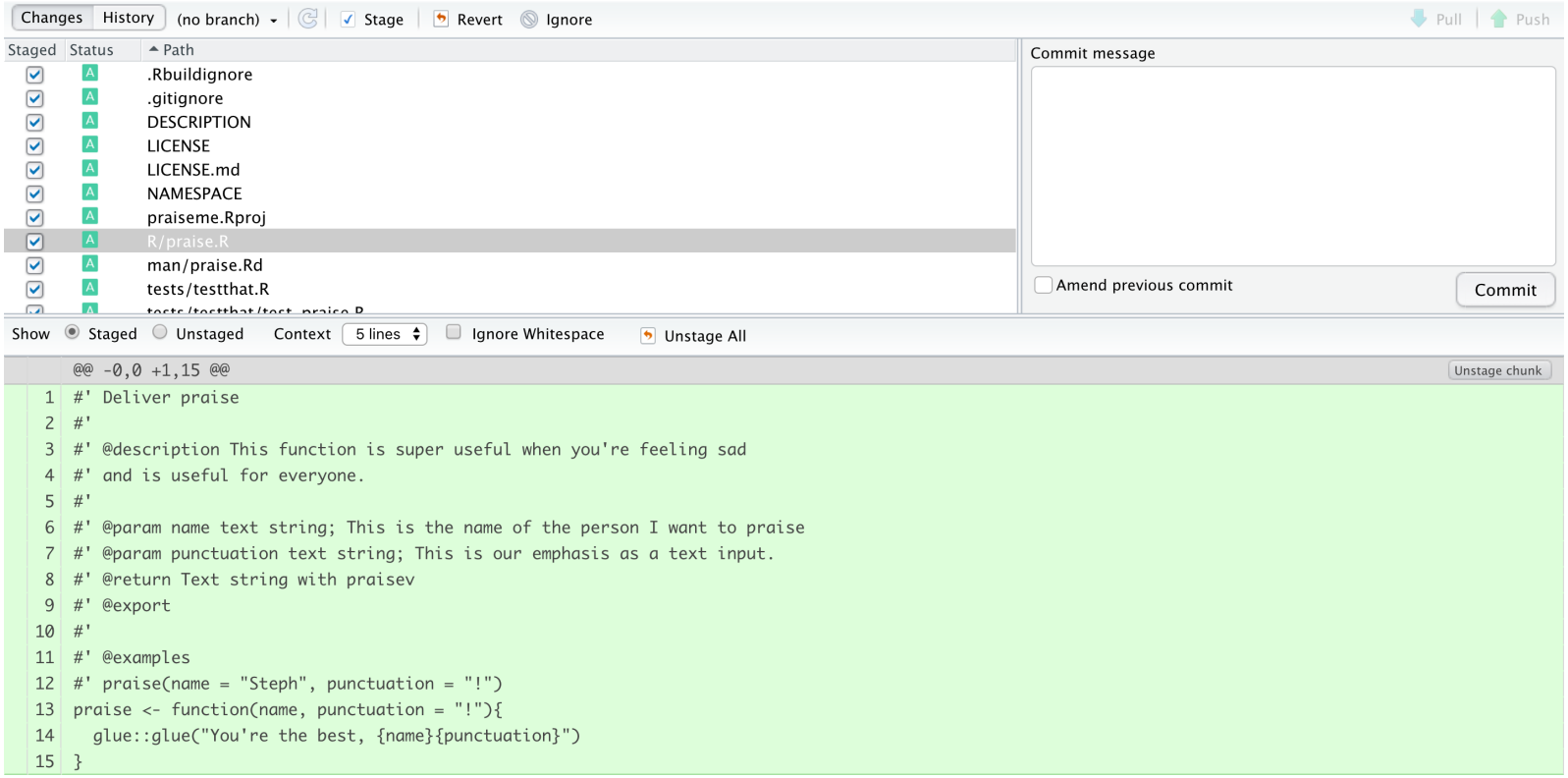
- If you click on the `diff` button, a new window should pop up.



- The background colours tells you whether the text has been added (green) or removed (red).

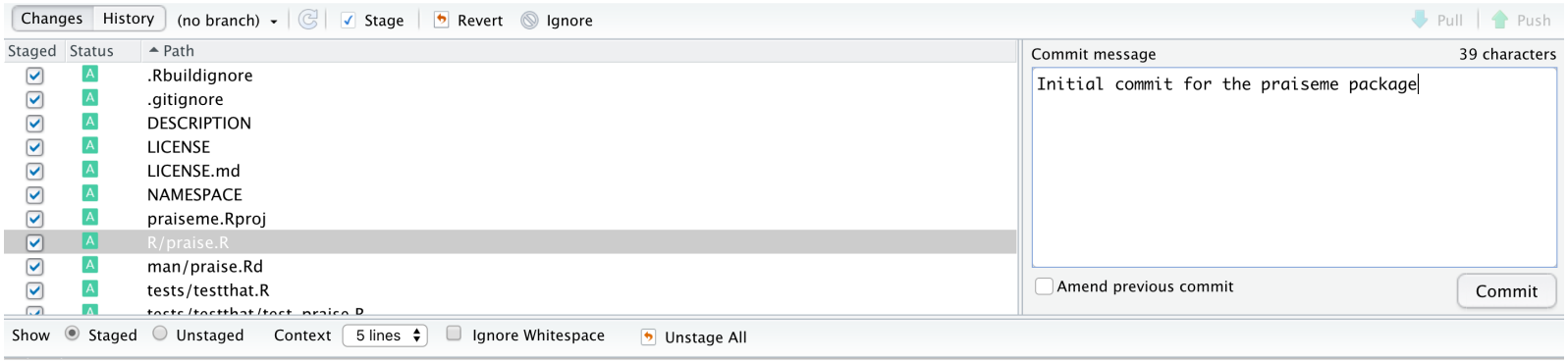
Existing project, Github last (cont.)

- Now let's create a **commit**.
- If you haven't clicked on the `diff` button, you can open the same window by clicking the `commit` button.



Existing project, Github last (cont.)

- The commit window is made up of three panels:
  - The top-left panel shows the current status of your files
  - The bottom panel shows the `diff` of the currently selected file.
  - The top-right panel is where you'll enter a message to be attached to your commit.
- To create a new **commit**
  1) Select the files for inclusion and click `Stage`.
  2) Write an appropriate message.
  3) Click `Commit` and you should see a pop-up window confirming that.



Existing project, Github last (cont.)

- Now we are ready to make and connect a repo on `Github`.
- Assuming you followed our advice and configured a PAT, we can do this with `usethis`

```
usethis::use_github()
```

```
> usethis::use_github()
ℹ Defaulting to 'https' Git protocol
✓ Setting active project to '/Users/thomasfung/Desktop/praiseme'
✓ Checking that current branch is default branch ('master')
Error: Repo 'thomas-fung/praiseme' already exists on 'github.com'
> usethis::use_github()
✓ Checking that current branch is default branch ('master')
✓ Creating GitHub repository 'thomas-fung/praiseme'
✓ Setting remote 'origin' to 'https://github.com/thomas-fung/praiseme.git'
✓ Setting URL field in DESCRIPTION to 'https://github.com/thomas-fung/praiseme'
✓ Setting BugReports field in DESCRIPTION to 'https://github.com/thomas-fung/praiseme/issues'
There is 1 uncommitted file:
* 'DESCRIPTION'
Is it ok to commit it?

1: Yes
2: Nope
3: No way

Selection: 1
✓ Adding files
✓ Making a commit with message 'Add GitHub links to DESCRIPTION'
✓ Pushing 'master' branch to GitHub and setting 'origin/master' as upstream branch
✓ Opening URL 'https://github.com/thomas-fung/praiseme'
>
```

- All done! Congratulation in publishing your first repo on Github!

Existing project, Github last (cont.)

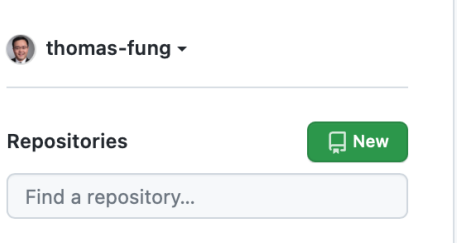
- Alternatively, you can create a repo on Github and then attach the info to your project.
- More information is available here: `https://happygitwithr.com/existing-github-last.html#make-a-new-repo-on-github`.

New project, Github first

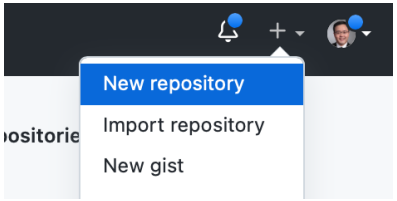
- Do this once per new project.

1) Go to `https://github.com` and make sure you are logged in.

2) Click the green `New` (repository) button.

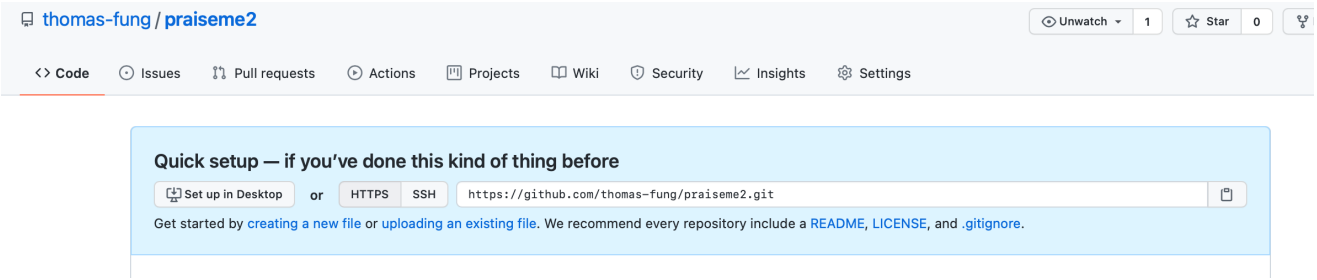


Or, click on the + sign -> `New repository`



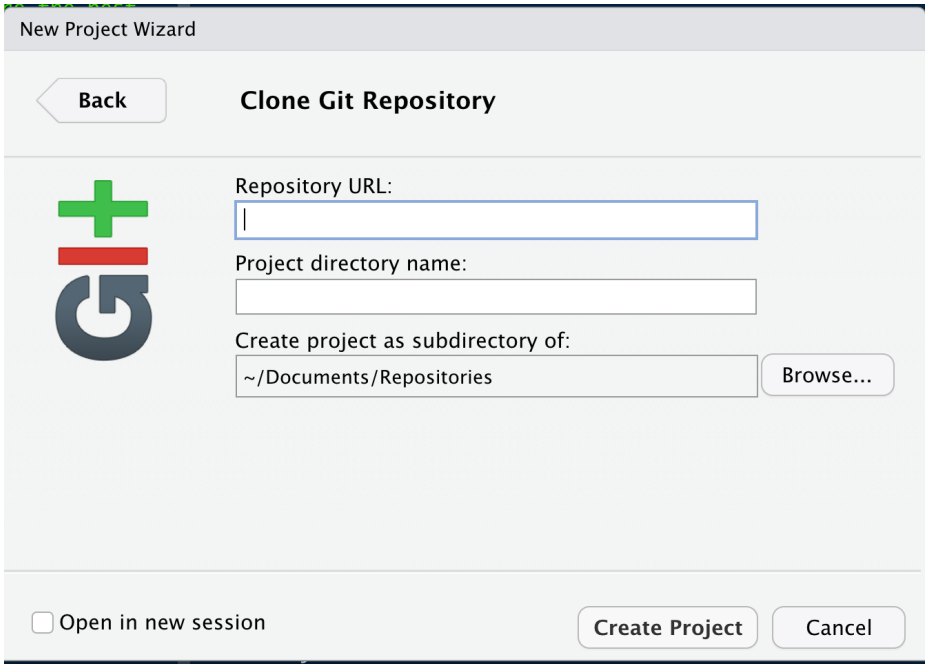
New project, Github first (cont.)

3) Then enter some basic info of your repository:
   - Repository name: myrepo (or whatever you wish)
   - Public
4) Click the big green button `Create repository` at the bottom.
5) Copy the URL to your clipboard.



6) Head to RStudio and create a new project -> Version Control -> Git

New project, Github first (cont.)



7) Paste the URL in, select where you want the project to be created and click `Create Project`.

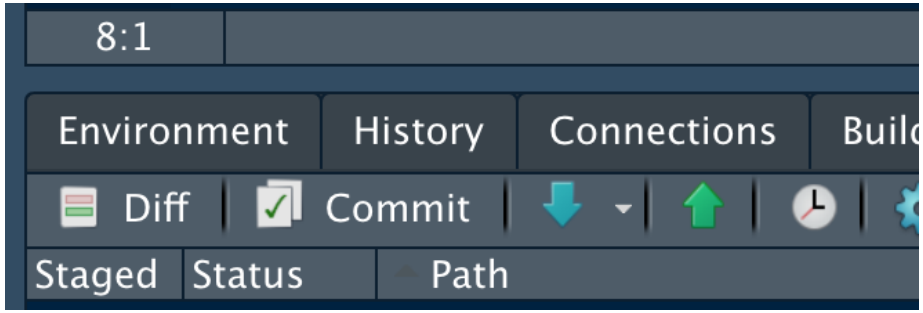
8) Go ahead and create your package as before.

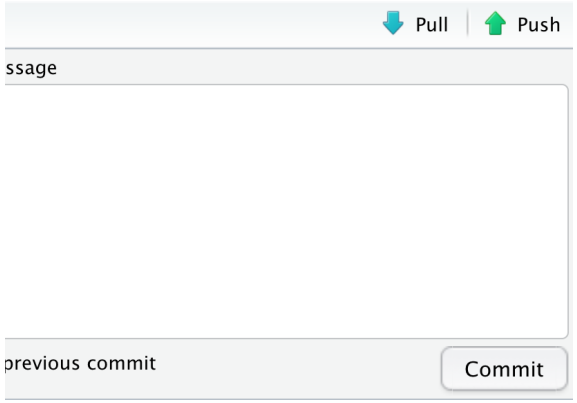
Push your local changes to `Github`

- Suppose you have made some further changes in your local repository, but those changes are not online yet.
- To send your commits to `Github`, we call that a **push**.
- This will seem counterintuitive, but you should **pull** from GitHub first.
- To ask `Github` to send you the latest changes, we call that a **pull**.
  - For a **pull**, `Git` first downloads all of the changes and then merges them with the changes you've made.
  - If changes are made to the same place in a file, you'll need to resolve the **merge conflict** yourself.
  - `RStudio` currently doesn't provide any tools to help with merge conflicts, you are recommended to use a client for this.

Push your local changes to `Github` (cont.)

- To do a pull, click the blue `Pull` button in the `Git` pane in RStudio.





- (Commit your changes if necessarily). Click the green `Push` button to send your local changes to `GitHub`.

Create `README`

- To test the **PUSH** feature, we need to make further changes.
- We will create a `README.md` file, where `md` stands for markdown.
- `Github` renders this `README.md` to be the landing page of your repo.
- But we will generate programmatically from `README.rmd`.
- The difference between `.rmd` and `.md` is that it can handle R code in `.rmd`. More details from next topic.

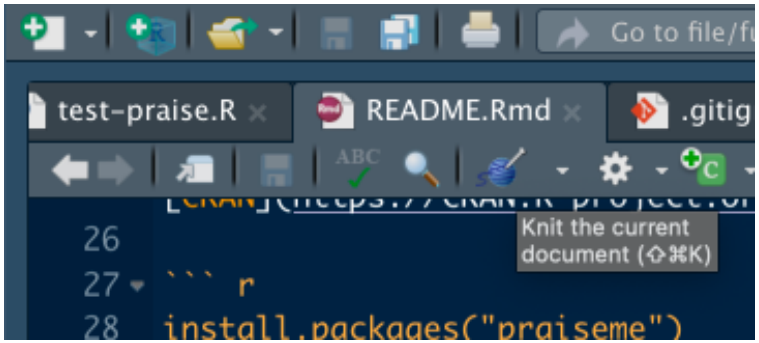
```
usethis::use_readme_rmd()
```

```
> usethis::use_readme_rmd()
✓ Writing 'README.Rmd'
✓ Adding '^README\\.Rmd$' to '.Rbuildignore'
● Modify 'README.Rmd'
✓ Writing '.git/hooks/pre-commit'
```

Create `README` (cont.)

- As our package is not on CRAN yet, we will have to remove the paragraphs on `Installation` and also the part with `library(praiseme)`.

- We will need to **knit** (aka convert) it to `.md` format.



  - You can use the keyboard short cut: Cmd/Ctrl + shift + k.

- At this stage, `Git` tells us we changed a few files.
  - Feel free to use `diff` to see what've been changed. For instance, `.Rbuildignore`.

- Now stage and commit all the changes and **PUSH** them to Github.

- Go to the `Github` page for your repo and see the difference.

Commit best practices

- Each commit should be minimal but **complete**.
  - Minimal: It should only contain changes related to a single problem.
  - Complete: It should solve the problem that it claims to solve.
- Each commit message should:
  - be concise, yet evocative.
    - You should be able to guess what a commit does at a glance.
  - Describe the why, not the what.
    - People can see the what from the `diff`.

Ignoring files

- Often, there are files that you don't want to include in the repository.
- If you are not using the right setup, you may not want to share some "personal" R files like

```
.Rproj.user
.Rhistory
.RData
.Ruserdata
.Rdata
```

- You may not want to share some system files around like

```
.DS_Store
```

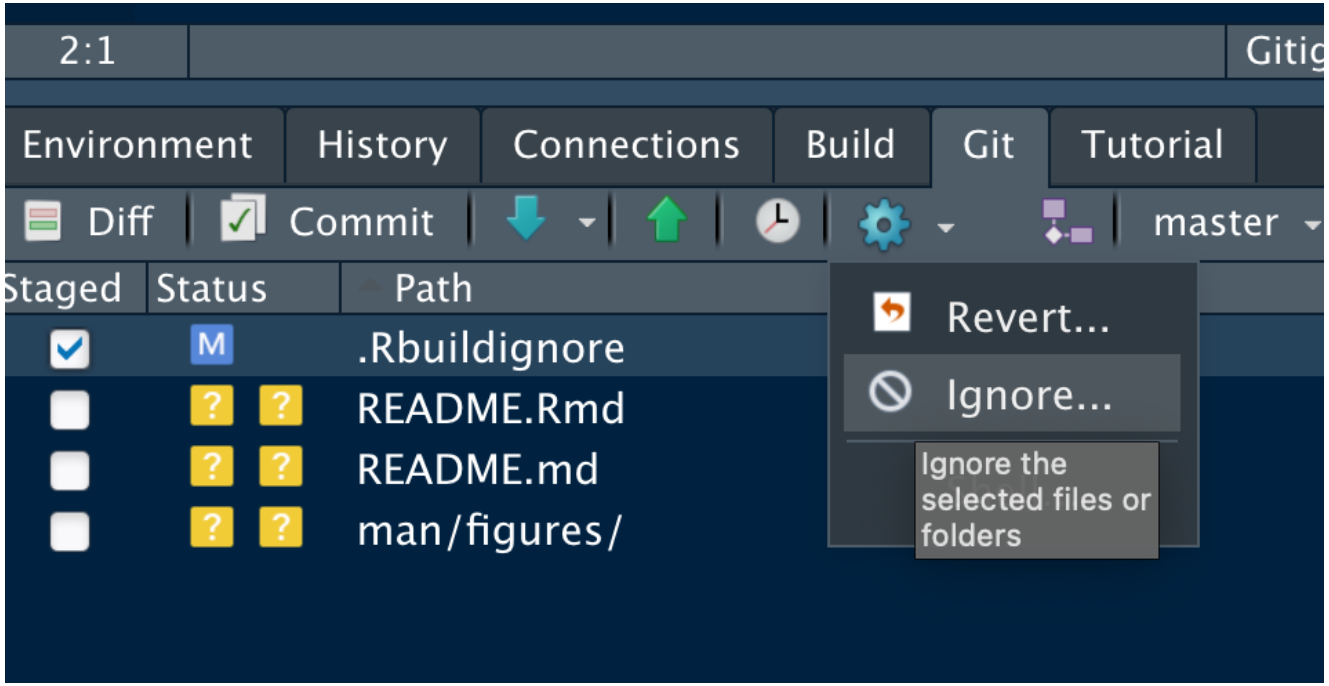
- Instead of carefully not staging them manually each time, you should add them to `.gitignore`

Ignoring files (cont.)

- One way to automate this task is to use `usethis`.

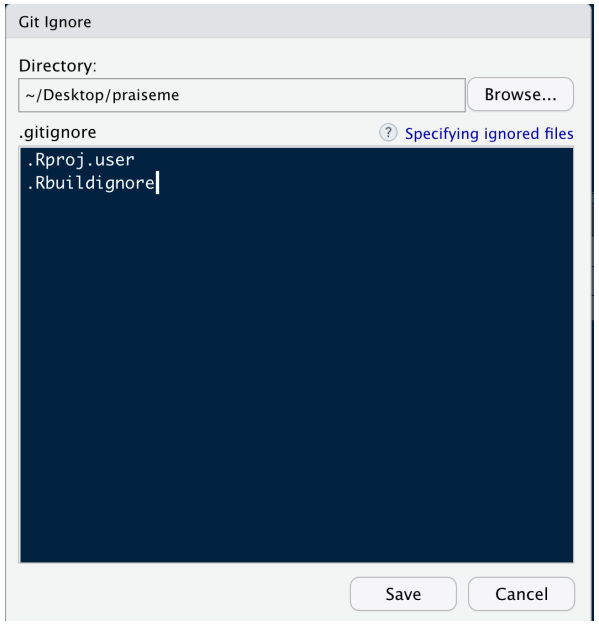
```
usethis::git_vaccinate()
```

- If you want to customise this, there is an `ignore` button in one of the submenu:



Ignoring files (cont.)

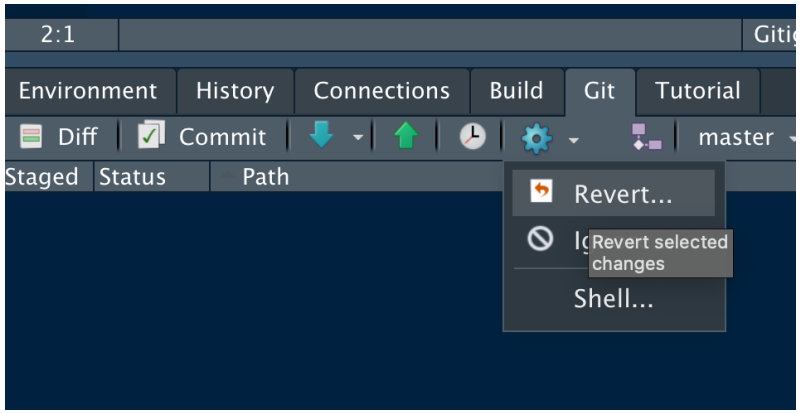
- You can add more file by typing their names in:



- If you want to ignore multiple files, you can also use a wildcard like `*.png`.
- Remember to stage, commit and push your changes!

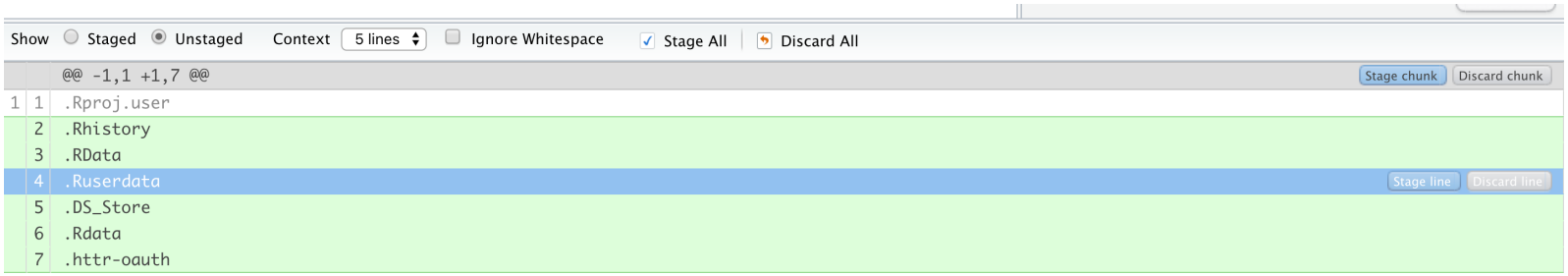
Undoing a mistake

- A commit takes a snapshot of your code.
- Commit at a regular interval (or whenever you are introducing something you are unsure of) to make sure you can go back in time and fix them.
- There is a `Revert` button which will roll back any changes back to the previous commit.
  - Beware: you can't undo this!



Undoing a mistake (cont.)

- You can also undo changes to just part of a file in the `diff` window.



- Sometimes we didn't catch the mistake right away, and we wish to copy the version from the past back to the present. This is called a **checkout**.
  - This is not provided by RStudio and so you will need to either use a client or some shell commands to do that.

What's more

- So far we used Git & Github as we are working on solo project.
- In the SGTA, we will also learn the following tasks:
  - folk a repo
  - create an issues/bug report
  - create a pull request
  - use github action
  - create a simple webpage for our package.

Learning outcomes:

- Version Control with git and Github
  - `praiseme`
- Stage, commit, pull, push